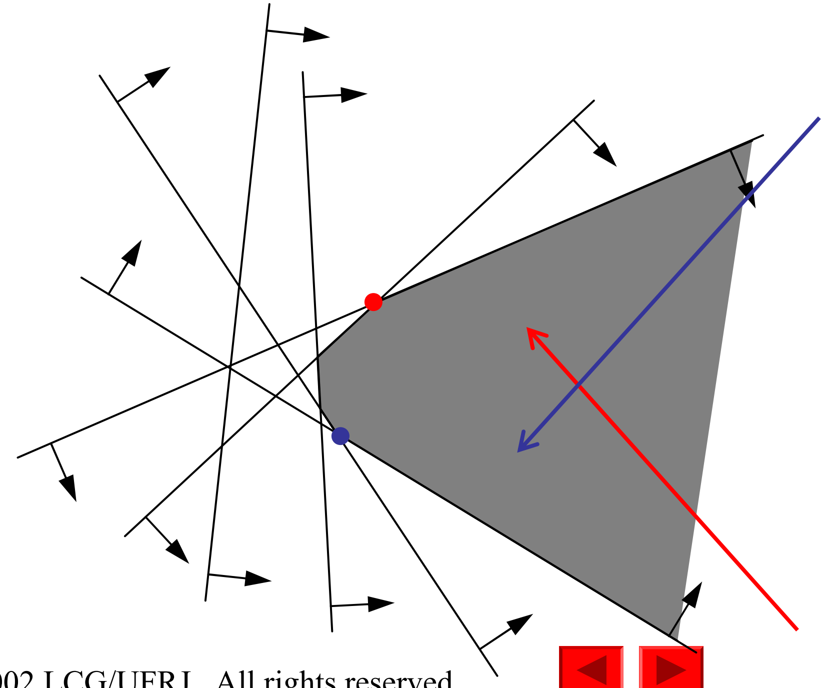
Programação Linear

Claudio Esperança
Paulo Roma

Programação Linear

- Vimos algoritmos para computar a interseção de n semiplanos em tempo (ótimo) $O(n \log n)$
- Muitas vezes não precisamos computar o politopo completo mas apenas um ponto extremo em uma dada direção



Programação Linear

- A programação linear se ocupa com esse tipo de problema que formalmente pode ser expresso como
  - Maximizar

$$c_1x_1 + c_2x_2 + \ldots + c_dx_d$$

Função Objetivo

  - Sujeito a

$$\left.\begin{array}{l} a_{1,1}x_1 + a_{1,2}x_2 + \ldots + a_{1,d}x_d \le b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \ldots + a_{2,d}x_d \le b_2 \\ \ldots \\ a_{n,1}x_1 + a_{n,2}x_2 + \ldots + a_{n,d}x_d \le b_n \end{array}\right\}$$

Restrições
(politopo *realizável*)

- O politopo formado pelas restrições é chamado de polígono / poliedro *realizável*
- A solução nem sempre existe ou é única
  - Quando existe, a solução é chamada de *vértice ótimo*

Programação Linear

- Observe que o problema em geral é definido num espaço d-dimensional e que podemos pensar na função objetivo como sendo um vetor $c = [c_1 \;\; c_2 \; \ldots \; c_d]^T$
  - Em 2D, é comum expressar o problema de tal forma que o vetor c aponta para baixo
    - Solução é o vértice do politopo com menor coordenada y
- Programação linear é uma das formulações mais importantes de problemas de otimização

Métodos de Solução

- Para *d* alto, os métodos mais comuns são
  - Método *simplex*
    - Acha-se um ponto na fronteira do politopo realizável
    - Caminha-se pela fronteira até encontrar o vértice ótimo
      - Caminho é feito *simplex* a *simplex,* ex.: por arestas adjacentes em 2D ou por triângulos adjacentes em 3D
  - Métodos de *pontos interiores*
    - Ao invés de caminhar pela fronteira, caminha-se pelo interior da região realizável

