ALGORITMOS DE TRAJETÓRIA CENTRAL PARA PROGRAMAÇÃO LINEAR: IMPLEMENTAÇÃO E TESTES

Niceli Novelo Rigo

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Clóvis Caesar Gonzaga, Ph. D.
(presidente)

Prof. Sérgio Granville, Ph. D.

Prof. Antonio Alberto Fernandes de Oliveira, D. Sc.

RIO DE JANEIRO, RJ – BRASIL
JULHO DE 1991

RIGO, NICELI NOVELO

Algoritmos de Trajetória Central para Programação Linear: Implementação e Testes [Rio de Janeiro] 1991

VIII, 68 p., 29.7 cm, (COPPE/UFRJ, M. Sc., ENGENHARIA DE SISTEMAS E COMPUTAÇÃO, 1991)

TESE – Universidade Federal do Rio de Janeiro, COPPE

1 – Programação Linear 2 – Algoritmos de Trajetória Central

I. COPPE/UFRJ II. Título(Série).

Aos meus pais

Agradecimentos

Agradeço em particular ao professor Clóvis C. Gonzaga pela eficiente orientação, dedicação e incentivo demonstrados ao longo de todo o desenvolvimento do trabalho.

Aos colegas que frequentaram o Laboratório de Sistemas e demais colegas de Otimização pela ajuda e companheirismo durante o desenvolvimento deste trabalho.

Agradeço também aos funcionários técnicos e administrativos do Programa de Engenharia de Sistemas e Computação pela atenção e prestatividade.

Resumo da Tese apresentada à COPPE como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M. Sc.)

Algoritmos de Trajetória Central para Programação Linear: Implementação e Testes

Niceli Novelo Rigo

Maio de 1991

Orientador: Clóvis Caesar Gonzaga
Programa: Engenharia de Sistemas e Computação

Neste trabalho estudamos os Algoritmos de Pontos Interiores para Programação Linear que seguem a trajetória central. A trajetória central é uma curva ao longo da qual o custo decresce e que permanece sempre longe da fronteira do conjunto viável. Os pontos próximos à trajetória central tem importantes propriedades teóricas que apresentamos considerando os métodos primais e primais-duais.

Os algoritmos foram implementados e testados para um conjunto de problemas pequenos gerados aleatóriamente. Através dos testes comparamos os métodos primal e primal-dual, estudamos o efeito da utilização de buscas unidirecionais e bidirecionais e o efeito de diferentes critérios de proximidades dos pontos em relação à trajetória central.

Palavras-chave: Algoritmos de Pontos Interiores, Programação Linear.

Abstract of Thesis presented to COPPE as partial fulfillment of the requirements for the degree of Master of Science (M. Sc.)

Path Following Algorithms for Linear Programming : Implementation and Tests

Niceli Novelo Rigo

July, 1991

Thesis Supervisor: Clóvis Caesar Gonzaga
Department: Programa de Engenharia de Sistemas e Computação

In this work we study the Interior Point Algorithms for Linear Programming that follow the central path. The central path is a curve along which the cost decreases and wich stays always far from the boundary of the feasible set. The points near the central path have important theoretical properties, which we study for primal and primal-dual methods.

The algorithms were implemented and tested for a set of randonly generated small problems. Through the tests we compare the primal and primal-dual methods, study the effect of using one-directional and bidirectional search procedures and the effect of different criterio to estimate the proximity from each point to the central path.

Key Words: Linear Programming, Interior Points Algorithms.

Índice

Capítulo I

INTRODUÇÃO

O Método Simplex de Dantzig [3] sempre se apresentou muito eficiente, não há dúvidas sobre seus bons resultados práticos que o tornam largamente usado. Apesar disso, discute-se muito a respeito da complexidade do mesmo. Temos o PPL de Klee e Minty [14] onde o simplex percorre todos os vértices do conjunto viável para chegar na solução ótima comportando-se com complexidade exponencial. Embora isso não ocorra nos problemas práticos.

A questão teórica para o problema de programação linear foi resolvida por Khachiyan [13] em 1978, com o método de elipsoides que apresenta complexidade polinomial resolvendo o PPL em $O(n^4L)$ operações aritméticas onde L é o número de bits da entrada de dados do problema. Esse método foi considerado teoricamente mais eficiente que o simplex, mas os resultados práticos ficaram muito longe disso e a situação era a seguinte: tínhamos um algoritmo eficiente que não era polinomial e um algoritmo polinomial que não era eficiente.

Em 1984 Karmakar [12] publicou um algoritmo que além de complexidade polinomial de $O(n^{3.5}L)$ operações, foi anunciado como mais eficiente que o Simplex, principalmente para problemas com tamanho superior a mil variáveis. O método de Karmakar trabalha com o interior relativo do conjunto viável e utiliza uma função logarítmica chamada função potencial.

Através de uma simplificação do algoritmo de Karmakar reproduziu-se o algoritmo afim-escala de Dikin [4]. O algoritmo afim-escala pode gerar pontos

muito próximos à fronteira do conjunto viável prejudicando sua eficiência. Já o algoritmo de Karmakar evita se aproximar da fronteira com o uso da função barreira penalizando as variáveis que se aproximam de zero.

Após a publicação do algoritmo de Karmakar muitos pesquisadores começaram a trabalhar nesta área, quando surgiram os algoritmos de trajetória central que se assemelham ao Karmakar pelo uso da função barreira. Tivemos o algoritmo de Renegar [15], o primeiro a conseguir um limitante $O(\sqrt{n}L)$ iterações com $O(n^{3.5}L)$ operações em contraste com o limitamte de $O(nL)$ iterações do método de Karamakar. Vaidya [18] e Gonzaga [7] seguindo a trajetória central conseguiram um limitante de $O(\sqrt{n}L)$ iterações com $O(n^3L)$ operações.

O objetivo deste trabalho é estudar estes algoritmos de trajetória central que tem apresentado excelentes características teóricas através de uma implementeção computacional combinando algumas técnicas que podem melhorar o desempenho do método. Verificaremos o comportamento dos algoritmos pelos resultados obtidos na resolução de problemas de programação linear. Como os testes foram feitos com problemas pequenos não podemos afirmar que os problemas práticos relativamente grandes sejam resolvidos da mesma forma. Acreditamos, entretanto, que o comportamento dos algoritmos testados sejam semelhantes para a resolução de grandes problemas, ou seja, a tendência é que o algoritmo que se comportou com desempenho fraco na resolução de pequenos problemas não melhorará na resolução de grandes problemas. Em uma implementação de grande porte deve ser dada a preferência às metologias que se mostraram eficientes nestes problemas.

No capítulo II mostraremos o problema de programação linear, as hipóteses assumidas para o nosso trabalho e algumas ferramentas importantes como o cálculo de projeções e mudanças de escala. Aí se encontra o relacionamento entre as duas formas diferentes de enunciar problemas (primal e dual).

O capítulo III fala da Trajetória Central propriamente dita. Define centro de um politopo e lista propriedades de pontos próximos de um centro. A contribuição mais importante do capítulo está na definição cuidadosa do conceito "proximidade" de uma maneira útil ao desenvolvimento dos algoritmos.

No capítulo IV apresentaremos os algoritmos que foram implementados. No capítulo V falaremos da implementação, dos problemas testados e dos resultados obtidos e no capítulo VI falaremos das conclusões finais. Apresentaremos agora a lista dos principais símbolos que serão usados neste trabalho.

Notação

$e = [1 \dots 1]^T$

x^{-1}: $1/x_1, \dots, 1/x_n$.

$X = diag(x_1, \dots, x_n)$

$\Re^n_+, \Re^n_{++}$: vetores não negativos e positivos em $\Re^n$.

x, w, z, A, b, c: variáveis e dados do problema primal e dual.

$\bar{w}, \bar{z}, \bar{c}, \bar{A}$: variáveis e dados para o problema após a mudança de escala.

$\hat{x}, \hat{z}, \hat{w}$: soluções ótimas para o problema (P) e (D).

$\hat{v} = c^T x$: valor ótimo.

S, S^0: conjunto viável e seu interior relativo de (P).

R, R^0: conjunto de folgas duais viáveis e seu interior relativo.

$P_A, \tilde{P}_A$: matriz de projeção no $N(A)$ e seu complemento ortogonal.

$c_p = Pc$: projeção de c no espaço do contexto.

$p(\cdot)$: função barreira.

α: multiplicador que determina a penalidade.

$x(\alpha)$: ponto central associado a α.

$f_\alpha(\cdot)$: função penalizada.

$h(x, \alpha)$: direção escala máximo declive de $f_\alpha(\cdot)$ para x.

$\bar{h}(x, \alpha) = X^{-1}h(x, \alpha)$: direção escala máximo declive no ponto e, após a mudança de escala.

$\delta(x, \alpha) = \|\bar{h}(x, \alpha)\|$: medida de proximidade de x a $x(\alpha)$.

Capítulo II

O PROBLEMA DE PROGRAMAÇÃO LINEAR

II.1 Problema Primal

Podemos apresentar um problema de programação linear qualquer no formato padrão do seguinte modo:

(P)

$$\begin{array}{ll} \min & c^T x \\ \text{sujeito a} & Ax = b \\ & x \geq 0 \end{array}$$

onde c, x $\in \Re^n$, b $\in \Re^m$, A é uma matriz de rank máximo $m \times n$ onde $m < n$.

Se a matriz A não for de rank máximo possuindo equações redundantes podemos eliminá-las do problema e isso não vai alterar a solução do problema original.

Caso o nosso problema tenha restrições de desigualdade do tipo $\leq$ somamos uma variável de folga (não negativa) e eliminamos a desigualdade. Se a desigualdade for do tipo $\geq$ esta será eliminada ao subtrairmos uma variável de folga na restrição.

Definição 1.1 : O problema na forma representada acima será chamado problema primal.

O conjunto viável do problema (P) é definido por:

$$S = \{x \in \Re^n | Ax = b, x \geq 0\}$$

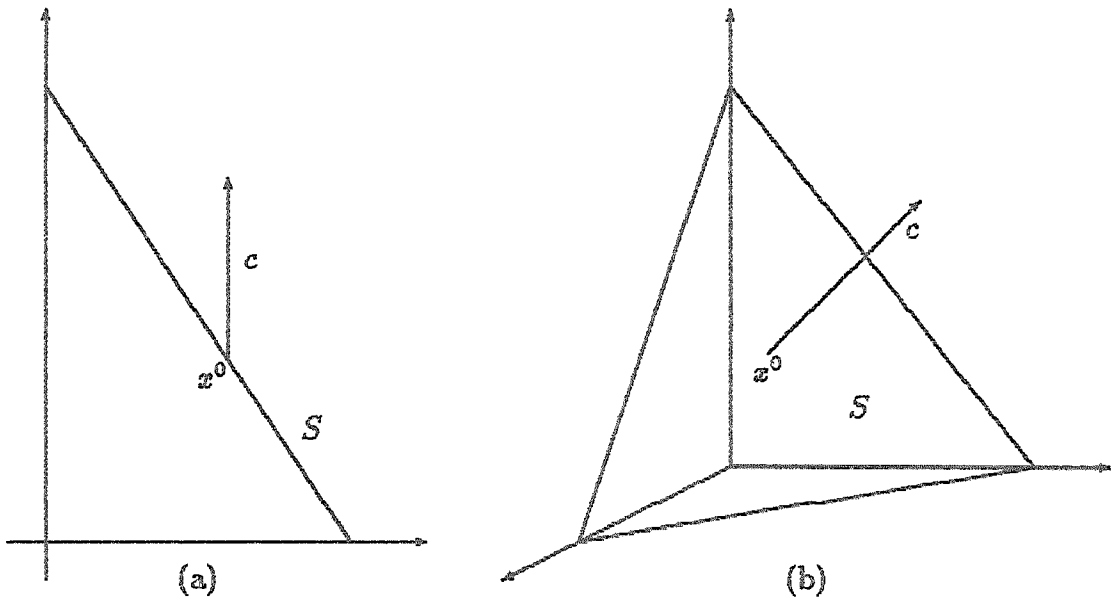

Figura II.1: Em (a) temos m= 1 e n= 2, em (b) temos m= 1 e n= 3 .

A representação geométrica de algumas situações do problema primal para uma melhor visualização do modelo nós vemos na figura II.1.

O conjunto de pontos interiores viáveis de (P) é definido por:

$$S^0 = \{x \in \Re^n | Ax = b, x > 0\}$$

Vamos ver as condições de otimalidade de Karush-Kuhn-Tucker para o problema de programação linear (P).

As restrições de igualdade são $A_i x - b_i = 0$, onde A_i são as linhas de A. As restrições de desigualdade são $-x_i \leq 0$. Aplicando as condições, obtém-se

$$c + \sum_{i=1}^{m} \mu_i A_i^T - \sum_{i=1}^{n} \nu_i I_i = 0$$

$$\nu_i \geq 0 \qquad ,i = 1, \ldots, n$$

$$\sum_{i=1}^{n} \nu_i x_i = 0$$

x *viável*,

onde I_i são as colunas da matriz identidade. Um formato mais interessante é obtido notando que o segundo somatório reduz-se ao vetor $\nu \in \Re^n$, e definindo o vetor $y \in \Re^m$ por $y_i = -\mu_i$. A primeira condição reduz-se a $A^T y + \nu = c$ e a terceira pode ser reescrita como $x^T(A^T y - c) = 0$. Finalmente, unindo as duas primeiras condições, obtém-se $A^T y \leq c$ e as condições reduzem-se a:

$$
\begin{aligned}
& A^T y \leq c \\
& Ax = b \\
& x \geq 0 \\
& x^T(A^T y - c) = 0
\end{aligned}
$$

Para este trabalho serão assumidas as seguintes hipóteses:

- O conjunto viável de (P) é limitado com interior relativo não vazio.
- É dada uma solução inicial x^0 pertencente a S^0 e o problema (P) tem ao menos uma solução ótima $\hat{x}$ com custo ótimo $\hat{v} = c^T \hat{x}$.

II.2 Problema Dual

O problema dual associado a (P) é:

(D)

$$
\begin{aligned}
\max \quad & b^T w \\
\text{sujeito a} \quad & A^T w \leq c
\end{aligned}
$$

onde $w \in \Re^m$ e $A^T \in \Re^{m \times n}$

O conjunto viável do problema (D) é definido por:

$$R = \{w \in \Re^m | A^T w \leq c\}$$

Podemos reescrever o problema (D) obtendo o mesmo problema em $\Re^m$ no formato padrão:

(D2)

$$\begin{array}{ll} \max & b^T w \\ \text{sujeito a} & A^T w + z = c \\ & z \geq 0 \end{array}$$

As variáveis $z \in \Re^n$ chamamos folgas duais tais que $A^T w + z = c$ para algum $w \in \Re^m$, uma folga dual é viável se $z \geq 0$. Com nossas hipótese podemos garantir que (D2) tem uma solução ótima $(\hat{w}, \hat{z})$ (não necessariamente única) e $b^T w = \hat{v}$.

O gap de dualidade é definido da seguinte forma:

$$gap = c^T x - b^T w$$

Lema II.1 *Se $x \in S$ e z é uma folga dual viável então o gap de dualidade será dado por:*

$$x^T z = c^T x - b^T w$$

Demonstração: Do problema (D2) temos

$$z = c - A^T w$$

multiplicando a expressão por x temos

$$x^T z = c^T x - A^T w x$$

como $Ax = b$ obtemos $x^T z = c^T x - b^T w$ completando a demonstração.

II.3 Espaço Imagem, Espaço Nulo e Projeção

O espaço imagem de A corresponde ao conjunto de todas as combinações lineares das colunas de A e é definido por:

$$I(A) = \{y \in \Re^m | y = Ax, x \in \Re^n\}$$

O espaço nulo de A corresponde ao conjunto de pontos $\Re^n$ que são mapeados na origem e é definido por:

$$N(A) = \{x \in \Re^n | Ax = 0\}$$

Uma relação geométrica muito interessante existe entre o $N(A)$ e o espaço imagem da transposta de A, $I(A^T)$: esses dois espaços são sub-espaços ortogonais de $\Re^n$ e geram o espaço todo, como veremos a seguir: Qualquer vetor $c \in \Re^n$ pode ser unicamente decomposto como

$$c = c_p + \tilde{c}_p$$

onde $c_p \in N(A)$ e $\tilde{c}_p \in I(A^T)$, c_p e $\tilde{c}_p$ são respectivamente a projeção de c no espaço nulo de A e seu complemento ortogonal.

A operação da projeção é linear e pode ser representada por uma matriz P_A de modo que $c_p = P_A c$. O complemento ortogonal é $\tilde{c}_p = \tilde{P}_A c$ onde $\tilde{P}_A = I - P_A$.

Se A é uma matriz de rank máximo, então a matriz de projeção de A é:

$$P_A = I - A^T(AA^T)^{-1}A$$

A projeção de c no $N(A)$ é o ponto pertencente ao $N(A)$ com a menor distância euclidiana até c.

$$c_p = argmin_x\{\|x - c\| | x \in N(A)\} \qquad \text{(II.1)}$$

A figura II.2 mostra a geometria de projeção de um determinado problema de programação linear.

Lema II.2 *$z \in \Re^n$ é uma folga dual viável para (D2) se e somente se $z \geq 0$ e*

$$P_A z = P_A c$$

Demonstração: Temos de (D2) que $A^T w + z = c$ onde z é uma folga dual viável então

$$z = c - A^T w$$

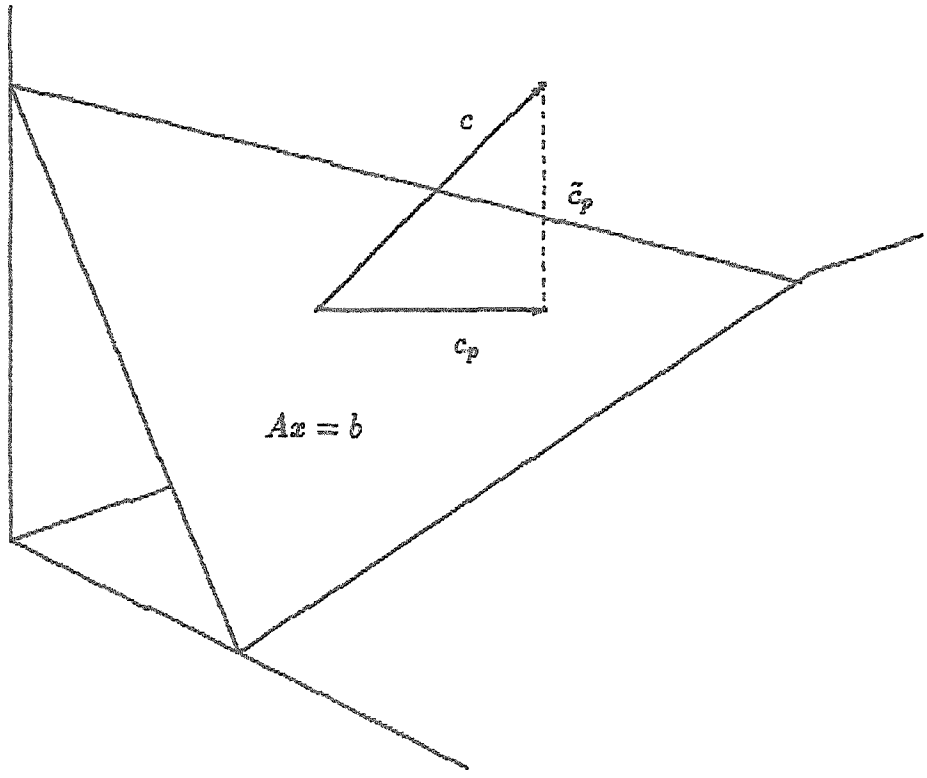

Figura II.2: A geometria da projeção.

multiplicando pela P_A temos

$$P_A z = P_A c - P_A A^T w$$

como $A^T w \in I(A^T)$ que é ortogonal ao $N(A)$, $P_A A^T w = 0$, então obtemos

$$P_A z = P_A c$$

Considere agora $z \in \Re^n$ tal que $z \geq 0$ e $P_A z = P_A c$. Como $P(c - z) = 0$, pois $c - z \in I(A^T)$ então existe algum $w \in \Re^m$ tal que $A^T w = c - z$, z é dual viável completando a demonstração.

II.4 Mudança de Escala

A mudança de escala em torno de $x^k \in S^0$ consiste em uma transformação do problema (P) onde são mudadas as variáveis $x = Xy$ sendo X a matriz diagonal positiva dada por $X = diag(x_1^k, x_2^k, \cdots, x_n^k)$.

Dado um ponto $x^k \in S^0$ o nosso problelma (P) após a mudança de escala será:

$$\begin{array}{ll} \min & c^T X y \\ \text{sujeito a} & AXy = b \\ & Xy \geq 0 \end{array}$$

Fazendo a substituição $\bar{A} = AX$ e $\bar{c} = Xc$ teremos

(SP)

$$\begin{array}{ll} \min & \bar{c}^T y \\ \text{sujeito a} & \bar{A}y = b \\ & y \geq 0 \end{array}$$

É importante observar que a mudança de escala afeta os métodos de primeira-ordem, e é totalmente irrelevante ao comportamento dos métodos de Newton-Raphson. A principal razão para o uso da mesma é que simplifica o tratamento matemático.

Capítulo III

MÉTODOS DE TRAJETÓRIA CENTRAL

III.1 Função Barreira

A função barreira foi primeiramente usada em otimização por Frisch [6] em 1955.

A função barreira $p : \Re^n_{++} \mapsto \Re$ é definida por:

$$p(x) = -\sum_{i=1}^{n} \log x_i$$

e tem derivadas

$$\nabla p(x) = -x^{-1}$$

$$\nabla^2 p(x) = X^{-2}$$

Após a mudança de escala as derivadas no ponto $e = (1, \cdots, 1)$ são

$$\nabla p(e) = -e$$

$$\nabla^2 p(e) = I \qquad \text{(III.1)}$$

Desde que $\nabla^2 p(x)$ é positiva definida em S^0, $p(\cdot)$ é estritamente convexa.

Esta função pode ser usada para penalizar as variáveis que se aproximam de zero, ou seja, para evitar pontos próximos à fronteira. Se uma variável x se aproxima de zero a função cresce indefinidamente sendo que o único ponto que minimiza $p(\cdot)$ em S^0 é chamado centro analítico de S. A figura III.1 mostra as curvas de nível da função barreira.

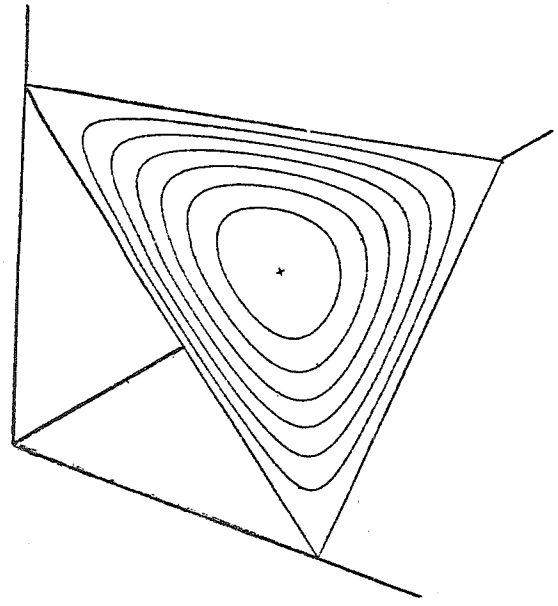

Figura III.1: Curvas de nível da função barreira.

III.2 Algoritmos de Trajetória Central

Um algoritmo de pontos interiores parece funcionar bem se tentarmos gerar uma sequência de pontos reduzindo o custo e ao mesmo tempo permanecendo longe de fronteira de S. Para o algoritmo não gerar pontos muito próximos à fronteira vamos contar com a ajuda da função barreira.

Uma maneira de evitar a fronteira é definir um "centro" do politopo S considerando simultaneamente a redução de custo e centragem. Agora vamos esquecer o problema do custo para tratar do problema de encontrar um "centro" para o politopo S.

Centro

A melhor definição para centro é provavelmente o centro de gravidade, mas encontrar o centro de gravidade é mais difícil que resolver o problema de programação linear. Uma outra maneira de encontrar um centro é construir o maior elipsóide possível inscrito no politopo S e usar o centro do mesmo, o que ainda é muito difícil. Os centros acima são centros geométricos.

O centro analítico definido por Sonnevend [16] é o mais útil no momento. O

centro analítico de S^0 é o único ponto dado por

$$x = argmin_{x \in S^0} p(x)$$

Observação: O ponto é único, pois $p(x) = -\sum_{i=1}^{n} \log x_i$ é estritamente convexa.

Mostraremos agora a função *"centragem* x *custo"* que usamos para satisfazer os dois objetivos reduzir o custo e não se aproximar da fronteira. Essa função é proveniente de uma combinação linear das funções custo $f(x) = c^T x$ e barreira $p(x)$ da seguinte maneira: $f_\alpha(\cdot) = \alpha f(\cdot) + p(\cdot)$. Multiplicamos a função custo pelo parâmetro de penalidade α e veremos posteriormente que na medida que aumentarmos o α a função custo $f(\cdot)$ terá um peso maior. Assim, permitimos que os pontos gerados se afastem do centro e se aproximem da solução ótima. Com isso, podemos definir a chamada função penalizada da seguinte maneira:

$$f_\alpha(x) = \alpha c^T x + p(x)$$

onde $\alpha \in \Re, x \in \Re^n$.

A mesma função penalizada pode ser usada para a resolução do problema dual do seguinte modo:

$$f_\alpha(z) = \alpha b^T w + p(z)$$

onde $\alpha \in \Re, z \in \Re^n$.

Agora que foi apresentada a função que será usada para os algoritmos estudados vamos fazer um esboço dos passos necessários para a resolução do problema primal. Definimos um subproblema (P_{α^k}) da seguinte forma:

(P_{α^k})

$$\begin{array}{ll} \min & f_{\alpha^k}(x) \\ \text{sujeito a} & Ax = b \\ & x > 0 \end{array}$$

onde $x \in \Re^n$ e $\alpha \in (0, \infty)$ é um parâmetro qualquer, posteriormente será mostrada a forma para calculá-lo.

A cada atualização do parâmetro de penalidade α geramos um novo subproblema (P_{α_k}) e a solução de cada subproblema faz parte de uma curva chamada "trajetória central".

Com esta idéia podemos escrever o seguinte algoritmo:

Algoritmo III.1 Algoritmo Conceitual: *Dados* $x^0 \in S^0$ *e* $\alpha^0 \in (0, +\infty)$.
$k = 0$

Repita
  calcule $\alpha^{k+1} > \alpha^k$
  chame um algoritmo interno de minimização para encontrar
  x^{k+1} = solução $(P_{\alpha^{k+1}})$
  $k = k + 1$
Até convergir

Tomamos um algoritmo conceitual, pois na prática é impossível encontrar a solução exata de $(P_{\alpha^{k+1}})$.

Efeito da mudança de escala sobre a função barreira

Considere a matriz diagonal positiva D. Temos para a função barreira

$$p(Dx) = p(x) - \sum_{i=1}^{n} \log d_i$$

As operações de escala não afetam as variações de $p(\cdot)$. Temos para dois pontos x_1 e $x_2 > 0$.

$$p(Dx_2) - p(Dx_1) = p(x_2) - p(x_1)$$

Faremos agora um estudo das direções de minimização. A função custo $f(e) = c^T e$ (para o problema (P) após a mudança de escala) é uma função diferenciável no ponto e e sua aproximação linear é dada por

$$f(e + h) \cong f(e) + \nabla f(e)^T h$$

e a direção de máximo declive projetada no conjunto viável é dada por

$$h_1 = -P\nabla f(e) \qquad \text{(III.2)}$$

A função penalizada $f_\alpha(e)$ (após a mudança de escala) é extritamente convexa em e, sua aproximação quadrática em torno do ponto e é dada por

$$f_\alpha(e + h) \cong f_\alpha(e) + \nabla f_\alpha(e) + \nabla f_\alpha(e)^T h + \frac{1}{2} h^T (\nabla^2 f_\alpha(e)) h$$

Neste caso $\nabla^2 f_a(e)$ é inversível e a direção que minimiza a aproximação quadrática (direção de Newton-Raphson) projetada no conjunto viável é dada por

$$P\nabla f_a(e) + P\nabla^2 f_a(e)h_2 = 0 \qquad \text{(III.3)}$$

como $P\nabla^2 f_a(e) = I$ para a função barreira no ponto e temos

$$P\nabla f_a(e) + PIh_2 = 0$$

como $Ph_2 = h2$ por ser uma direção viável temos

$$h_2 = -P\nabla f_a(e)$$

coicidindo com h_1.

Com isso, temos várias razões para nos levar ao uso da mudança de escala.

- A variação da função barreira não é afetada pela mudança de escala
- A derivada no ponto e é extremamente fácil
- A matriz hessiana no ponto e é a matriz identidade
- A direção de máximo declive no ponto e coincide com a direção de Newton-Rahpson para a função penalizada.

O mais importante nos algoritmos estudados é o critério de atualização do parâmetro. A regra de atualização do parâmetro será vista mais adiante em proximidade de pontos centrais.

O algoritmo interno consiste em encontrar um ponto central, trata-se de um problema de minimização de uma função objetivo não linear com uma matriz hessiana simples, e a escolha natural para a resolução desse problema é o algoritmo de Newton-Rahpson. Nos algoritmos estudados utilizaremos a direção de máximo declive depois da mudança de escala III.2 que como vimos coincide com a direção de Newton-Rahpson para a função barreira.

Os resultados apresentados estão na formulação primal por ser mais simples, no final deste capítulo faremos o estudo de propriedades primais-duais . Todo o tratamento pode ser feito para a formulação dual com resultados equivalentes [8].

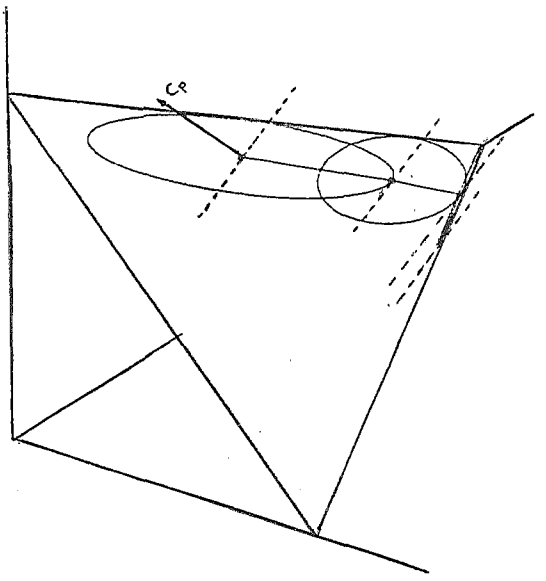

Figura III.2: Algoritmo afim-escala.

Se permitirmos que um algoritmo de pontos interiores dê passos longos demais é possível que ele gere uma sequência de pontos muito próximos à fronteira comprometendo o funcionamento do mesmo. Esta é a razão para tentarmos gerar pontos que se aproximem da "trajetória central" do politopo S. Agora vamos falar sobre o algoritmo afim-escala onde minimizamos uma função objetivo simples (aproximação linear) em uma região simples(uma bola). A bola é uma escolha óbvia para a região de confiança. Ao fazermos a mudança de escala transformamos a bola em um elipsóide simples com eixos paralelos às coordenadas (o maior elipsóide simples contido em S) e com isso aumentamos o tamanho da nossa região de confiança. A direção de minimização é a direção oposta ao gradiente projetado após a mudança de escala a qual chamamos direção escala máximo declive (SSD). Essa direção em princípio pode ser usada para qualquer função objetivo diferenciável contínua e será usada para os algorítmos deste trabalho. Vejamos o algoritmo afim-escala:

Algoritmo III.2 Algoritmo SSD: *Dados* $x^0 \in S^0$ *e* $f : S^0 \longrightarrow \Re$ *uma função diferenciável.*

$k = 0$

Repita

Mudança de Escala: $\bar{A} = AX_k$, $\bar{g} = X_k \nabla f(x^k)$

Direção: $\bar{h} = -P_{\bar{A}}\bar{g}$

Busca Unidirecional: Encontre $\bar{\lambda}$ tal que $\bar{g}(e + \bar{\lambda}\bar{h}) < \bar{g}(e)$, com $e + \bar{\lambda}\bar{h} > 0$

Passo: $\bar{y} = e + \bar{\lambda}\bar{h}$

Volta a escala original: $x^{k+1} = (X_k)^{-1}\bar{y}$

$k = k + 1$

Ate convergir

Nós podemos escrever o algoritmo SSD diretamente no espaço original. Vejamos:

Algoritmo III.3 Algoritmo SSD: *Dados $x^0 \in S^0$ e $f : S^0 \to \Re$ uma função diferenciável.*

$k = 0$

Repita

Direção: $h = -X_k P_{AX_k} X_k \nabla f(x^k)$

Busca Unidirecional: Encontre λ tal que $f(x^k + \lambda h) < f(x^k)$, com $x^k + \lambda h > 0$

Passo: $x^{k+1} = x^k + \lambda h$

$k = k + 1$

Ate convergir

O método afim-escala foi primeiramente proposto por Dikin [5]. Dikin tomou o passo unitário dado por $\lambda = 1/\|\bar{h}\|$. Esse algoritmo pode apresentar certos problemas como vemos na figura III.2 onde temos uma sequência de pontos muito próximos à fronteira do conjunto viável para esse PPL. Agora vamos ver uma forma de evitar o problema de congestionamento de pontos na fronteira gerando pontos centrais. Antes de definirmos a trajetória central, veremos a definição de pontos centrais.

Pontos Centrais

Considere os conjuntos da seguinte forma

$$S_K = \{x \in S | c^T x = K\}$$

Esses conjuntos são "fatias" com custo constante.

Definição 2.1: Pontos centrais do problema são os centros analíticos das fatias S_K com interior relativo não vazio, isto é, para todo $K \in \Re$ tal que $S_K^0 \neq 0$ define-se o ponto central

$$x(K) = argmin_x\{p(x)|Ax = b, c^T x = K, x > 0\}$$

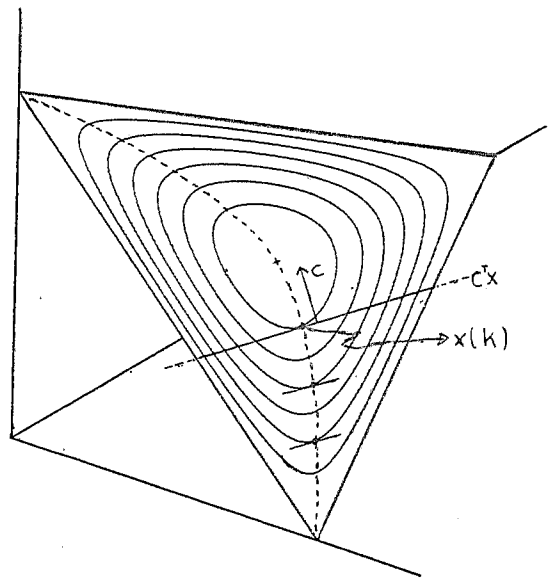

Figura III.3: Pontos centrais e curvas de nível da função barreira do PPL.

Na figura III.3 visualizamos os pontos centrais para as diversas fatias de custo constante. Geometricamente pode-se entender ponto central como o centro de uma fatia de custo constante K.

Embora a caracterização por fatias de custo constante vista acima seja a de melhor apelo geométrico, a seguinte definição equivalente de ponto central é mais útil. Podemos considerar a seguinte definição também como propriedade de pontos centrais.

Lema III.1 *Um ponto é central se e somente se existe $\alpha \in \Re$ tal que*

$$\overline{x} = argmin_{x \in S^0} f_\alpha(x) \qquad \text{(III.4)}$$

isto é, os pontos centrais são os minimizadores da função penalizada.