Complexidade

- Durante muito tempo houve dúvidas se problemas de programação linear eram polinomiais
  - Há casos conhecidos em que o método simplex roda em tempo exponencial
- Em 1984, Karmarkar mostrou um algoritmo polinomial de pontos interiores
  - Na verdade, polinomial com relação a *n* (número de restrições), *d* (dimensão) e o *número de bits* usados na representação das coordenadas
- Um problema ainda em aberto é se há um algoritmo *fortemente polinomial* para programação linear, i.e., que não dependa da precisão dos números

Programação Linear no Plano

- Em 2D conhecemos vários algoritmos para computar a interseção de semiplanos em $O(n \log n)$
  - Computar o vértice ótimo, portanto, requer apenas fazer uma busca seqüencial pela fronteira do polígono realizável em tempo $O(n)$
- Pode-se resolver o problema sem computar o polígono em tempo $O(n)$
- Na verdade, se estipularmos uma dimensão fixa, existem algoritmos que resolvem o problema em tempo $O(n)$
  - Na prática, entretanto, tais algoritmos rodam em tempo exponencial com relação à dimensão
  - Aceitáveis apenas para dimensões baixas

Programação Linear no Plano

- Os dados do problema são
  - Um conjunto de n inequações na forma

$$a_{i,x}x + a_{i,y}y \leq b_i$$

  - Uma função objetivo dada por um vetor

$$c = (c_x, c_y)$$

- Busca-se encontrar um ponto $p = (p_x, p_y)$ na fronteira do polígono realizável e que maximize o produto escalar dado por

$$c_x p_x + c_y p_y$$

  - Em nossos exemplos vamos assumir que o vetor da função objetivo é (0, -1) e portanto o ponto ótimo é o ponto do polígono realizável com menor coordenada y