Demonstração: Um ponto $\overline{x}$ é central se e só se existe $K \in \Re$ tal que

$$\overline{x} = argmin_x\{p(x)|Ax = b, x > 0, c^T x = K\}$$

Usando a condições de Karush-Kuhn-Tucker, $\overline{x} \in S^0$ é central se e só se existe $y \in \Re^m$, $\epsilon \in \Re$ tais que

$$\nabla p(\overline{x}) - A^T y + \epsilon c = 0$$

Mas estas são exatamente as condições necessárias e suficientes de otimalidade para o problema III.4, completando a demonstração.

Trajetória Central

O conjunto de pontos centrais gerados pelas fatias de custo constante formam uma curva chamada trajetória central.

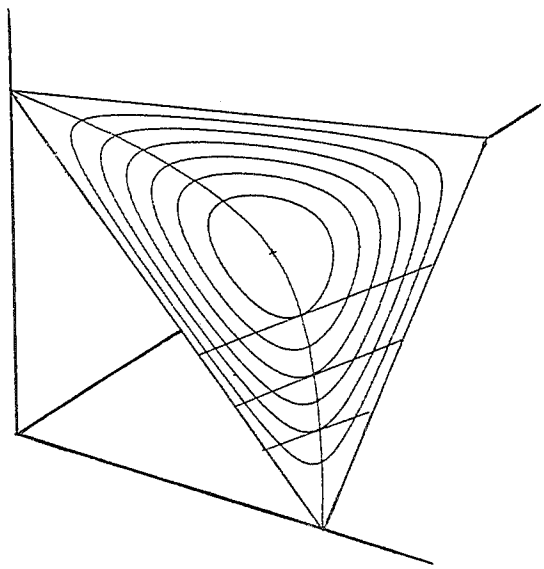

Figura III.4: A trajetória central e o centro analítico das fatias de custo constante.

A figura III.4 descreve o centro analítico das fatias de custo constante de S e a curva formada por esses centros chamada trajetória central.

Definição 2.3: Chamamos de trajetória central a curva que a cada $\alpha \in (0, +\infty)$ associa um único ponto $x(\alpha)$ onde

$$x(\alpha) = argmin_{x \in S^0} f_\alpha(x)$$

Vimos anteriormente que um ponto central é o minimizador da $f_\alpha(\cdot)$ para um determinado parametro de penalidade α e a cada atualização de α podemos encontrar um novo ponto central. Associamos a cada ponto central $x(\alpha)$ na trajetória central uma folga dual viável $z(\alpha)$ através do seguinte lema.

Lema III.2 *Seja $x(\alpha)$ um ponto central associado a $\alpha > 0$. Então,*

$$z(\alpha) = \frac{(x(\alpha))^{-1}}{\alpha}$$

é uma folga dual viável e o ponto central associado a α na trajetória central. O gap de dualidade associado a $x(\alpha)$ e $z(\alpha)$ é

$$x(\alpha)^T z(\alpha) = \frac{n}{\alpha}$$

Demostração [9] : Do lema II.1 temos que $b^T w = c^T x - x^T s$. A função dual penalizada pode ser escrita como

$$f_\alpha(z) = \alpha b^T w + p(s)$$

ou

$$f_\alpha(z) = \alpha c^T x(\alpha) - \alpha x(\alpha)^T s + p(s)$$

Desde que o primeiro termo é constante, o gradiente é dado por

$$\nabla f_\alpha(z) = \alpha x(\alpha) - z^{-1}$$

Substituindo $z = \frac{x(\alpha)^{-1}}{\alpha}$, ou $z^{-1} = \alpha x(\alpha)$, nós obtemos $\nabla f_\alpha(z) = 0$. Calculando

$$x(\alpha)^T z(\alpha) = \frac{x(\alpha)^T x(\alpha)^{-1}}{\alpha}$$

obtemos

$$x(\alpha)^T z(\alpha) = \frac{n}{\alpha}$$

completando a demonstração.

Este lema é muito importante, mas não tem muita utilidade prática. Não podemos encontrar um ponto central $x(\alpha)$ exato, como supõe o lema. Para encontrar $x(\alpha)$ precisamos realizar uma minimização de uma função não linear e o máximo que podemos obter é uma aproximação desse ponto. Mais adiante apresentaremos uma extensão deste lema para pontos próximos de $x(\alpha)$, muito mais útil para a prática.

Proximidade de Pontos Centrais

É impossível encontrar um ponto central exato no ponto de vista prático. O algoritmo interno deve ser resolvido com a maior rapidez possível. Em vista disso, será renunciada a opção ponto central e assumido um ponto "próximo" a um ponto central $x(\alpha)$. O algoritmo interno vai parar com tal critério de proximidade e a cada parada a função penalizada terá o parâmetro α atualizado.

Podemos reescrever o algoritmo III.1 da seguinte forma:

Algoritmo III.4 Algoritmo Geral: *Dados* $x^0 \in S^0$ *próximo de* $x(\alpha_0)$ *e* $\alpha_0 \in (0, +\infty)$.
$k = 0$

Repita
  calcule $\alpha^{k+1} > \alpha^k$
  chame um algoritmo interno de minimização para encontrar
  x^{k+1} = próximo a $x(\alpha^{k+1})$
  $k = k + 1$
Até convergir

Acima temos que $x(\alpha^{k+1}) = argmin\{f_{\alpha^{k+1}}(x) | x \in S^0\}$ onde $f_\alpha(\cdot)$ é a função penalizada.

Tentaremos fazer agora uma visualização geométrica da função penalizada. Veremos que quando $\alpha \mapsto +\infty$, $x(\alpha) \mapsto \hat{x}$, onde $\hat{x}$ é uma solução ótima do nosso problema primal. É bom resaltar que isso não é uma prova de convergência. Suponha que estamos trabalhando no espaço $\Re^2$. Na figura III.5 representamos as curvas de nível de $p : \Re^2 \longrightarrow \Re$ e a região $S \subset \Re^2$.

Observação: Note que as curvas de nível de $p(\cdot)$ em S vistas na figura III.5 são representadas por pares de pontos somente no centro analítico temos um ponto.

Os gráficos de $p(x) = -\sum_{i=1}^{n} \log x_i$ e $\alpha c^T x$ (para um dado α) estão representados em $\Re^3$.

Imagine agora (através das curvas de nível) o gráfico de $p(\cdot)$ e de $\alpha c^T x$ em $\Re^3$, façamos então um corte ortogonal a $\Re^2$ sobre a região viável S. Obtemos a figura III.6.

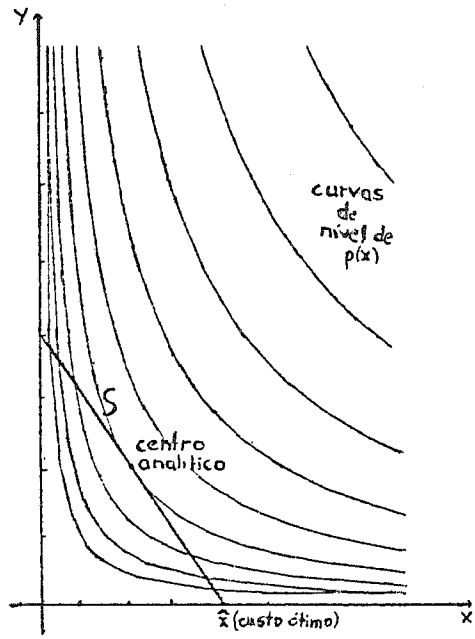

Figura III.5: Curvas de nível de $p : \Re^2 \longrightarrow \Re$ e a região $S \subset \Re^2$.

Com a figura III.6 já podemos representar a função $f_\alpha(x) = \alpha c^T c + p(x)$ e o respectivo ponto central associado a α na figura III.7. Veja que quando escolhemos $\alpha_2 > \alpha_1$ o ponto central associado a α_2 se desloca em direção ao ótimo. Sabemos que $\hat{x}$ é um ótimo devido a inclinação de $\alpha c^T x$.

III.3 Proximidade

É impossível encontrar um ponto exato $x(\alpha)$, então nós vamos assumir um critério para decidir quando um ponto x está "próximo" a um ponto central $x(\alpha)$.

Um ponto x pode ser considerado próximo ao centro analítico $x(\alpha)$ quando é possível encontrar $x(\alpha)$ a partir de x com poucas operações aritméticas. Encontrar aqui significa obter um ponto muito próximo de $x(\alpha)$ por um critério absoluto.

O algoritmo de Newton-Raphson que mostramos ser equivalente ao escala-máximo declive é muito eficiente para reduzir o valor da função barreira $p(\cdot)$ a partir de um ponto x. Supondo que e é o cento analítico temos $p(x)-p(e)$ como uma grandeza controlável. Se $p(x) - p(e)$ é pequeno, x está dentro da região de convergência quadrática do algoritmo, ou seja, se aproxima de $x(\alpha)$ muito

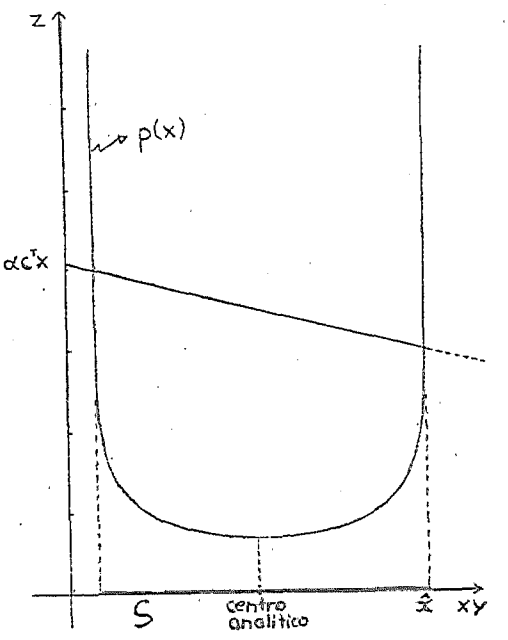

Figura III.6: Corte ortogonal a $\Re^2$ sobre a região viável S.

rapidamente. É pequeno aqui pode significar $p(x) - p(e) < 0.05$. Usamos $p(e)$ para simplificar nosso trabalho, pois todo tratamento será independente de escala, faremos o estudo no ponto $e = (1, \ldots, 1)$ estendendo o resultado para qualquer ponto.

A caracterização da proximidade ao centro analítico usada acima, da variação da função barreira, é inútil no ponto de vista dos algoritmos, pois a medida é não linear, provém de uma norma e é preciso conhecer o centro. Outra forma de medir a proximidade ao centro analítico: se $\|x - e\| < \beta$ onde β pode ser $\beta < 1$, mas como no anterior é um critério sem uso prático, pois seu cálculo depende do conhecimento do centro.

Lema III.3 *A direção de Newton-Raphson $h(x, \alpha)$ para a função penalizada é dada por*

i) $h(e, \alpha) = -P_A \nabla f_\alpha(e) = -P_A(\alpha c - e)$

ii) $h(x, \alpha) = X\bar{h}(x, \alpha)$ *onde*

$$\bar{h}(e, \alpha) = P_{AX} \nabla \bar{f}_\alpha(e) = -P_{AX} X \nabla f_\alpha(e)$$

e $e = (1, \ldots, 1)^T$.

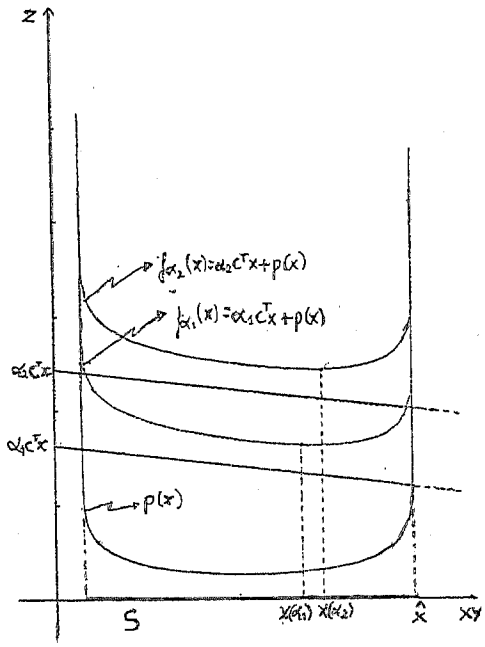

Figura III.7: Função $f_\alpha(x)$ e o respectivo ponto central associado a α.

Demonstração:

(i) Aproximação linear a partir de e para $\nabla f_\alpha(\cdot)$ é dada por

$$\nabla f_\alpha(e+d) \approx \nabla f_\alpha(e) + \nabla^2 f_\alpha(e)d = \nabla f_\alpha(e) + Id$$

O ponto que minimiza a aproximação quadrática sobre o $N(A)$ satisfaz

$$P_A(\nabla f_\alpha(e) + d) = 0$$

como $d \in N(A)$, temos $P_A(\nabla f_\alpha(e) + d) = 0$ completando a demonstração de (i).

(ii) Vamos usar o fato que o passo de Newton-Raphson não é afetado pela mudança de escala. A operação de escala $x^k = X\bar{x}$ leva o ponto x^k para e e reduz o problema ao caso anterior. Temos então,

$$\nabla \bar{f}_\alpha(e) = \alpha X c + e = X \nabla f_\alpha(e)$$

De (i) a direção de Newton-Raphson após a mudança de escala é

$$\bar{h}(x,\alpha) = -P_{AX} \nabla \bar{f}_\alpha(e)$$

ou

$$\bar{h}(x,\alpha) = -P_{AX} X \nabla f_\alpha(x) \qquad \text{(III.5)}$$

Na escala original temos

$$h(x,\alpha) = X\bar{h}(x,\alpha) \tag{III.6}$$

completando a demonstração.

Nossos cálculos são independentes de escala. No algoritmo escala-máximo declive todos os cálculos são feitos após a mudança de escala e o método de Newton-Raphson é naturalmente independente de escala. Para fixar melhor a notação da direção escala-máximo declive de III.5 e III.6 usaremos $h(x,\alpha)$ no espaço original e $\bar{h}(x,\alpha)$ no espaço transformado como

$$h(x,\alpha) = X\bar{h}(x,\alpha)$$

e agora definimos

$$\delta(x,\alpha) = \|\bar{h}(x,\alpha)\| \tag{III.7}$$

Vamos agora considerar um centro analítico $\tilde{x}$ de S . Imagine $\tilde{x} = e$. Neste caso simples, a função penalizada em torno de e pode ser escrita como

$$p(e+h) = p(e) + \nabla p(e)^T h + \frac{1}{2}h^T\nabla^2 p(e)h + 0(h)$$

Substituíndo $p(e) = 0$, $\nabla^2 p(e) = I$ de III.1 e desde que $\nabla p(e)^T h = 0$ porque e é o centro analítico,

$$p(e+h) = \frac{1}{2}\|h\|^2 + 0(h)$$

Para valores pequenos de $\|h\|$ ($\|h\| << 1$), as curvas de nível são quase esféricas como consequência do excelente comportamento da função logarítmica perto de 1. Vejamos a figura III.8.

A aproximação quadrática perto do centro é muito boa. É possível mostrar que para $\|h\| < \beta$, para $\beta < 1$ o erro na aproximação quadrática é muito pequeno, independente da dimensão do espaço.

Nós vimos no lema III.2 que pontos centrais tem importantes propriedades primais-duais. A direção escala-máximo declive permite a extensão dessas propriedades para a grande região que está próxima aos pontos centrais como mostraremos no lema a seguir. O lema mostra como obter uma folga dual viável mesmo para pontos não necessariamente próximos a um ponto central o que não ocorria no lema III.2.

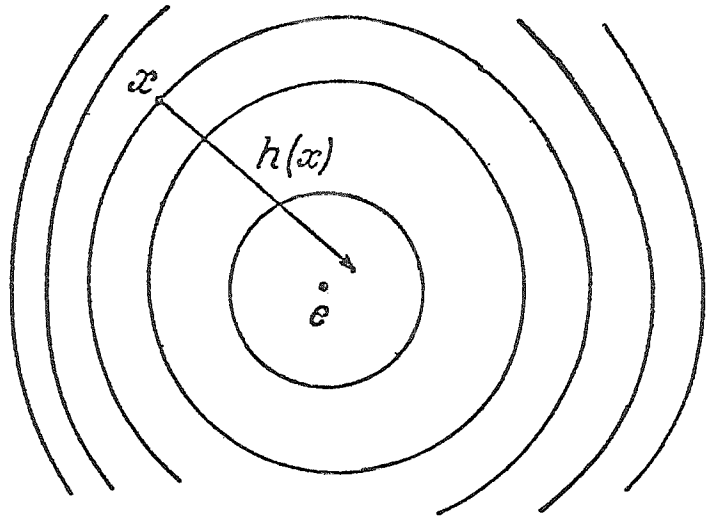

Figura III.8: Curvas de nível de $p(\cdot)$ perto do centro analítico.

Lema III.4 *Considere um parâmetro $\alpha > 0$ e um ponto $x \in S^0$. Se $\bar{h}(x,\alpha) > -e$ então*

$$z = X^{-1}\frac{e - \bar{h}(x,\alpha)}{\alpha}$$

é uma folga dual viável, e o gap de dualidade é

$$x^T z = \frac{n - e^T\bar{h}(x,\alpha)}{\alpha} \leq \frac{n + \delta(x,\alpha)\sqrt{n}}{\alpha}$$

Demonstração: Depois da mudança de escala temos

$$\bar{z} = \frac{e - \bar{h}(x,\alpha)}{\alpha}$$

ou

$$\bar{z}^T\alpha = e - \bar{h}(x,\alpha)$$

Sabemos que $\bar{h}(x,\alpha) = -\alpha c_p + e_p$, se fizermos a projeção

$$\bar{z}_p^T\alpha = e_p - \alpha\bar{c}_p + e_p$$

ou

$$\bar{z}_p = \bar{c}_p$$

Temos pelo lema II.2 que $P_A z = P_A c$ completando a prova de viabilidade.

O valor do gap no problema escalado (e também no problema original) é dado por

$$e^T \varepsilon = \frac{n + e^T \bar{h}(x, \alpha)}{\alpha}$$

A última desigualde vem do fato que

$$e^T \bar{h}(x, \alpha) \leq \|e\| \|\bar{h}(x, \alpha)\| = \sqrt{n}\delta(x, \alpha)$$

completando a demonstração.