Construção Incremental

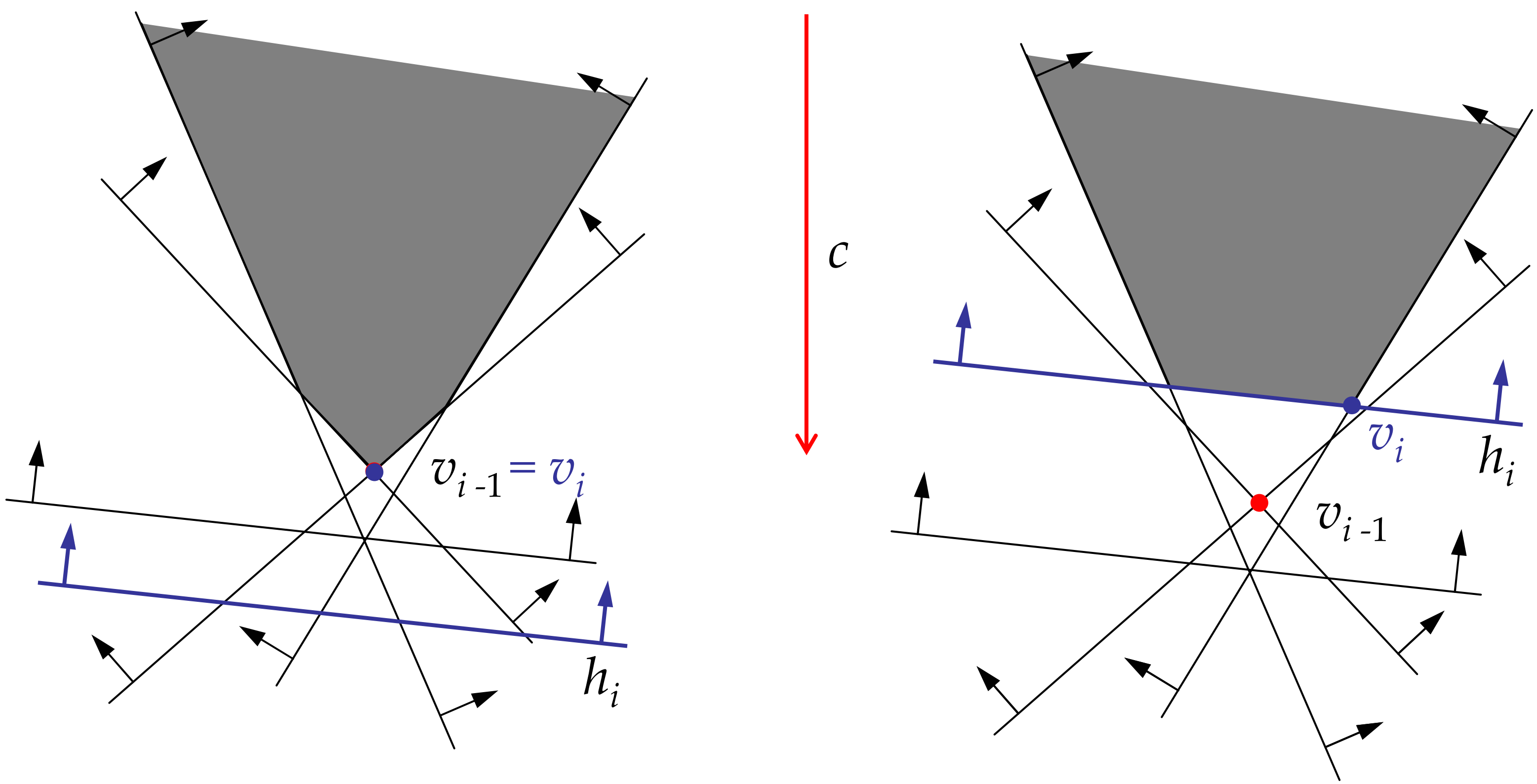
- Ao lado da técnica de varredura, a construção incremental é um modelo muito usado para a construção de algoritmos em geometria computacional
- A idéia geral é construir a solução para poucos dados e acrescentar um dado de cada vez tentando observar como a solução do problema se comporta

Construção Incremental

- No nosso caso, vamos acrescentar um semiplano (restrição) por vez e observar como isso afeta o vértice ótimo do polígono
- Precisamos começar com uma solução válida:
  - Em 2D, podemos escolher um par de semiplanos, digamos h_1 e h_2, cujo ponto de interseção seja ótimo
  - Em d dimensões, precisamos de d semiplanos
- Supondo que conhecemos a solução v_{i-1} que leva em conta os i - 1 primeiros semiplanos, o que pode acontecer quando acrescentamos o i'ésimo semiplano h_i?

Construção Incremental

- Existem duas possibilidades:

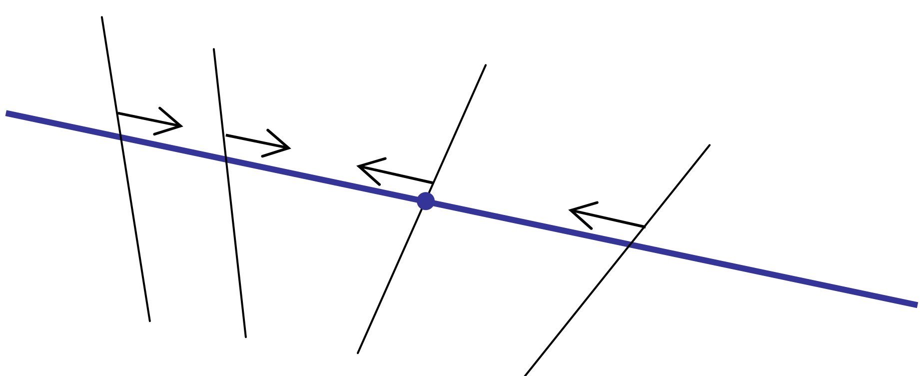


Caso 1 : $v_{i\text{-}1}$ satisfaz h_i

Caso 2 : $v_{i\text{-}1}$ não satisfaz h_i

Construção Incremental

- Testar v_{i-1} com respeito ao semiplano h_i é trivial
- O caso 1 não oferece dificuldade, já que o ponto ótimo não se altera ($v_i = v_{i-1}$)
- No caso 2, o novo ponto ótimo v_i se encontra sobre a reta de suporte de h_i
  - Basta intersectar todos os semiplanos vistos até o momento com a reta de suporte de h_i
  - O processo se dá em 1D
  - A interseção de cada semiplano com a reta resulta num intervalo semi-infinito
  - A interseção de todos os intervalos pode ser feita facilmente em $O(n)$



Generalização para nD

- Observamos que a solução do problema em 2D recai na solução de um problema 1D, que é fácil de resolver
- O artifício é semelhante ao utilizado na técnica de varredura
- Para resolver problemas de programação linear em dimensões mais altas, basta reduzir o problema a dimensões cada vez mais baixas

Análise de Complexidade

- O melhor caso claramente acontece sempre quando encontramos o caso 1
  - Temos então que o algoritmo executa em tempo linear, isto é, $O(n)$
- A complexidade de pior caso acontece sempre quando encontramos o caso 2
  - O fator dominante é computar o problema em 1D
  - Ao adicionar o semiplano h_i podemos ter que considerar os $i - 1$ semiplanos precedentes

$$\sum_{i=3}^{n}(i-1) \leq \sum_{i=1}^{n} i = \frac{n(n+1)}{2} = O(n^2)$$

Construção Incremental Randomizada

- É importante notar que o pior caso é bastante pessimista, já que requer que o vértice ótimo corrente *sempre* seja irrealizável na iteração seguinte
- Na verdade, demonstraremos que se os semiplanos forem acrescentados em uma ordem *verdadeiramente randômica*, o pior caso é bastante improvável
- Esta observação leva à versão randomizada do algoritmo, que roda em tempo esperado $O(n)$
- O algoritmo randomizado é quase idêntico ao determinístico. A única exceção é que o conjunto de semiplanos (restrições) é "embaralhado" antes de iniciar a construção incremental

Como embaralhar?

- Seja H [1 .. n] um array com os semiplanos e $rand(k)$ uma função que retorna um número aleatório entre 1 e k

proc *Embaralha* (H)