Colorário 2.1: *Considere um parâmetro de penalidade $\alpha \geq 0$ e um ponto $x \in S^0$. Se $\delta(x, \alpha) < 1$ as afirmações do lema III.4 são válidas.*

Demonstração: Suponha que $\delta(x, \alpha) = \|\bar{h}(x, \alpha)\| < 1$. Então, $|\bar{h}(x, \alpha)| < 1$ para $i = 1, \ldots, n$. Logo, $\bar{h}(x, \alpha) < e$ completando a demonstração.

Os resultados principais de proximidade serão resumidos em um último lema.

Lema III.5 *Considere um parâmetro $\alpha \geq 0$, e um ponto $x \in S^0$ e sua proximidade $\delta \equiv \delta(x, \alpha)$.*

i) Eficiência do passo escala-máximo declive. Se $\delta < 1$ então $\tilde{x} = x + h(x, \alpha)$ é viável e

$$\delta(\tilde{x}, \alpha) \leq \delta^2$$

ii) Proximidade e distância euclidiana: Se $\delta < 1$ então

$$\|X^{-1}(x - x(\alpha))\| \leq \frac{\delta}{1 - \delta}$$

iii) Proximidade e valores da função: Se $\delta < 1$ então

$$f_\alpha(x) - f(x(\alpha)) \leq \frac{\delta^2}{1 - \delta^2}$$

iv) Decréscimo garantido: $\tilde{x} = x + \frac{1}{1-\delta} h(x, \alpha)$ é viável e

$$f_\alpha(\tilde{x}) \leq f_\alpha(x) - (\delta - log(1 + \delta))$$

Demonstração: Veja [10] onde aparece uma elegante demonstração e para isso são usados vários lemas.

O lema III.5 é o lema mais importante dessa seção, onde temos as propriedades:

(i) Dado um ponto inicial x^0 com $\delta(x,\alpha) < 1$, temos após um passo Newton-Raphson um ponto x^1 bem mais perto do ponto central. A distância do novo ponto encontrado (no caso x^1) ao ponto central reduziu numa taxa de convergência quadrática.

(ii) Na medida em que δ decresce diminui a distância de um ponto x ao ponto central $x(\alpha)$, mostrando que do ponto de vista geométrico temos uma boa definição de proxidade para $\delta < 1$.

(iii) Na medida em que aumenta a proximidade diminuindo o δ o valor da função penalizada no ponto x e no ponto central $x(\alpha)$ ficam mais próximos (com diferença menor) o que diz novamente que δ é uma boa medida de proximidade para $\delta < 1$.

(iv) Mostra que para pontos onde o δ é grande, uma iteração Newton-Raphson reduz bastante a função objetivo. Vejamos os exemplos.

Para $\delta(x,\alpha) = 2$ com um passo de aproximadamente $0.33h(x,\alpha)$ temos um decréscimo na função objetivo de

$$f_\alpha(y) \leq f_\alpha(x) - (2 - \lg 3) \approx f_\alpha(x) - 0.9$$

Para $\delta(x,\alpha) = 9$ temos $y = 0.1h(x,\alpha) + x$ com um decréscimo na função objetivo de

$$f_\alpha(y) \leq f_\alpha(x) - (9 - \lg 10) \approx f_\alpha(x) - 6.7$$

Com isso, acreditamos que temos uma boa maneira para medir a proximidade, vamos agora apresentar alguns resultados úteis para os algoritmos estudados.

Proximidade à Trajetória Central

Dado $\alpha > 0$, sabemos que a direção de Newton-Raphson $h(x,\alpha)$ corresponde a minimização quadrática de $f_\alpha(\cdot)$, e consequentemente de $e + h(x,\alpha)$ resulta

uma aproximação do ponto $x(\alpha)$, e $\|h(x,\alpha)\|$ aproxima a $\|\varepsilon - x(\alpha)\|$. A proximidade de ε a trajetória central é dada por

$$d = inf\{\|h(x,\alpha)\| | \alpha > 0\} \tag{III.8}$$

Lema III.6 *Se $c_p^T e_p > 0$ então o problema III.8 assume uma solução ótima em*

$$\bar{\alpha} = \frac{c_p^T e_p}{\|c_p\|^2} \tag{III.9}$$

por outro lado, $d = \|e_p\|$ corresponde a $\alpha \to 0$.

Demonstração: Se $c_p^T e_p \leq 0$ então para qualquer $\alpha > 0$

$$\| - \alpha c_p + e_p\| > \|e_p\|$$

e permite que $d = \|e_p\|$ para $\alpha \to 0$.

Considere agora o caso no qual $c_p^T e_p > 0$. Nós temos que

$$\bar{\alpha} = argmin\{\| - \alpha c_p + e_p\| | \alpha > 0\} \tag{III.10}$$

A solução do problema III.10 é facilmente obtida igualando a zero a expressão $\|h(x,\alpha)\|^2$.

Definimos

$$s(\alpha) = \| - \alpha c_p + e_p\|^2$$

ou

$$s(\alpha) = \alpha^2 \|c_p\|^2 - 2\alpha c_p^T e_p + \|e_p\|^2$$

derivando obtemos

$$\nabla s(\alpha) = 2\alpha \|c_p\|^2 - 2c_p^T e_p$$

igualando $\nabla s(\alpha)$ a zero temos

$$2\alpha \|c_p\|^2 = 2c_p^T e_p$$

ou

$$\alpha = \frac{c_p^T e_p}{\|c_p\|^2}$$

para $c_p^T e_p > 0$, completando a demonstração.

Na escolha do parâmetro de penalidade para os algoritmos usamos as duas possibilidades do lema anterior. No caso em que $c_p^T e_p \leq 0$, nós só podemos concluir que e está longe da trajetória central, com isso determinamos uma regra heurística para a escolha do parametro de penalidade. No contrário, usamos a expressão III.9 com um multiplicador μ que determina um aumento da penalidade para a iteração seguinte. O cálculo do parametro de penalidade para o algoritmo primal dado um $\mu \in (0, \infty)$ é:

$$\text{Se } c_p^T e_p \leq 0 \text{ então } \alpha^{k+1} = \frac{\|e_p\|}{\|c_p\|}$$

$$\text{senão } \alpha^{k+1} = \mu \frac{c_p^T e_p}{\|c_p\|^2}$$

O multiplicador do parâmetro de penalidade será discutido no próximo capítulo juntamente com um cálculo para determinar um aumento ao mesmo.

III.4 Métodos Primais-Duais

O lema II.2 é muito importante por fornecer uma regra simples para testar a viabilidade de uma folga dual e algumas conclusões são obtidas da posse deste resultado.

Dado $\tilde{x} \in S$ um problema dual equivalente a (D2) (no sentido de que possui as mesmas soluções ótimas) pode ser escrito como:

(D3)

$$\begin{array}{ll} \min & \tilde{x}^T z \\ \text{sujeito a} & P_A z = P_A c \\ & z \geq 0 \end{array}$$

Neste caso o conjunto dual viável e seu interior relativo são respectivamente:

$$R = \{z \in \Re^n | P_A z = P_A c, z \geq 0\}$$

$$R^0 = \{z \in \Re^n | P_A z = P_A c, z > 0\}$$

Aqui o objetivo é reduzir o gap de dualidade, e o valor ótimo é $\tilde{x}^T z = c^T \tilde{x} - \hat{v}$. O problema primal (P) também pode ser modificado através da troca de $c_p^T x$ por $\tilde{c}_p^T x$ para qualquer folga dual viável $\tilde{z}$. Vejamos o problema primal equivalente.

(P2)

$$\begin{array}{ll} \min & \tilde{z}^T x \\ \text{sujeito a} & Ax = b \\ & x \geq 0 \end{array}$$

No problema primal, $\tilde{z} \in R^0$ é uma folga dual viável, no problema dual $x \in S^0$ é uma solução primal viável.

Note que como $P_A x = P_A c$, para todo x viável, minimizar $c^T x$ é idêntico a minimizar $z^T x$ para $z \in R^0$.

A função penalizada para o problema primal-dual é definida como

$$f_\alpha(x, z) = \alpha x^T z - \sum_{i=1}^{n} \log x_i - \sum_{i=1}^{n} \log z_i$$

onde $x \in S^0$, $z \in R^0$ e $\alpha \geq 0$.

Para resolver o problema primal-dual geramos uma sequência de subproblemas mais simples (PD_{α^k}) semelhantes aos que usamos para resolver o problema primal.

(PD_{α^k})

$$\begin{array}{ll} \min & f_{\alpha^k}(x, z) \\ \text{sujeito a} & x \in X^0 \\ & z \in R^0 \end{array}$$

A cada atualização do parâmetro de penalidade α nós teremos um novo subproblema e como anteriormente o algoritmo reduz o valor de $f_\alpha(\cdot, \cdot)$ a cada resolução de (PD_{α^k}) assegurando convergência para um ótimo de (P2) e de (D3).

Direção

Agora devemos encontrar duas direções viáveis, uma em X^0 e outra em R^0. Dado $\alpha > 0$ e um par interior (x, z) teremos duas direções viáveis h_x e h_z de forma que $h_x \in N(A)$ e $h_z \in N(P_A) = I(A^T) \perp N(A)$.

$$h_x = -P_A \nabla_x f_\alpha(x, z)$$

$$h_z = -\tilde{P}_A \nabla_z f_a(x, z)$$

onde $\tilde{P}_A = I - P_A$.

No problema primal-dual procuramos resolver simultaneamente os dois problemas (primal e dual) na tentativa de melhorar o desempenho do método pelo acréscimo de informações a cada iteração.

Mudança de Escala

Veremos a seguir como a variável dual será particularmente útil na definição de uma nova mudança de escala.

Se fizermos uma mudança de escala primal $x = X\bar{x}$ teremos uma boa direção $h_{\bar{x}}$ e a direção h_x será prejudicada. Caso tentarmos gerar uma boa direção h_x fazendo a mudança de escala $x = Z^{-1}\bar{z}$ a direção $h_{\bar{x}}$ não será boa. É necessário, portanto, escolher com cuidado a mudança de escala a ser realizada, pois estamos interessados em resolver (P2) e (D3) simultaneamente, devemos escolher uma mudança de escala que nos possibilite encontrar boas direções de otimização para os dois problemas. Se fizermos duas mudanças de escala independentes, uma para o problema primal e outra para o problema dual será necessário calcular duas projeções. O cálculo da matriz de projeção é muito dispendioso para Algoritmos de Pontos Interiores, pois envolve uma inversão de matriz. Em vista disso, vamos usar a mudança de escala primal-dual

$$\bar{x} = \bar{z} = (XZ)^{\frac{1}{2}}$$

Com isso, os pontos x e z serão levados ao mesmo ponto e será necessário, apenas, calcular uma projeção, pois a direção relativa a uma região viável é o complemento ortogonal da outra.

A justificativa para se utilizar a mudança de escala primal-dual é que funciona melhor por aproveitar informações provenientes da variável dual [17].

Juntando Escala e Direção

Para aproveitar a dualidade vamos usar a matriz diagonal D, com $x = D\bar{x}$ e $z = D^{-1}\bar{z}$ obtendo resultados interessantes.

Vejamos a matriz diagonal

$$D = X^{\frac{1}{2}} Z^{-\frac{1}{2}}$$

Agora x e z são mapeados em $D^{-1}x = Dz = (XZ)^{-\frac{1}{2}}$ e as direções de ecala-máximo declive são:

$$h_x = -DP_{AD}(\alpha(XZ)^{\frac{1}{2}}e - (XZ)^{-\frac{1}{2}}e)$$

$$h_z = -D^{-1}\tilde{P}_{AD}(\alpha(XZ)^{\frac{1}{2}}e - (XZ)^{-\frac{1}{2}}e)$$

Pontos Centrais

Os pontos centrais primais-duais são os minimizadores da $f_\alpha(x,z)$ para $x \in S^0$ e $z \in R^0$. Nós associamos a cada $\alpha > 0$ um ponto central $x(\alpha)$ e uma folga dual $z(\alpha)$, definidos no lema III.2 como $z(\alpha) = x^{-1}(\alpha)$. Agora mostraremos que pontos centrais primais-duais são consistentes com estas definições.

Lema III.7 *Seja $\alpha > 0$ e (x,z) um par de pontos interiores viáveis. Então as seguintes afirmações são equivalentes:*
i) (x,z) forma um ponto central primal-dual associado a α.
ii) $\nabla_x f_\alpha(x,z) = 0$ e $\nabla_z f_\alpha(x,z) = 0$.
iii) $\alpha XZ = I$.
iv) $x = x(\alpha)$ e $z = z(\alpha)$.

Demonstração [10]: Assuma que $(\bar{x},\bar{z})$ é central. Fixando $\bar{z}$,

$$f_\alpha(\bar{x},\bar{z}) = min_{x \in R^0} f_\alpha(x,\bar{z}),$$

logo $\bar{x} = x(\alpha)$.

Para $z \in R^0$ qualquer, temos que

$$\nabla_x f_\alpha(\bar{x},z) = \alpha z - \bar{x}^{-1}$$

$$\nabla_z f_\alpha(\bar{x},z) = \alpha \bar{x} - z^{-1}$$

Substituindo $z = z(\alpha) = \bar{x}^{-1}/\alpha$ nestas expressões,

$$\nabla_x f_\alpha(\bar{x},z) = \nabla_z f_\alpha(\bar{x},z) = 0.$$

Como $f_\alpha(\cdot,\cdot)$ é estritamente convexa, z é o único mínimo de $f_\alpha(\bar{x},\cdot)$, logo $\bar{z} = z = z(\alpha)$. Isto prova a equivalência de (i) e (ii) e (iv). A equivalência de (ii) e (iii) é imediata das expressões dos gradientes, completando a demonstração.

Do lema anterior temos um ponto central $(x(\alpha), z(\alpha))$ composto pelo ponto central primal $x(\alpha)$ e sua folga dual $z(\alpha)$, então, podemos definir trajetória central.

Trajetória Central

A trajetória central é formada pelo conjunto de pontos centrais primais-duais $(x(\alpha), z(\alpha))$ com $\alpha > 0$.

O lema III.7 fornece uma maneira simples de medir a proximidade

$$\delta(x, z, \alpha) = \|\alpha X z - e\|$$

Vejamos a interpretação desse critério: Quando $\delta(x, z, \alpha) = 0$, $\alpha x - e = 0$ pelo item (iii) do lema III.7. Como (iii) é equivalente a $x_i z_i = \frac{1}{\alpha}$, $i = 1, \ldots, n$, temos que as folgas complementares estão balanceadas. Além disso, quando α tende para infinito $x_i z_i$ tende para zero, indicando que o gap $x^T z$ tende para zero no ótimo. A relação entre este critério de proximidade e o critério primal pode ser vista no seguinte lema:

Lema III.8 *Considere um parâmetro de penalidade $\alpha > 0$ e um ponto $x \in S^0$. Então,*

$$\delta(x, \alpha) = min\{\|\alpha X z - e\| | A^T w + z = c, w \in \Re^m\}$$

Demonstração: Nós vimos em II.1 que

$$c_p = argmin_x\{\|x - c\| | x \in N(A)\}$$

e uma declaração similar pode ser associada ao complemento ortogonal

$$\|c_p\| = min_w\{\|c - A^T w\| | w \in \Re^m\}$$

Desse modo, podemos dizer que

$$\begin{aligned}
\delta(x, \alpha) = \|\bar{h}(x, \alpha)\| &= min_w\{\alpha\bar{c} - e - \alpha\bar{A}^T w | w \in \Re^m\} \\
&= min_w\{\|\alpha X(c - A^T w) - e\| | w \in \Re^m\} \\
&= min_w\{\|\alpha X z - e\| | A^T w + z = c, w \in \Re^m\}
\end{aligned}$$

Resta-nos mostrar que o mínimo ocorre em $z(\alpha)$. Substituindo a expressão do lema III.4 em $\delta(x,z,\alpha)$,

$$\delta(x,z,\alpha) = \|\alpha X(X^{-1}\frac{e-\tilde{h}(x,\alpha)}{\alpha}) - e\| = \|\tilde{h}(x,\alpha)\| = \delta(x,\alpha),$$

completando a demonstração.

O cálculo do parametro de penalidade para o algoritmo primal-dual é simples para este caso

$$\delta(x,z)^2 = min\|\alpha ZXe - e\|^2$$

Derivando a expressão em relação a α e igualando a zero nós obtemos:

$$\frac{1}{\alpha} = \frac{x^T z}{n}$$

Com isso podemos escrever o calculo do parametro de penalidade para o algoritmo primal-dual: Dado um $\mu \in (0,\infty)$ temos:

$$\alpha^{k+1} = \mu\frac{n}{x^{kT}z^k}$$

Concluímos este capítulo acreditando que em poucas palavras foram mostrados os resultados mais importantes para o entendimento dos algoritmos apresentados no próximo capítulo.

Capítulo IV

ALGORITMOS DE BARREIRA

Vamos testar um conjunto de algoritmos primais e primais-duais, todos baseados na função penalizada.

O algoritmo primal básico segue o modelo III.4, e estudamos variantes dos tipos seguintes:

i) Algoritmos com ou sem iterações internas: algoritmos com iterações internas geram para cada valor do parâmetro α um ponto próximo ao ponto central $x(\alpha)$, e são portanto métodos que seguem a trajetória central. Algoritmos sem iterações internas variam α a partir de cada ponto gerado, sem se preocupar com a obtenção de um ponto próximo a $x(\alpha)$.

ii) Algoritmos com passos fixos ou acelerados por α: os algoritmos com ou sem iterações internas, fazem periodicamente um incremento do parâmetro α,dado por $\alpha^{k+1} = \mu\alpha^k$. O valor do μ pode ser fixo ou variar adaptativamente, acelerando-se o algoritmo dependendo do seu comportamento na iteração anterior.

iii) Buscas unidirecionais e bidirecionais: em cada iteração interna de qualquer algoritmo, é necessário fazer uma busca para determinar o próximo ponto. Essa busca é normalmente feita ao longo da direção de máximo declive (após escala) para a função critério. Examinando essa direção, nota-se que ela é sempre dada por uma combinação de duas direções bem conhecidas. A primeira é $-c_p$, a direção afim-escala para a redução do custo e a segunda é e_p, direção de centralização (direção de Newton-Raphson para a função barreira).