  para i desde n até 1 passo -1 fazer

    $j \leftarrow rand\ (i)$

    trocar $H\ [n] \leftrightarrow H\ [j]$

Análise do Algoritmo Randomizado

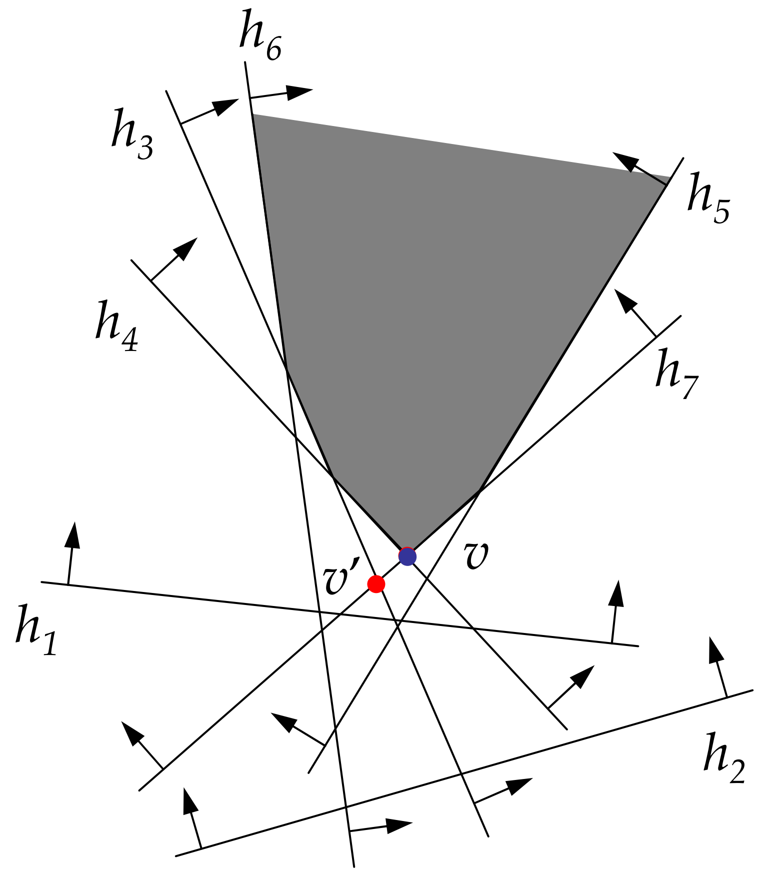
- Para obter o custo do caso médio, temos que analisar cada possível escolha feita pelo algoritmo
  - No nosso caso, a escolha do próximo semiplano
- A cada escolha é atribuída uma probabilidade de ocorrência
  - Em algoritmos randomizados, todas as escolhas têm a mesma probabilidade
- O impacto da escolha no custo do algoritmo tem que ser estimado
  - Esta é a principal dificuldade!
  - Cada escolha influencia o custo da próxima escolha
  - No nosso caso, teríamos que considerar n! possibilidades
    - Na verdade, $(n - 2)!$ possibilidades, já que a escolha dos 2 primeiros semiplanos é determinística

Análise "para trás"

- Na análise tradicional – ou "para a frente" – o raciocínio sempre acompanha o efeito da *próxima escolha* no tempo do algoritmo
- Na análise "para trás" (*backward analysis*) *o* raciocínio é voltado para o *passado*, isto é, qual o efeito da *última escolha* no progresso do algoritmo
  - Isto facilita a análise, uma vez que conhecemos o passado mas não o futuro

Análise "para trás" do algoritmo

- Examinemos o progresso do algoritmo após o acréscimo de 7 semiplanos
  - O vértice ótimo é formado por h_4 e h_7
  - Qual a influência do último semiplano inserido?
    - Se foi h_4 ou h_7 a inserção se deu em $O(n)$
      - Ex.: Se foi h_4, o vértice ótimo anterior era v'
    - Se foi qualquer outro, a inserção se deu em $O(1)$



Análise “para trás” do algoritmo

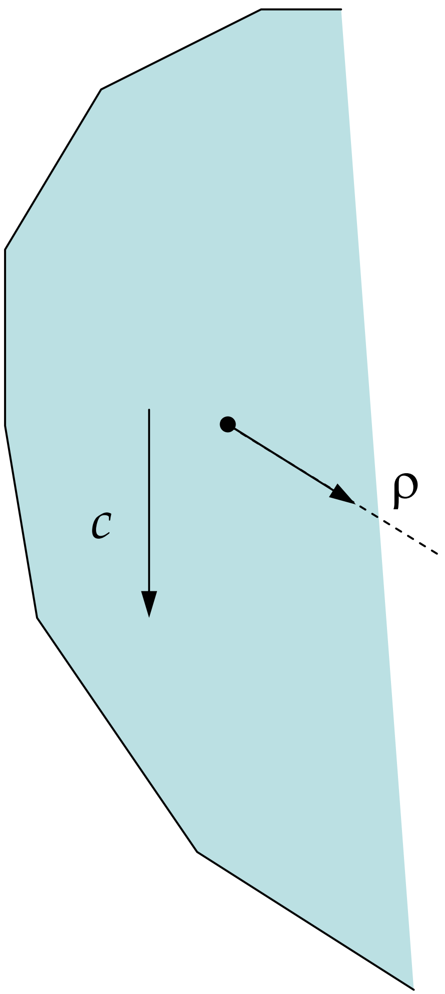
- Em geral, olhando para trás, a probabilidade da inserção do i'ésimo plano ter sido custosa é $2 / i$, já que somente dois dos i semiplanos constituem o vértice ótimo
- Os restantes $(i - 2)$ semiplanos têm custo constante
- Portanto, o custo médio da i'ésima inserção é dada por