O procedimento da busca bidirecional explora o espaço bidimensional gerado por c_p e e_p, ao invés de seguir uma determinada combinação delas.

A seguir, examinamos com mais detalhes cada um desses fatores.

IV.1 Minimização Unidirecional

A minimização unidirecional é mais simples. Para uma direção viável h (Newton-Raphson ou SSD) nós temos uma reta $\lambda \geq 0 \mapsto x^k + \lambda h$ na qual será feita a minimização. O problema da busca unidirecional consiste em resolver

$$min\{f_a(x^k + \lambda h), \lambda \geq 0 | x^k + \lambda h \in S^0\}$$

Para fazer a busca unidirecional em $f_a(\cdot)$ podemos utilizar vários métodos como: bisseção, Armijo, seção áurea[8]. Usamos o método da bisseção com uma precisão elevada, que é um método trivial e certamente pouco eficiente. Como o interesse deste trabalho está no estudo do número de iterações dos algoritmos, e não no tempo de computação de cada iteração, não nos preocupamos com a eficiência da busca unidirecional.

Algoritmo IV.1 Método da bisseção: *Dada a direção h e um ponto x, inicialmente x determina $w = max\{\lambda | x + \lambda h \in S\}$, o que é obtido por um teste de razão. A solução ótima do problema acima está no interior do conjunto $[0, w]$. Dado um valor $\lambda \in (0, w)$, a derivada da função objetivo $\lambda \mapsto f(x + \lambda h)$ é dada por $\theta(\cdot) = \nabla f(x + \lambda h)^T h$, o que é facilmente computável.*

$v = 0$

Repita

$\lambda = \frac{(v+w)}{2}$

Se $\theta(\lambda) < 0$ então $v = \lambda$

senão $w = \lambda$

Ate $(w - v) < 10^{-5}$

IV.2 Minimização Bidirecional

Estudamos o procedimento de busca bidirecional a partir do ponto e após uma mudança de escala.

A busca bidirecional apresentada por Gonzaga [11] dá o passo de maior declive da função objetivo em cada iteração combinando as direções $-c_p$ e e_p. Temos o problema

$$min\{f_a(e - d_1 c_p + d_2 e_p) | d_1, d2 \in \Re, e - d_1 c_p + d_2 e_p > 0\}$$

Resolver este problema consiste em examinar pontos $e - d_1 c_p + d_2 e_p$ fazendo uma busca em uma região bidimensional.

Na resolução do problema acima, muitas vezes queremos um decréscimo de custo garantido. Para isso, usamos a direção $-c_p$ e a projeção de e_p sobre o espaço nulo de c_p dada por

$$h_2 = e_p - \frac{e_p^T cp}{\|c_p\|^2} c_p \qquad \text{(IV.1)}$$

As direções c_p e h_2 são ortogonais, e o decréscimo do custo é garantido forçando d_1 a ser positivo. A direção h_2 é a direção de centragem de custo constante, foi primeiramente usada por Barnes [2] e nós acreditamos que ela deve ser usada ao invés de e_p. É lógico que o subespaço gerado por c_p e h_2 coicide com o subespaço gerado por c_p e e_p. A região da busca será

$$B = \{e - d_1 c_p + d_2 e_p | d_1, d2 \in \Re, e - d_1 c_p + d_2 e_p \in \Re^n_+\}$$

No procedimento a seguir, permitimos $d_2 < 0$ e usamos o algoritmo de Newton-Raphson no espaço bidimensional. Para fazer a busca bidirecional em B definimos $p(d_1, d_2) = e - d_1 c_p + d_2 e_p$ para cada par $(d_1, d_2) \in \Re^2$ e definimos $g_a(d_1, d_2) = f_a(p(d_1, d_2))$ para (d_1, d_2) tais que $p(d_1, d_2) > 0$. Agora o nosso problema pode ser escrito

$$min\{g_a(d_1, d_2) | p(d_1, d_2) > 0\}$$

Normalizando as direções c_p e h_2 da equação IV.1 temos

$$h_1 \leftarrow \frac{c_p}{\|c_p\|}$$

$$h_2 \leftarrow \frac{h_2}{\|h_2\|}$$

Fazendo a busca bidirecional ao longo das direções h_1 e h_2 partindo do ponto e a nossa função $g_\alpha(d_1, d_2)$ fica assim definida

$$g_\alpha(d_1, d_2) = \alpha c^T(e - d_1 h_1 + d_2 h_2) - \sum_{i=1}^{n} \log(e - d_1 h_1 + d_2 h_2)$$

As derivadas de $g_\alpha(d_1, d_2)$ são

$$\frac{\partial g_\alpha(d_1, d_2)}{\partial d_1} = -\alpha c_p^T h_1 + \sum_{i=1}^{n} \frac{h_{1_i}}{e - d_1 h_1 + d_2 h_2}$$

$$\frac{\partial g_\alpha(d_1, d_2)}{\partial d_2} = \alpha c_p^T h_2 - \sum_{i=1}^{n} \frac{h_{2_i}}{e - d_1 h_1 + d_2 h_2}$$

A matriz hessiana é:

$$\nabla^2 g_\alpha(d_1, d_2) = \begin{bmatrix} \sum_{i=1}^{n} \frac{h_{1_i}^2}{(e - d_1 h_1 + d_2 h_2)_i^2} & -\sum_{i=1}^{n} \frac{h_{1_i} h_{2_i}}{(e - d_1 h_1 + d_2 h_2)_i^2} \\ -\sum_{i=1}^{n} \frac{h_{1_i} h_{2_i}}{(e - d_1 h_1 + d_2 h_2)_i^2} & \sum_{i=1}^{n} \frac{h_{2_i}^2}{(e - d_1 h_1 + d_2 h_2)_i^2} \end{bmatrix}$$

Agora podemos encontrar uma solução para o nosso problema da busca bidirecional usando qualquer algoritmo de programação não linear. A função objetivo é estritamente convexa e o Algoritmo de Newton-Raphson pode ser usado resultando numa boa aproximação com apenas três iterações. Nós descrevemos o algoritmo a seguir.

Algoritmo IV.2 Busca Bidirecional: *Dado* c_p, e_p .

$d = 0$

$h_2 = e_p - \frac{c_p^T e_p}{\|c_p\|^2} c_p$

$G = \left(-\frac{c_p}{\|c_p\|}, \frac{h_2}{\|h_2\|}\right)$

Repita

$g = \left(\frac{\partial g_\alpha(d_1, d_2)}{\partial d_1}, \frac{\partial g_\alpha(d_1, d_2)}{\partial d_2}\right)$

$H = \nabla^2 g_\alpha(d_1, d_2)$

$p = -H^{-1} g$

$h = Gd$

$\bar{x} = e + d$

Busca unidirecional: $\bar{\lambda} = argmin_\lambda \{f_\alpha(\bar{x} + \lambda h) | \bar{x} + \lambda h > 0\}$

$d = d + \bar{\lambda} p$

Ate $\|h\| < 1$

$\bar{y} = e + Gd$

Mesmo com uma inversão de matriz os cálculos no algorimo são simples e rápidos por se tratar de matrizes 2x2.

Duas Buscas com Uma Mudança de Escala

Como o cálculo da projeção P_A é o cálculo mais demorado, ou mais dispendioso nos Algoritmos de Pontos Interiores procuramos reduzir o número de projeções fazendo duas buscas com uma mudança de escala. Neste caso o algoritmo faz uma busca ao longo da direção de Newton-Rapson e outra ao longo da direção de Cauchy dada por

$$h = -\alpha c_p + x_p^{-1}$$

esse algoritmo foi implementado para duas buscas com minimização unidirecional e bidirecional.

IV.3 Algoritmo com Iterações Internas

Agora vamos mostrar o algoritmo que segue a trajetória central usando um algoritmo interno para gerar pontos com uma proximidade grande ($\delta < 1$). O cálculo de α^{k+1} será estudado mais adiante. Vejamos o algoritmo:

Algoritmo IV.3 Algoritmo com Iterações Internas: *Dado um ponto* $x^0 \in S^0$, $\mu \geq 0$ *e uma precisão* $\in (0,1)$.

$k = 0$

$y^0 = x^0$

Mudança de Escala: $Y = diag(y_1^k, \ldots, y_n^k)$, $\bar{A} = AY$, $\bar{c} = Yc$

Projeção: $P = I - \bar{A}^T(\bar{A}\bar{A}^T)^{-1}\bar{A}$, $c_p = P\bar{c}$, $e_p = Pe$

Repita

Calculo do parâmetro de penalidade α^{k+1}

$h = -\alpha^{k+1} c_p + e_p$

$j = 0$

$y^0 := x^k$

Repita

Busca Unidirecional ou Bidirecional a partir de e determinando $\bar{y}$

Volta a escala original: $y^{j+1} = Y^{-1}\bar{y}$

$j = j + 1$

Mudança de Escala: $Y = diag(y_1^j, \ldots, y_n^j)$, $\bar{A} = AY$, $\bar{c} = Yc$

Projeção: $P = I - \bar{A}^T(\bar{A}\bar{A}^T)^{-1}\bar{A}$, $c_p = P\bar{c}$, $e_p = Pe$

$h = -\alpha^{k+1}c_p + e_p$

$\delta = \|h\|$

Ate $(\delta < 1)$

$x^{k+1} = y^j$

$k = k + 1$

Ate $(\frac{n}{\alpha} <$ precisão$)$

IV.4 Algoritmo sem Iterações Internas

Agora mostramos o algoritmo que não segue a trajetória central.

Algoritmo IV.4 Algoritmo sem Iterações Internas: *Dado um ponto* $x^0 \in S^0$, $\mu \geq 0$ *e uma precisão* $\in (0, 1)$.

$k = 0$

Repita

Mudança de Escala: $X = diag(x_1^k, \ldots, x_n^k)$, $\bar{A} = AX$, $\bar{c} = Xc$

Projeção: $P = I - \bar{A}^T(\bar{A}\bar{A}^T)^{-1}\bar{A}$, $c_p = P\bar{c}$, $e_p = Pe$

Calculo do parâmetro de penalidade α^{k+1}

$h = -\alpha^{k+1}c_p + e_p$

Busca Unidirecional ou Bidirecional a partir de e determinando $\bar{x}$

Volta a escala original: $x^{k+1} = X^{-1}\bar{x}$

$k = k + 1$

Ate $(\frac{n}{\alpha} <$ precisão$)$

IV.5 Cálculo do α^{k+1}

Dado um ponto x^k ao iniciar-se uma iteração do algoritmo externo, devemos escolher o valor de α^{k+1} do parâmetro de penalidade. Essa escolha será feita em todos os métodos pelo processo descrito no capítulo anterior: dado x^k, calcula-se um valor α_N associado ao ponto da trajetória central mais próximo (pelo critério de proximidade $\delta(\cdot)$) de x^k. O valor de α^k é abandonado e faz-se

$$\alpha^{k+1} = \mu\alpha_N$$

onde μ é um multiplicador.

O valor de μ pode ser fixo, e várias escolhas serão testadas no próximo capítulo, ou pode ser acelerado durante a execução do algoritmo. Descrevemos agora como é feita a aceleração.

Primeiro Método

Para algoritmos com iterações internas: se o algoritmo interno foi muito eficiente na centralização na iteração k, isto é, se terminar com $j < 3$, aumenta-se μ por $\mu \leftarrow 1.5\mu$.

Segundo Método

Para algoritmos sem iterações internas. Suponha que na iteração $k-1$, partíamos de x^{k-1}, calculando o valor α_N^- correspondente ao ponto da trajetória central mais próximo (pelo nosso critério) de x^{k-1}.

A iteração k do algoritmo fez $\alpha^k = \mu\alpha_N^-$ e encontrou o novo ponto x^k como mostra a figura IV.1.

Seja α_N o parâmetro do ponto central mais próximo de x^k. Inicialmente gostaríamos de obter $\alpha_N = \alpha^k$, com uma centralização perfeita em uma iteração. Isto é, idealmente, teríamos

$$\frac{\alpha_N}{\alpha_N^-} = \mu$$

ou equivalente, já que $\alpha^{k+1} = \mu\alpha_N$ e $\alpha^k = \mu\alpha_N^-$,

$$\frac{\alpha^{k+1}}{\alpha^k} = \mu$$

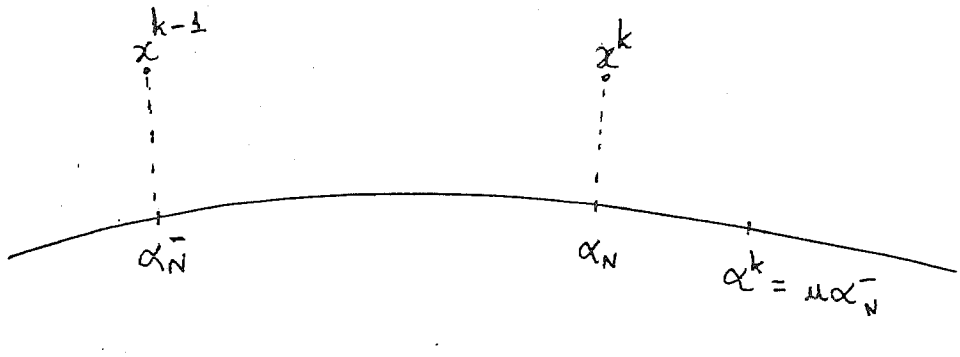

Figura IV.1: Pontos da Trajetória Central.

$$\log \alpha^{k+1} - \log \alpha^k = \log \mu$$

Consideramos que a iteração foi muito bem sucedida se $\log \alpha^{k+1} - \log \alpha^k \geq \beta \log \mu$, onde escolhemos $\beta = 0.5$. Neste caso, podemos acelerar o algoritmo: Se $\frac{\alpha^{k+1}}{\alpha^k} \geq \sqrt{\mu}$ então $\mu = 1.5\mu$.

IV.6 Algoritmo Primal - Dual

Para a execução do algoritmo primal-dual precisamos calcular uma folga dual viável inicial correspondente a um ponto x próximo à trajetória central. Pelo lema III.4 podemos escrever o cálculo de z inicial para o algoritmo primal-dual. Suponhamos que o algoritmo inicia a partir de um ponto x próximo a $x(\alpha)$, onde α é um parâmetro de penalidade. A obtenção desse ponto será feita por uma centralização. Para executar o algoritmo primal-dual precisamos iniciar pelo cálculo de uma folga dual viável z. Para isso, necessitamos um ponto inicial x próximo à trajetória central, a partir do qual aplicamos o lema III.4. Esse ponto é obtido por uma centralização. Para x próximo à trajetória, teremos $\|h\| < 1$, garantindo-se viabilidade pelo lema III.4.

Calculo do z:
$$\begin{cases} \text{Mudança de Escala: } X = diag(x_1^k, \ldots, x_n^k),\ \bar{A} = AX,\ \bar{c} = Xc \\ \text{Projeção: } P = I - \bar{A}^T(\bar{A}\bar{A}^T)^{-1}\bar{A},\ c_p = P\bar{c},\ e_p = Pe \\ \alpha = \dfrac{c_p^T e_p}{\|c_p\|^2} \\ \bar{h} = -\alpha c_p + e_p \\ \bar{z} = (e - \bar{h})\alpha \\ z = X^{-1}\bar{z} \end{cases}$$

O algoritmo primal-dual com iterações internas pode ser escrito:

Algoritmo IV.5 Primal-Dual: *Dado um ponto x próximo a $x(\alpha), z$ (calculado acima) $\mu = 10$ e uma precisão $\in (0,1)$.*

$k = 0$

Escala Primal-Dual $\begin{cases} D = \sqrt{XZ^{-1}} \\ \bar{x} = D^{-1}x \\ \bar{z} = Dz \\ \bar{A} = AD \end{cases}$

Repita

Penalidade: $\alpha = \mu \frac{n}{\bar{x}^T \bar{z}}$

$\delta = \|\alpha X z - e\|$

$j = 0$

Repita

$h_x = -P_{\bar{A}}(\alpha \bar{z} - \bar{x}^{-1})$

$h_z = -(I - P_{\bar{A}})\alpha \bar{x} - \bar{z}^{-1}$

$\bar{\lambda}_x = argmin\{f_\alpha(\bar{x} + \lambda_x h_x, \bar{z}) | \bar{x} + \lambda_x h_x > 0\}$

$\bar{\lambda}_z = argmin\{f_\alpha(\bar{x}, \bar{z} + \lambda_z h_z) | \bar{z} + \lambda_z h_z > 0\}$

$\bar{x} = \bar{x} + \bar{\lambda}_x h_x$

$\bar{z} = \bar{z} + \bar{\lambda}_z h_z$

Volta a escala original $\begin{cases} x = D\bar{x} \\ z = D^{-1}\bar{z} \end{cases}$

Escala Primal-Dual $\begin{cases} D = \sqrt{XZ^{-1}} \\ \bar{x} = D^{-1}x \\ \bar{z} = Dz \\ \bar{A} = AD \end{cases}$

$\delta = \|\alpha X z - e\|$

$j = j + 1$

Ate $(\delta < 1)$

$k = k + 1$

Ate $(\frac{n}{\alpha} <$ precisão$)$

Podemos resumir as combinações de algoritmos testados na tabela abaixo, todos com $\mu^0 = 10$, e x^0 próximo a trajetória central.

		Algoritmos Primais				Alg. Primal-Dual
		busca unidirecional		busca bidirecional		busca unidirecional
		uma busca	duas buscas	uma busca	duas buscas	uma busca
com iter. inter.	sem acel.	PT	PT3	PT2		PDT
	com acel.	PTA		PTA2		
	acel. $j > 3$	PTA_3		PT$A_3$2		
sem iter. inter.	sem acel.	PR		PR2		PDR
	com acel.	PRA	PR3	PRA2	PRA4	PDRA

Os algoritmos PTA e PTA2 consiste em fazer $\mu = 1.5\mu$ em todas iterações externas. Testamos os algoritmos primais sem iterações internas com um ponto inicial x^0 não necessariamente próximo a trajetória central. Foram, também, testados em alguns dos algoritmos anteriores diferentes valores iniciais para μ^0. Os resultados obtidos nos testes e os comentários dos mesmos são mostrados no próximo capítulo.

Capítulo V

TESTES COMPARATIVOS

Vamos apresentar na primeira seção alguns detalhes da implementação computacional, na segunda seção serão mostradas algumas características dos problemas testados e por último falaremos dos resultados dos testes.

V.1 Implementação Computacional

Os programas foram feitos em Turbo Pascal[1] 4.0 e o micro utilizado EBC PC/XT. A montagem dos algoritmos e o gerador de problemas foram implementados a partir do pacote desenvolvido por Arantes e Tortorelli [1]. Não nos preocupamos em fazer um código muito eficiente computacionalmente, pois apenas estudamos o comportamento dos algoritmos em relação ao número de iterações necessárias para resolver o problema.

A máquina utilizada não permitiu problemas de tamanho grande e não estavamos interessados em resolver problemas de aplicação prática. Dessa forma não usamos técnicas especiais para a inversão de matrizes esparsas e não nos preocupamos em escolher o melhor método para a busca unidirecional. Como os testes foram feitos com problemas pequenos não podemos afirmar que os problemas práticos relativamente grandes sejam resolvidos da mesma forma. Acreditamos, entretanto, que o comportamento dos algoritmos testados sejam semelhantes para a resolução de grandes problemas, ou seja, a tendência é que o algoritmo que se comportou com desempenho fraco na resolução de pequenos problemas não melhorará na resolução de grandes problemas. Em uma

[1] Borland International, Inc.

implementação de grande porte deve ser dada a preferência às metologias que se mostraram eficientes nestes problemas.

V.2 Geração dos Dados

Geramos 20 problemas aleatórios sem degenerações no formato primal. A matriz de restrições é de tamanho 20 x 30 e a densidade é de 30%. O problema possui o conjunto de soluções viáveis limitado e não vazio. O gerador [1] nos fornece um ponto viável inicial $x^0 = (1, \ldots, 1)$.

Para gerar um PPL fornecemos as seguintes características:

- número de variáveis e rtestrições
- densidade da matriz
- número de degenerações primais e duais
- valores mínimo e máximo para
  - coeficientes da matriz
  - pontos ótimos primais duais
  - folgas no ponto ótimo
- precisão

O gerador coloca os dados do PPL em dois arquivos compatíveis com os nossos algoritmos, de acordo com as características especificadas. Um arquivo contém o PPL e o outro contém uma solução interior viável para o problema.

Após gerar os problemas de teste foi necessário encontrar um ponto próximo a trajetória central que seria usado como ponto inicial. Com um ponto próximo a trajetória central tornamos possível o cálculo da folga dual viável associada a este ponto possibilitando utilização dos algoritmos primais-duais.

Para não colocarmos em desvantagem os problemas primais em relação aos primais-duais iniciamos a execução de ambos com o mesmo ponto, ou seja, com o ponto próximo a um ponto central. Para encontrar o ponto inicial utilizamos o algoritmo primal com iterações internas e paramos a execução ao encontrar o primeiro ponto com proximidade inferior a 0.1. O α inicial usado

é obtido pelo cálculo do parametro de penalidade usado pelos algoritmos sem aceleração. O número de iterações necessárias para chegar ao ponto desejado esta escrito na tabela.

Problemas	Iterações	Problemas	Iterações
L1	7	L11	9
L2	7	L12	7
L3	8	L13	7
L4	7	L14	7
L5	7	L15	7
L6	7	L16	7
L7	7	L17	7
L8	6	L18	7
L9	6	L19	7
L10	7	L20	7

Podemos dizer que se resolvesemos o PPL com o algoritmo primal com iterações internas usando o critério de proximidade 0.1 teríamos o número de iterações da tabela para encontrar o primeiro ponto próximo à trajetória. Mais adiante voltaremos a falar sobre o ponto inicial x^0.

Concluímos a discussão sobre os problemas a serem resolvidos e passaremos a comentar os resultados.

V.3 Resultados Obtidos

Nós apresentaremos os resultados divididos em itens onde em cada item trataremos de um fator importante que influenciou nossos testes e em um último item apresentaremos os melhores resultados. Os algoritmos iniciam a execução com um ponto x próximo a um ponto central $x(\alpha)$, com $\delta < 1$. O critério de parada é o descrito nos algoritmos anteriores para uma precisão 10^{-5}. Vejamos os itens:

a) Iterações Internas

Este é um recurso muito bom teoricamente, utilizado para demonstrar convergência, mas nas execuções dos algoritmos não resolveu os problemas de forma muito eficiente. Vejamos os resultados obtidos pelo algoritmo primal

com iterações internas e sem iterações internas e para o algoritmo primal-dual com iterações internas e sem iterações internas.

Prob.	PT	PR	PDT	PDR
L1	22	16	14	10
L2	21	16	14	11
L3	22	16	13	10
L4	21	17	13	10
L5	22	17	14	11
L6	22	16	15	12
L7	22	16	15	11
L8	22	17	14	12
L9	21	16	13	10
L10	21	16	13	10
L11	22	16	14	11
L12	21	16	15	11
L13	22	16	15	11
L14	21	17	13	11
L15	21	16	13	10
L16	22	16	13	11
L17	21	16	13	10
L18	22	16	13	11
L19	22	17	13	11
L20	22	16	13	10
Média	21.6	16.25	13.65	10.7
Melhor	21	16	13	10
Pior	22	17	15	12

Devido a este comportamento nós concluímos que para problemas pequenos uma implementação prática funciona melhor sem iterações internas, ou seja, sem nos preocuparmos em gerar pontos muito próximos a pontos centrais, principalmente quando os pontos estão se aproximando do ótimo onde convém dar passos mais longos. Na medida em que executamos várias iterações para permanecer próximos à trajetória central estamos fazendo iterações com pouca redução de custo, ou seja, sem muito valor prático na resolução geral do nosso problema. Esses resultados, como vimos foram válidos tanto para algoritmos primais como para algoritmos primais-duais.

b) Aceleração

Primeiramente falaremos da aceleração feita nos algoritmos com iterações in-

ternas PTA_3 e PTA_32. Esse recurso que usamos para tentar melhorar nosso algoritmo com iterações internas não trouxe nenhum benefício ao mesmo. Em alguns problemas essa aceleração aumentou o número total de iterações para a resolução do PPL. Portanto, não apresentamos os resultados, pois além do algorimo com iterações internas ter se mostrado ineficiente como vimos anteriormente a aceleração dele não apresentou nenhuma melhora.

Agora analisaremos os resultados para o segundo método de aceleração onde o algoritmo é acelerado nas iterações que $\frac{g^{k+1}}{g^k} \geq \sqrt{\mu}$. Na tabela estão os resultados para o algoritmo primal com busca unidirecional com e sem aceleração e o algoritmo primal-dual com e sem aceleração.

Prob.	PR	PRA	PDR	PDRA
L1	16	15	10	8
L2	16	16	11	9
L3	16	15	10	8
L4	16	15	10	8
L5	17	16	11	9
L6	17	16	12	9
L7	16	16	11	8
L8	17	16	12	10
L9	16	15	10	7
L10	16	15	10	8
L11	16	16	11	9
L12	16	16	11	9
L13	16	16	11	9
L14	17	16	11	9
L15	16	15	10	7
L16	16	16	11	9
L17	16	15	10	8
L18	16	15	11	8
L19	17	16	11	9
L20	16	16	10	8
Média	16.5	15.6	10.7	8.45
Melhor	16	15	10	7
Pior	17	16	12	10

Conforme podemos ver impresso na tabela a aceleração mostrou-se eficiênte nos algoritmos sem iterações internas. Como os resultados foram muito bons, testamos também para os algoritmos com iterações internas os quais tiveram

uma pequena melhora, mas não muito significativa.

c) Busca Bidirecional

Ao falarmos em fazer a melhor combinação linear das direções c_p e e_p para encontrar a direção de minimização da função penalizada $f_{a^k}(\cdot)$ parece que estamos resolvendo bem o nosso problema. Na prática isso foi comprovado. Vejamos os resultados para os algoritmos primais que apresentaram melhor desempenho até aqui, sem iterações internas. A tabela apresenta a comparação da busca unidirecional e bidirecional incluindo os algoritmos com aceleração que mostraram um bom desmpenho.

Prob.	PR	PR2	PRA	PRA2
L1	16	11	15	9
L2	16	11	16	9
L3	16	11	15	9
L4	16	12	15	10
L5	17	12	16	10
L6	17	13	16	12
L7	16	11	16	9
L8	17	12	16	10
L9	16	12	15	9
L10	16	12	15	11
L11	16	12	16	11
L12	16	12	16	11
L13	16	12	16	10
L14	17	11	16	9
L15	16	11	15	9
L16	16	11	16	9
L17	16	11	15	10
L18	16	11	15	9
L19	17	12	16	10
L20	16	11	16	9
Média	16.25	11.55	15.6	9.75
Melhor	16	11	15	9
Pior	17	13	16	12

Os cálculos feitos para a busca bidirecional são insignificantes comparados aos cálculos da projeção da matriz de restrições. A busca bidirecional é relativamente rápida, vale resaltar que foi resolvida sempre com poucos passos Newton-Raphson (1 a 3). A busca bidirecional reduziu bastante o número de

iterações e consequentemente o número de projeções da matriz de restrições conforme mostramos na tabela, com isso os resultados são favoráveis ao uso da busca bidirecional. Nos algoritmos com iterações internas e sem aceleração a busca bidirecional melhorou os resultados de maneira geral, mas isso não ocorreu para todos os problemas, ou seja, melhorou os resultados para a maioria dos problemas e piorou para outros. Já os algoritmos com iterações internas e com aceleração apresentaram bons resultados para os dois tipos de aceleração usados, mesmo assim, os resultados deixam a desejar quando comparados aos algoritmos sem iterações internas.

d) Duas Buscas e uma Mudança de Escala

A tentativa de reduzir o número de inversões de matrizes foi inútil. Para os algoritmos com iterações internas os resultados para uma busca foram os mesmos de duas buscas. Na maioria das execuções a segunda busca deu um passo insignificante e em outras vezes esse passo, embora pequeno, afastou o ponto da trajetória central piorando o desempenho do algoritmo. No algoritmo sem iterações internas e com aceleração os resultados foram semelhantes aos do algoritmo com iterações internas. Uma pequena melhora, mas não significativa tivemos no algoritmo com duas buscas bidirecionais, sem iterções internas e com aceleração. Devido ao desempenho fraco das duas buscas implementamos estes algoritmos só para alguns casos e não achamos necessário expor os resultados.

e) Primal-Dual

Como vimos na tabela do item (a) e (b) o método primal-dual superou o primal em eficiência. Realmente a informação da folga dual viável ajudou na resolução dos nossos problemas, isso se verifica em todos os resultados.

f) Ponto Inicial x^0

Até aqui não fizemos boas referências a algoritmos com iterações internas, o que nos mostra que não é bom tentarmos gerar pontos muito próximos a trajetória central. Por outro lado, nós devemos ter cuidado para não gerar pontos muito distantes da trajetória central o que pode permitir ao nosso

algoritmo um comportamento ruim, pois os pontos distantes da trajetória central não tem as mesmas propriedades teóricas e nesse caso o parâmetro de penalidade pode decrescer pondo em risco a convergência do algoritmo. É importante salientar que os nossos resultados são provenientes de um ponto inicial próximo a trajetória central (com proximidade inferior a 0.1). Aqui fica a dúvida sobre o comportamento dos algoritmos sem iterações internas com um ponto inicial qualquer. Para tirar essa dúvida executamos alguns algoritmos com o ponto inicial fornecido pelo gerador e comparamos com a centralização mais o algoritmo. Para isso, escolhemos os algoritmos que tem apresentado o melhor desempenho PRA e PRA2. Não é possível executarmos os algoritmos primais-duais porque só podemos calcular a folga dual viável inicial para um ponto próximo a trajetória central. Vejamos os resultados para os algoritmos primais.

Prob.	PRA	PRA2
L1	25	15
L2	28	19
L3	26	14
L4	29	17
L5	26	12
L6	24	14
L7	22	13
L8	27	14
L9	23	17
L10	25	16
L11	25	17
L12	31	19
L13	24	14
L14	29	21
L15	24	17
L16	26	14
L17	23	13
L18	25	14
L19	25	15
L20	25	15
Média	25.6	15.5
Melhor	22	13
Pior	31	19

Na resolução dos problemas acima tomamos o cuidado de não deixar o parâ-

metro de penalidade decrescer: quando o parâmetro tentou decrescer nós o mantivemos com o valor da iteração anterior. Podemos ver na tabela que o algoritmo sem iterações internas com a busca unidirecional apresenta problemas, basta ver que o L12 pode ser resolvido pelo algoritmo com iterações internas iniciando no mesmo ponto com um número inferior de iterações. Somamos as iterações do PT (21) com as iterações necessárias para encontrar o x^0 próximo a $x(\alpha)$ do problema L12 (7) e teremos 28 iterações.

Na busca bidirecional não ocorreu um problema tão evidente, mas vimos que o comportamento variou com os problemas e alguns resultados não foram muito bons. Com isso, podemos dizer que o ponto inicial interfere no desempenho dos algoritmos sem iterações internas.

g) Multiplicador Inicial do Parâmetro de Penalidade (μ^0)

Nos testes feitos até aqui usamos um $\mu^0 = 10$ (ou seja, $\alpha^{k+1} = 10\alpha^k$). Para o estudo do μ nós vamos usar gráficos tentando mostrar o efeito do μ^0 (valor inicial) sobre os nossos algoritmos. Não colocamos os resultados em tabelas por ser um número muito grande de execuções diferentes o que envolveria muitas tabelas e ficaria mais complicada a visualização. Os gráficos permitem uma visualização melhor, mas envolvem os resultados de um único problema. Colocamos os resultados de um determinado problema e falamos dos resultados gerais, ou seja, dos resultados de todos os problemas de maneira geral. Desconsideramos o caso da diferença de uma única iteração final, analisamos a curva feita e o número de iterações em que μ se manteve melhor na resolução, pois sabemos que um μ grande pode ser muito eficiente em uma última iteração e com isso reduzir bem o número de iterações com esse passo grande. Esses passos grandes com um μ muito grande podem, as vezes, determinar iterações semelhantes às do afim-escala com congestionamento próximo à fronteira. Isso não ocorreu no exemplo dos gráficos expostos, mas citamos os casos em que ocorreram problemas, como por exemplo o algoritmo com iterações internas e busca bidirecional.

Vamos tratar de cada algoritmo separadamente, iniciamos com os algoritmos primais sem aceleração e com iterações internas para:

- Busca unidirecional: Um μ pequeno demais pode determinar passos de tamanho reduzido deixando o algoritmo ineficiente conforme mostra o gráfico V.1. Para μ acima de um certo valor a convergência não depende muito do μ. Para μ menor temos algumas iterações pouco eficientes quando estamos com uma proximidade grande $\delta \cong 1$. Para μ grande esse fenômeno ocorre poucas vezes.
- Busca bidirecional: Um μ pequeno demais ocorre o problema do caso anterior conforme mostra o gráfico V.2. O μ acima de um certo valor não nos diz muita coisa em alguns problemas ajudou mais $\mu = 10$ do que $\mu = 1000$.

  Vejamos os algoritmos sem iterações internas:

- Sem aceleração: Para a busca unidirecional (gráfico V.3) e bidirecional (gráfico V.4) o comportamento foi semelhante. A inclinação da curva $\frac{a^{k+1}}{a^k}$ depende do μ para um μ abaixo de um certo valor. Se μ é muito pequeno o comportamento dos algoritmos é ruim.
- Com aceleração: Depende pouco do μ^0. A aceleração determina o formato da curva, conforme mostra o gráfico, isso ocorre na busca unidirecional (gráfico V.5) e bidirecional (gráfico V.6).