$$\frac{2}{i} \cdot i + \frac{i-2}{i} \cdot 1 \leq 3 \in O(1)$$

- Somando os n estágios da construção incremental, chegamos ao custo médio total $O(n)$

Problemas Ilimitados

- Assumimos até agora que o problema de otimização tem uma solução limitada
- Existe um lema que permite determinar se um problema é ilimitado (*unbounded*) ou não
  - Um problema é ilimitado se podemos achar uma semi-reta ρ (raio) totalmente contido na região realizável cuja direção faz um ângulo agudo com *c*

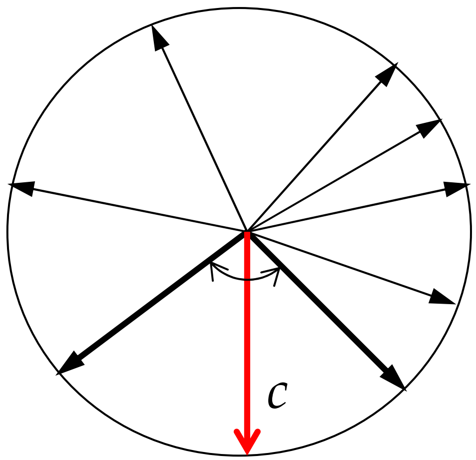


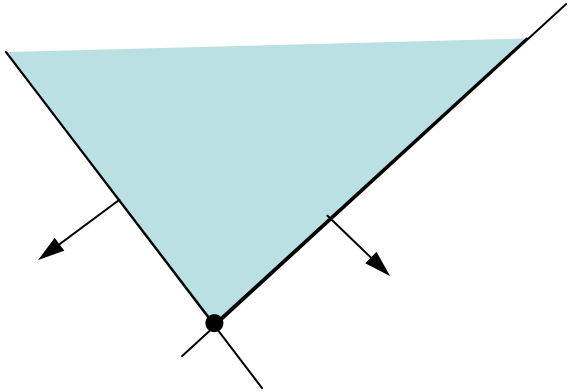
Problemas Ilimitados

- O livro mostra como encontrar ρ resolvendo um problema de programação linear de dimensão 1
  - Se o problema não puder ser resolvido (região realizável é vazia) então o problema original é realizável
  - Neste caso, como subproduto, o algoritmo encontra 2 semiplanos cuja interseção é o vértice ótimo inicial
  - Este método pode ser generalizado para qualquer dimensão

Encontrando 2 semiplanos iniciais

- Em 2D, existe uma maneira mais simples de determinar 2 semiplanos iniciais ou então descobrir que o problema é ilimitado
- Basta ordenar angularmente as normais (apontando para fora) de todos os semiplanos
  - Se as 2 normais mais próximas de c nos 2 sentidos fizerem entre si um ângulo < 180º, então os semiplanos correspondentes convergem num ponto p e a região de interseção entre eles contém o polígono realizável





Interseções com um hiperplano

- Uma operação freqüente em programação linear é computar a interseção de semiplanos com um hiperplano dado. Por exemplo,
  - Para computar o vértice ótimo inicial
  - Para computar o "caso 2" da construção incremental
- Um semiplano em *d* dimensões é dado por

  h_i: $a_{i,1}x_1 + a_{i,2}x_2 + \ldots + a_{i,d}x_d \leq b_i$
- O hiperplano correspondente é dado por