Nos algoritmos primais-duais o μ pequeno demais segue um comportamento semelhante aos algoritmos primais. Um μ muito pequeno deixa o algoritmo ineficiente (gráficoV.7), isso não ocorre se usarmos aceleração (gráficoV.8). Como a direção primal-dual é boa os algoritmos tiveram bons resultados para μ grande.

h) Melhores Resultados

Os algoritmos que parecem conseguir resolver os problelmas da melhor maneira de acordo com os resultados obtidos foram estes:

Prob.	PRA2	PDRA
L1	9	8
L2	9	9
L3	9	8
L4	10	8
L5	10	9
L6	12	9
L7	9	8
L8	10	10
L9	9	7
L10	11	8
L11	11	9
L12	11	9
L13	10	9
L14	9	9
L15	9	7
L16	9	9
L17	10	8
L18	9	8
L19	10	9
L20	9	8
Média	9.75	8.45
Melhor	9	7
Pior	12	10

Podemos dizer que merece atenção as seguintes características que determinaram a eficiência dos algoritmos acima:

- O método primal-dual
- A aceleração
- A busca bidirecional
- Um ponto inicial x^0 próximo a um ponto central $x(\alpha)$
- Um multiplicador inicial μ^0 para o parâmetro de penalidade não muito grande

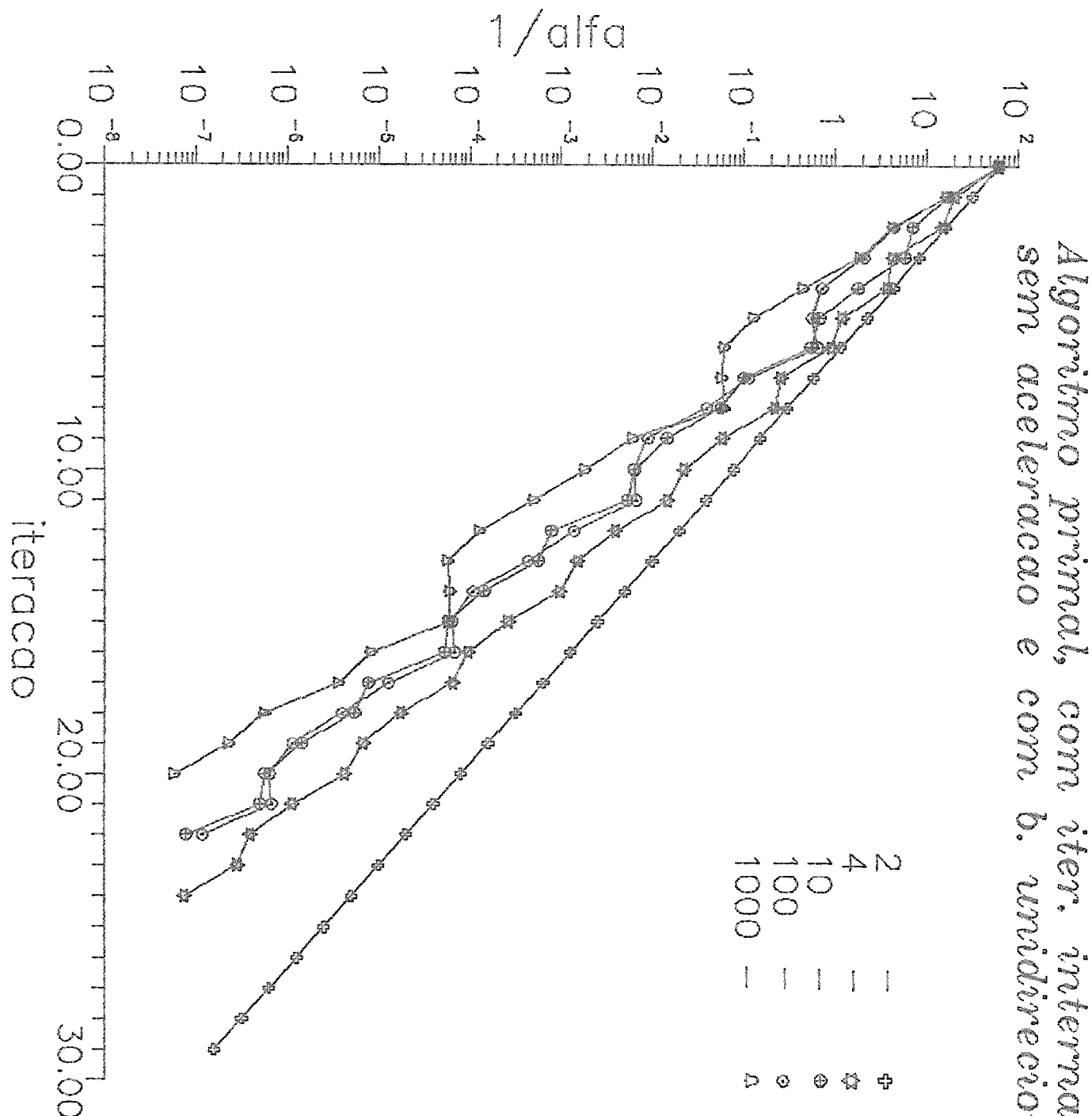

Figura V.1: Gráfico do algoritmo PT

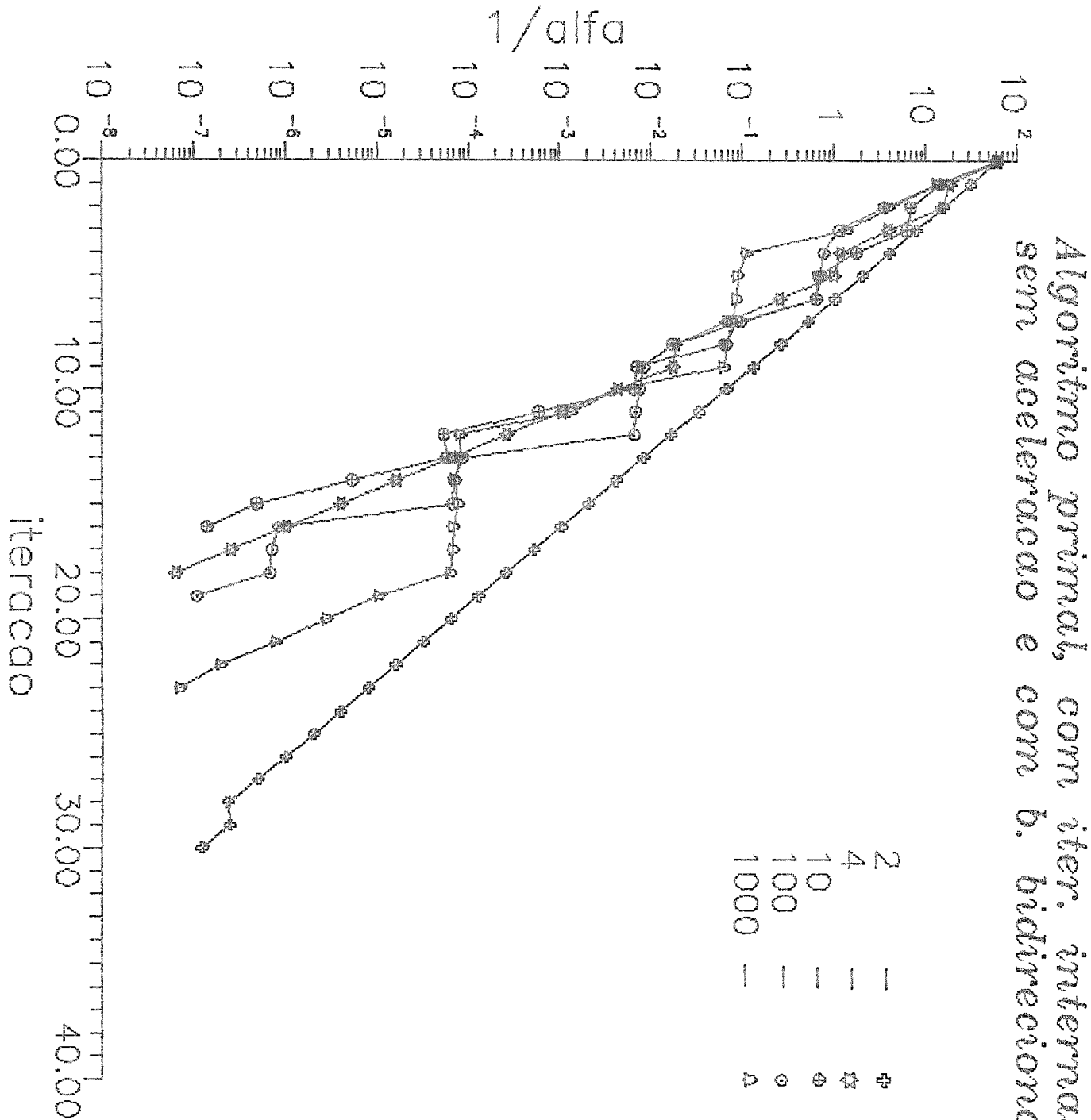

Figura V.2: Gráfico do algoritmo PT2.

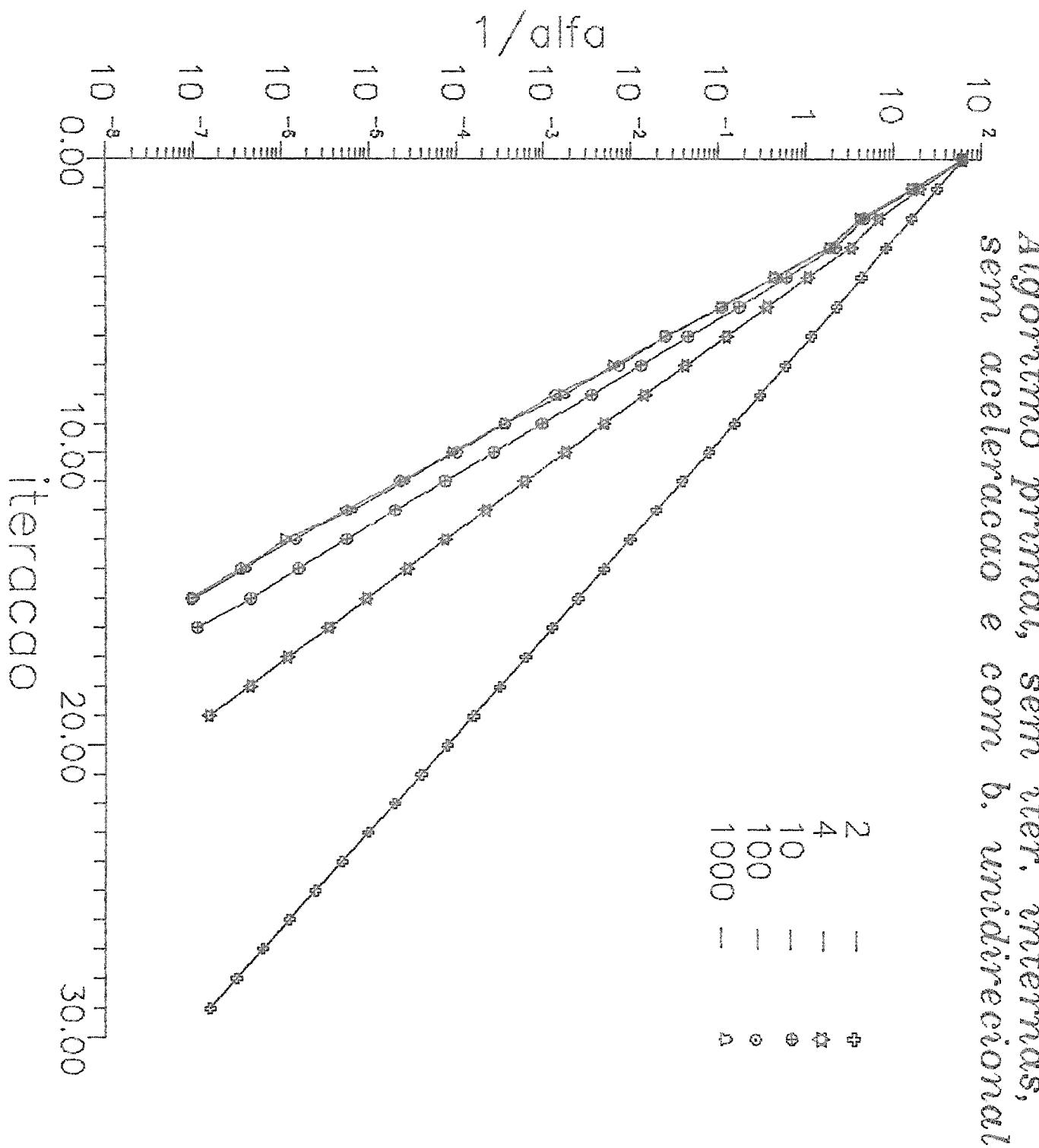

Figura V.3: Gráfico do algoritmo PR.

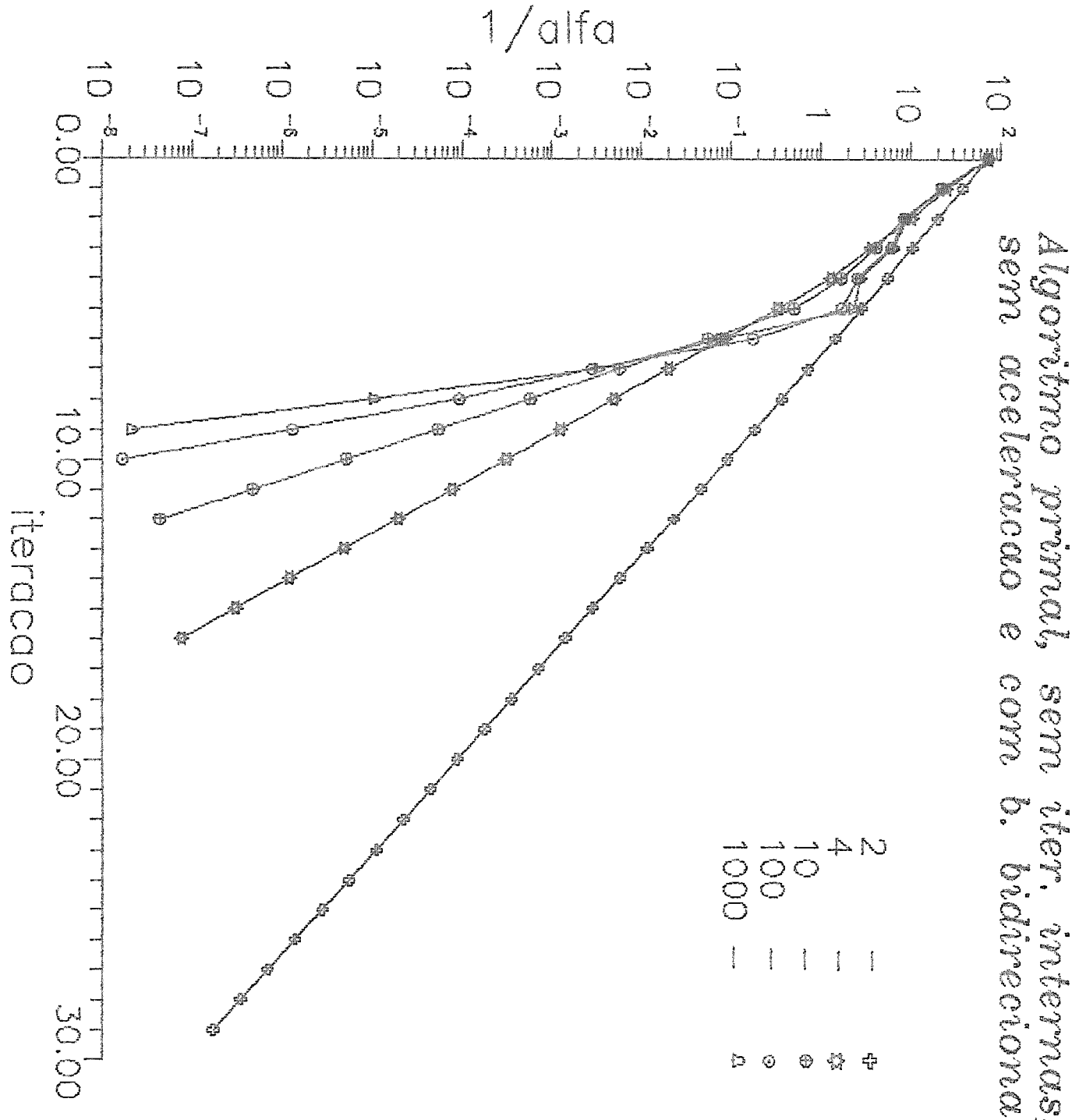

Figura V.4: Gráfico do algoritmo PR2.

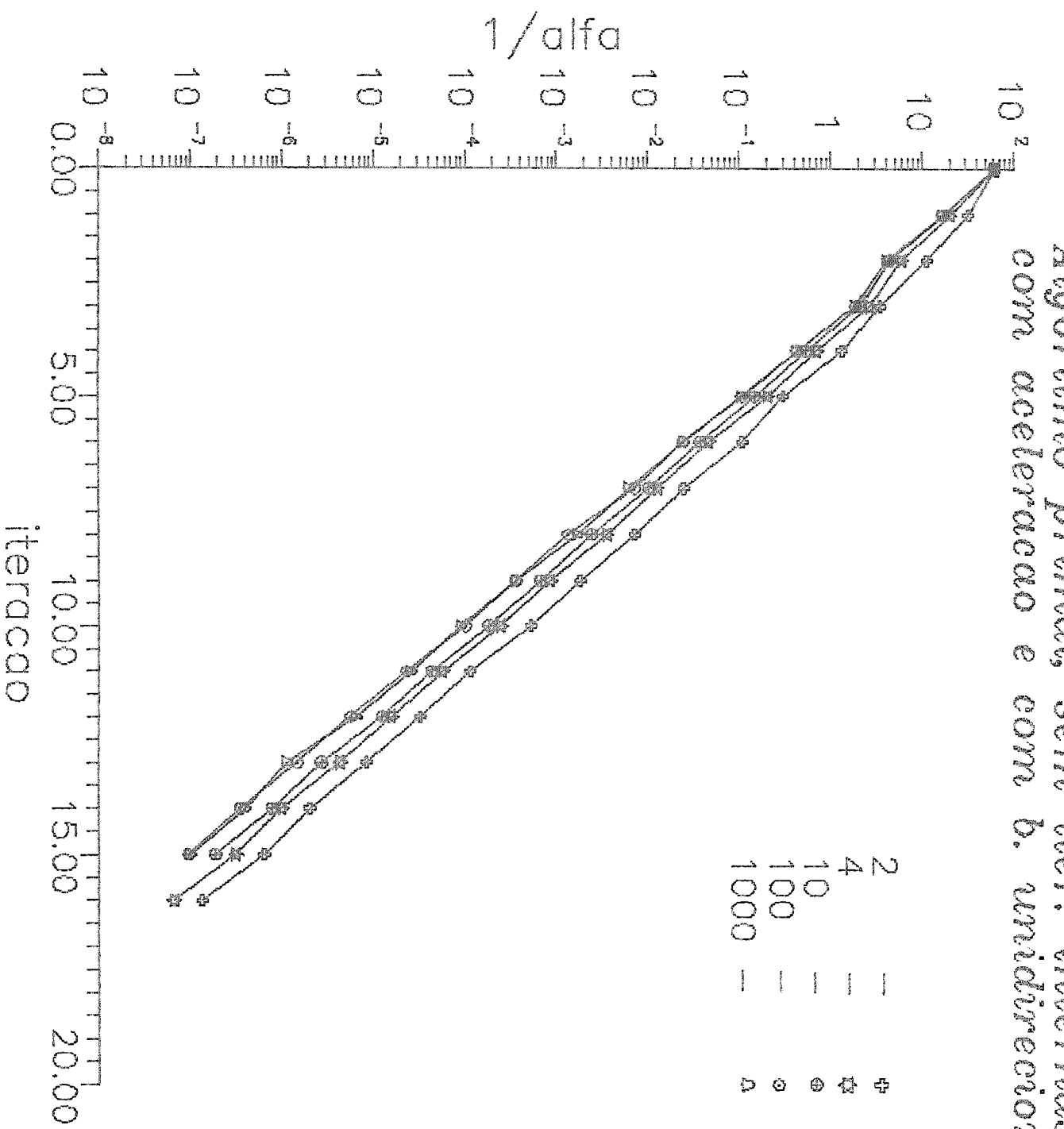

Figura V.5: Gráfico do algoritmo PRA.