  $\ell_i : a_{i,1}x_1 + a_{i,2}x_2 + \ldots + a_{i,d}x_d = b_i$

Interseções com um hiperplano

- Podemos representar o conjunto de restrições matricialmente por

  $A\,x = b,\quad$ onde

  - A é a matriz $n \times d$ de coeficientes $a_{i,j}$
  - x é um vetor $d \times 1$ de incógnitas e
  - b é um vetor $d \times 1$ de termos independentes

- Assim, a equação de um hiperplano pode ser escrita

  $A_i\,x = b_i\,,\quad$ onde

  - A_i é a linha i da matriz A e
  - b_i é o i'ésimo termo independente

Interseções com um hiperplano

- Se queremos intersectar os demais semiplanos ℓ_j com ℓ_i, podemos reduzir as restrições a um problema de dimensão d–1 efetuando um passo do método de eliminação de Gauss
- Assumindo que o coeficiente $a_{i,1}$ seja não nulo, podemos eliminar a 1ª coluna da matriz
  - Se $a_{i,1} = 0$, escolhe-se outra coluna
  - Para cada semiplano ℓ_j:

$$A'_j = A_j - \left(\frac{a_{j,1}}{a_{i,1}} \right) A_i$$

$$b'_j = b_j - \left(\frac{a_{j,1}}{a_{i,1}} \right) b_j$$

Interseções com um hiperplano

- A eliminação de uma coluna tem o significado de reduzir as restrições ao hiperplano ℓ_i
  - É fácil ver que o ponto x satisfaz $A\,x = b$ se e somente sua projeção x' sobre o hiperplano ℓ_i satisfaz $A'\,x' = b'$
  - O vetor objetivo c também pode ser projetado sobre ℓ_i e temos então um problema de programação linear em $d - 1$ dimensões
- O processo inverso pode ser aplicado para trazer a solução para o espaço d-dimensional

Resumo do algoritmo em *d* dimensões

- $H = \{h_1, h_2, \ldots h_n\}$ é um conjunto de semi-espaços e c é um vetor em d dimensões
- Seleciona-se d semi-espaços cuja interseção é realizável com relação a c e computa-se o vértice ótimo v
- Adiciona-se um semi-espaço restante h_i por vez
  - Se v não é realizável com relação a h_i
    - Intersecta-se todos os semi-espaços $h_1 \ldots h_{i-1}$ com h_i gerando um problema $d - 1$ –dimensional
    - Resolve-se o problema e o vértice ótimo obtido é levado novamente para o espaço d-dimensional

Complexidade do algoritmo em *d* dimensões

- Seja $T(d,n)$ o tempo médio de execução do algoritmo para n semi-espaços em d dimensões
- Prova-se que $T(d,n) \in O(d\,!n)$
  - Ao inserir o i'ésimo semi-espaço, testa-se se v é realizável em $O(d)$. Tem-se as possibilidades:
    - v não é realizável: probabilidade $d\,/\,i$
      - resolve-se o problema d-1-dimensional em tempo $T(d-1, i-1)$
    - v é realizável: probabilidade $(i-d)\,/\,i$

Complexidade do algoritmo em *d* dimensões

- Temos que resolver a recorrência

$$\begin{cases} T(1,n) = n \\ T(d,n) = \displaystyle\sum_{i=1}^{n}\left(\frac{i-d}{i}d + \frac{d}{i}T(d-1,i-1)\right) \end{cases}$$

- Prova por indução que $T(d,n) \in O(d\,!n)$:

$$T(d,n) = \sum_{i=1}^{n}\left(\frac{i-d}{i}d + \frac{d}{i}T(d-1,i-1)\right)$$

$$\le dn + \sum_{i=1}^{n}\frac{d}{i}(d-1)!(i-1)$$

$$\le dn + d\,!\sum_{i=1}^{n}\frac{i-1}{i} \le dn + d\,!n$$

  - Não é exatamente o que queríamos, mas é suficiente. Uma prova mais criteriosa pode ser encontrada no livro