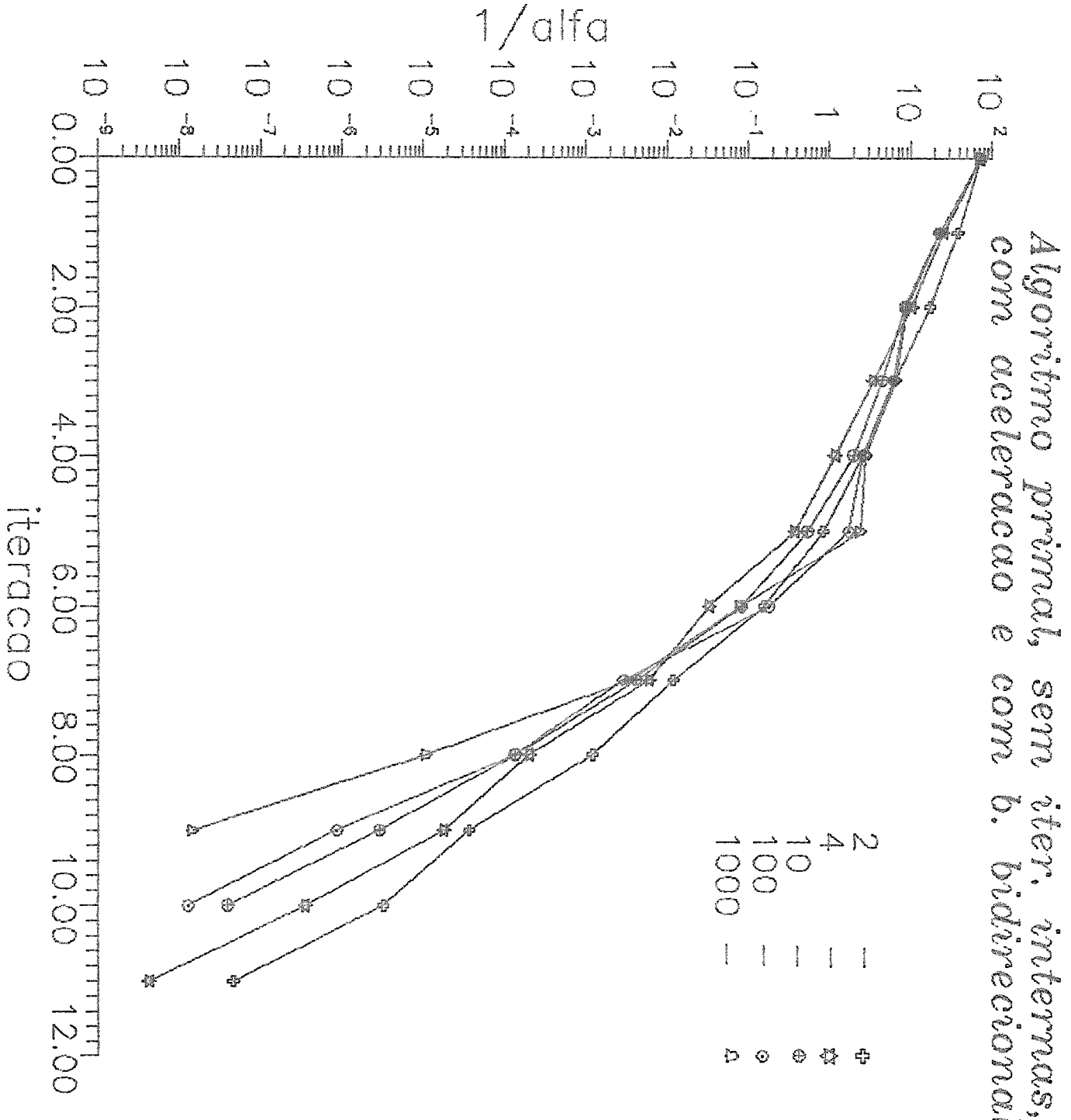

Figura V.6: Gráfico do algoritmo PRA2.

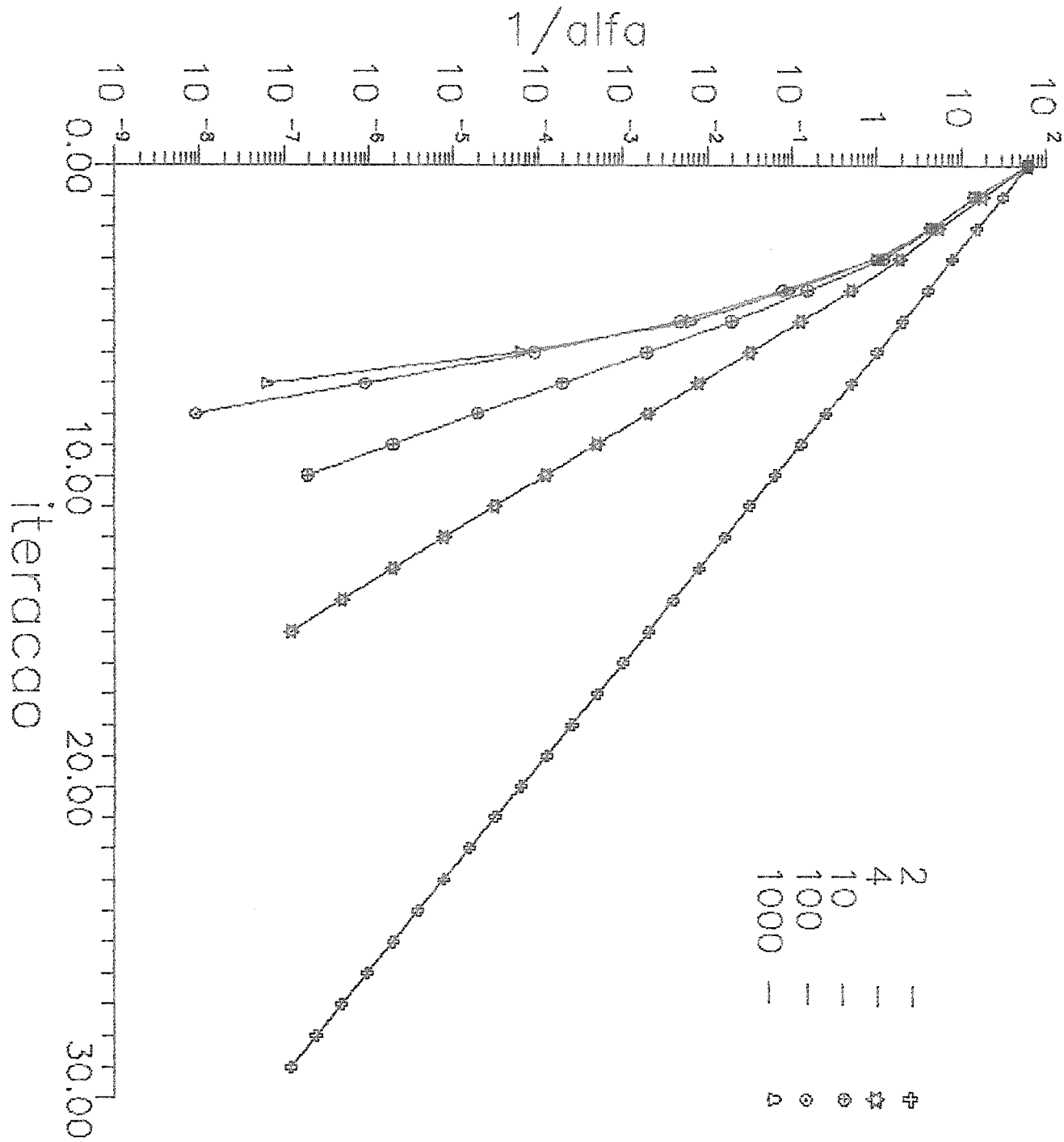

Figura V.7: Gráfico do algoritmo PDR.

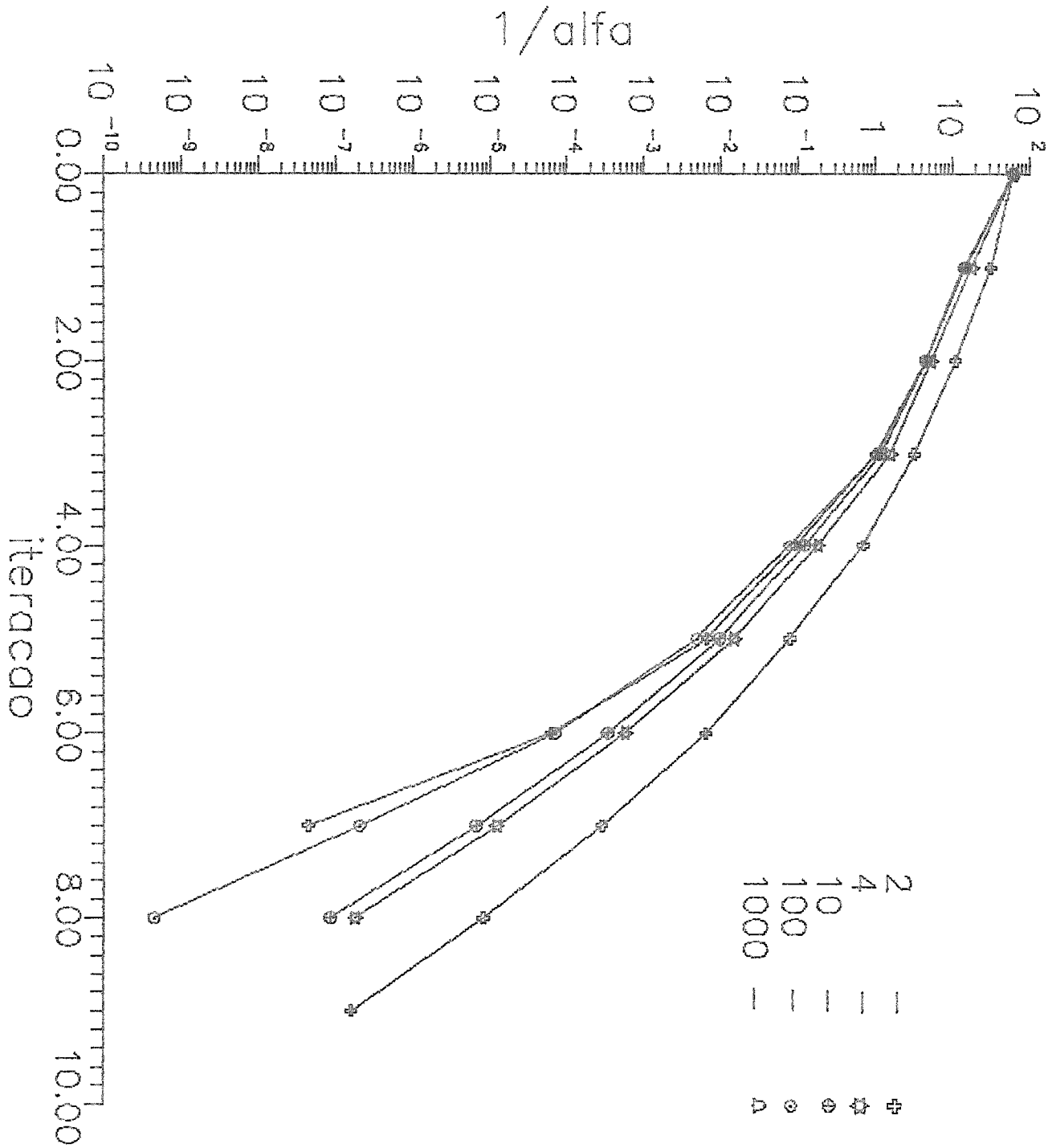

Figura V.8: Gráfico do algoritmo PDRA.

Capítulo VI

CONCLUSÃO

Nós agora apresentaremos as conclusões obtidas a partir dos testes realizados para este trabalho.

Inicialmente concluímos que os algoritmos com iterações internas perdem tempo tentendo permanecer próximos à trajetória central, portanto achamos que os algoritmos sem iterações internas tem tendência a resolver melhor os problemas. Os algoritmos sem iterações internas devem começarar a execução com um ponto próximo à trajetória central, caso contrário, o funcionamento pode ser prejudicado deixando-o em certos casos mais lento que o algoritmo com iteraçoes internas. Não devemos permitir que o algoritmo sem iterações internas gere pontos que se afastem demais da trajetória central de modo que no cálculo da penalidade o parâmetro de penalidade seja reduzido, neste caso devemos fazer com que o parâmetro de penalidade permaneça com o valor da iteração anterior, com isso estamos forçando o nosso algoritmo a aproximar-se da trajetória central.

A aceleração calculada para o algoritmo sem iteraçcões internas mostrou-se muito eficiente. O uso da aceleração permite iniciar o algoritmo com um valor de μ não muito grande, o que evita que os pontos se afastem da trajetória central. Um valor inicial muito grande para μ corresponde a passos muito longos, que podem reprodizir o comportamento do algoritmo afim-escala, com possíveis congestionamentos próximos da fronteira.

O tamanho ideal do passo $\alpha^{k+1} = \mu\alpha^k$ depende na realidade da curvatura da trajetória central, e sabe-se que essa curvatura é pequena no trecho final da

trajetória, para α grande. Isso justifica teoricamente o uso da aceleração.

Outro fator que mostrou um bom desempenho, além da aceleração foi a busca bidirecional que com o acréscimo de alguns cálculos rápidos resolveu os problemas reduzindo bem o número de iterações, a busca bidirecional demonstrou o desempenho melhor principalmente quando está nos pontos não muito distantes do ótimo, ou seja, nas iterações finais do algoritmo. Não temos como prever o efeito da busca bidirecional no tempo de competição para problemas de grande porte, mas nossas conclusões indicam que vale a pena tentar.

Finalmente falaremos do método primal-dual o qual também tem um acréscimo de cálculos em relação ao método primal. Além dos cálculos do método primal, nós acrescentamos ao primal-dual o cálculo referente a folga dual viável. Como não temos o acréscimo de inversão de matriz, pois as direções primal e dual são complementares o número de cálculos a mais feitos no método primal-dual não é tão relevante. Nos resultados a redução ocorrida no número de iterações deve compensar os cálculos feitos a mais.

Assim, julgamos que seria interessante uma implementação eficiente de grande porte que resolva um conjunto de problemas práticos para algoritmos sem iterações internas, verificando o efeito da aceleração nos métodos primal e primal-dual, ambos iniciando a execução em um ponto x^0 próximo à trajetória central.

Seria interessante comparar o tempo de execução para ver se realmente os cálculos da busca bidirecional não são significativos sendo compensados pelo número de iteraçoes reduzidas em problemas grandes. Pode também ser comparado o tempo gasto para os cálculos referentes ao método primal-dual verificando se a redução no número de iterações determinado pelo método é significativo.

Referências Bibliográficas

[1] R. Arantes and M. Tortorelli. *Implementação e Estudo Comparativo de Algoritmos de Pontos Interiores para Programação Linear.* Relatório Técnico, Pontifícia Universidade Católica, Rio de Janeiro, 1988.

[2] E. Barnes, D. Jensen, and Chopra. *A Polynomial-Time Version of the Affine-Scaling Algorithm.* Manuscript, New York University, New York, NY, 1988.

[3] G. Dantzig. Maximization of a linear function of variables subject to linear inequalities. In Tj. C. Koopmans, editor, *Activity Analysis of Production and Allocation*, pages 339–347, Wiley, N. York, 1951.

[4] I. I. Dikin. Iterative solution of problems of linear and quadratic programming. *Soviet Mathematics Doklady*, 8:674–675, 1967.

[5] I. I. Dikin. On the speed of an iterative process. *Upravlyaemye Sistemi*, 12:54–60, 1974.

[6] K. R. Frisch. *The Logarithmic Potential Method of Convex Programming.* Memorandum, University Institute of Economics, Oslo, Norway, 1955.

[7] C. Gonzaga. An algorithm for solving linear programming problems in $O(n^3L)$ operations. In N. Megiddo, editor, *Progress in Mathematical Programming – Interior Point and Related Methods*, chapter 1, Springer Verlag, Berlin, 1989.

[8] C. Gonzaga. *Algoritmos de Pontos Interiores para Programação Linear.* Instituto de Matemática Pura e Aplicada, Rio de Janeiro, 1889.

[9] C. Gonzaga. *Large-Steps Path-Following Algorithms for Linear Programming: Barrier Function Method.* Internal report , COPPE – Federal

University of Rio de Janeiro, Rio de Janeiro, Brasil, July 1989. To appear in *Siam Journal on Optimization.*

[10] C. Gonzaga. *Path Following Methods for Linear Programming.* Manuscript, COPPE – Federal University of Rio de Janeiro, Rio de Janeiro, Brasil, 1990. To appear in *SIAM Review.*

[11] C. Gonzaga. *Search Directions for Interior Linear Programming Methods.* Memorandum UCB/ERL M87/44, Electronics Research Laboratory, University of California, Berkeley, CA, March 1987.

[12] N. Karmarkar. A new polynomial time algorithm for linear programming. *Combinatorica,* 4:373–395, 1984.

[13] L. G. Khachiyan. A polynomial algorithm for linear programming. *Soviet Mathematics Doklady,* 20:191–194, 1979.

[14] V. Klee and G. Minty. How good is the simplex algorithm? In O. Sisha, editor, *Inequalities III,* Academic Press, New York, NY, 1972.

[15] J. Renegar. A polynomial-time algorithm based on Newton's method for linear programming. *Mathematical Programming,* 40:59–94, 1988.

[16] G. Sonnevend. An analytical centre for polyhedrons and new classes of global algorithms for linear (smooth, convex) programming. In *Lecture Notes in Control and Information Sciences 84,* pages 866–876, Springer Verlag, New York, NY, 1985.

[17] M. Tortorelli. *Algoritmos de Trajetória Central para Programação Linear.* Master's thesis, PUC, Rio de Janeiro, Brasil, 1991.

[18] P. Vaidya. *An Algorithm for Linear Programming which Requires* $O(((m+n)n^2 + (m+n)^{1.5}n)L)$ *Arithmetic Operations.* Preprint, AT&T Bell Laboratories, Murray Hill, NJ, 1987.