Introdução à Otimização

Lucas Hermann Negri
Claudio Cesar de Sá

Departamento de Ciência da Computação (DCC)
Centro de Ciências Tecnológicas (CCT)
Universidade do Estado de Santa Catarina (UDESC)

24 de Outubro de 2012

Tópicos I

1. Otimização
   - Introdução
   - Classes de Problemas
   - Classes de Algoritmos de Otimização
   - Espaço de Busca
   - Função Objetivo e Heurísticas
   - Ótimos Locais e Globais
2. Problemas Combinatoriais
   - Métodos Gulosos
   - Programação Dinâmica
3. Programação Linear
   - Conceitos
   - Redes de Fluxo
4. Análise Numérica
   - Descida de Gradiente
   - Gauss-Newton
   - Levenberg-Marquardt

Tópicos II

Introdução


menor...
máximo...
maior...
na menor área possível...
mínimo...
cumprir os requisitos...
menor distância...
mais barato...

Introdução

Ótimo

O melhor ou mais vantajoso; supera todos os outros.

Introdução

Problema de Otimização

Encontrar a melhor solução no conjunto de soluções possíveis.

Introdução

Problema de Otimização

Determinar os valores de um domínio definido que resultem nos valores ótimos da função objetivo pré-definida.

Classes de Problemas

Problemas de otimização são geralmente divididos em dois tipos: *otimização combinatorial* e *otimização numérica* [1].

Combinatorial Problemas definidos em um espaço de estados finito (ou infinito mas enumerável)

Numérica Definidos em subespaços infinitos e não enumeráveis, como os números reais e complexos

Otimização Combinatorial

- Verificar quais são as combinações de valores que tenham um resultado ótimo por uma métrica definida [2]

- **Exemplo 1**: escolher o trajeto que minimize a distância percorrida em uma viagem de carro entre duas cidades

- **Exemplo 2**: determinar qual será o trabalho de cada membro da equipe de forma a maximizar a eficiência da equipe

Otimização Combinatorial - Exemplo 1

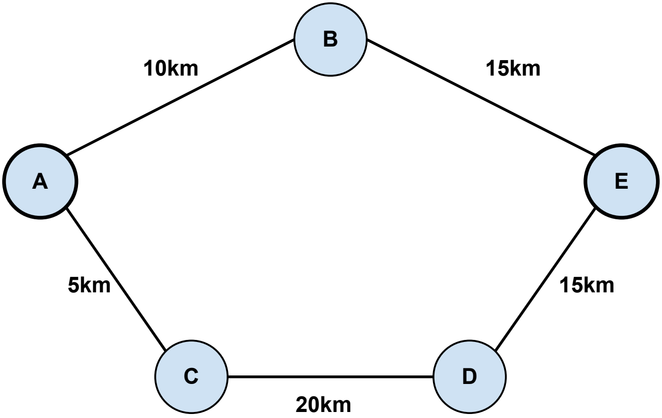


Figura : Exemplo 1: caminho mínimo entre duas cidades

Otimização Combinatorial - Exemplo 2

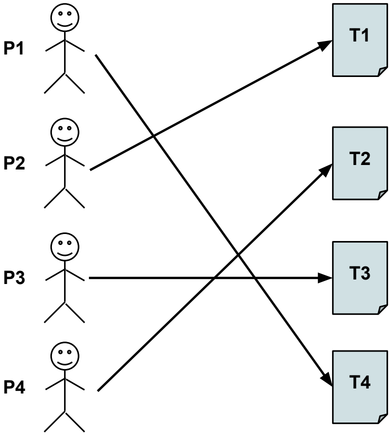


Figura : Exemplo 2: designação de trabalhos

Otimização Numérica

- Determinação dos parâmetros de uma função que resultem em seu valor ótimo, sendo que os valores possíveis para os parâmetros não podem ser enumerados

- Problemas usualmente modelados na forma de **minimização**

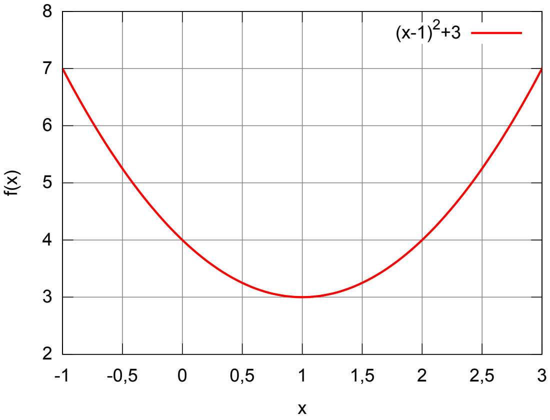
Otimização Numérica

- **Exemplo 1:** Determinar o valor mínimo de uma função

- **Exemplo 2:** Calcular qual a mistura mais barata de ingredientes em ração canina que satisfaça os requisitos nutricionais

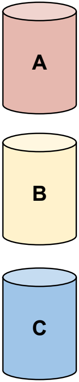
Otimização Numérica - Exemplo 1

Minimizar a função $f(x) = (x-1)^2 + 3$.



Otimização Numérica - Exemplo 2

Ingredientes:



Cada ingrediente tem uma certa concentração de nutrientes e um custo. Qual a combinação de ingredientes que minimiza o custo, satisfazendo os requitos nutricionais do cachorro?

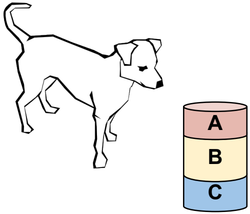


Figura : Exemplo 2 - Mistura ótima para cães.

Classes de Algoritmos de Otimização

- O universo de algoritmos de otimização existes é amplo e diversificado
- Muitos algoritmos empregam técnicas híbridas

Classes de Algoritmos de Otimização

- Área com contribuições de pesquisadores de diversas disciplinas
- Não faz sentido criar uma taxonomia precisa das diferentes classes
- Apesar disto, podemos ter uma visão geral

Algoritmos Determinísticos e Probabilísticos

Determinístico Em um algoritmo determinístico todas as decisões são feitas com base nas informações de entrada (problema). Para um mesmo problema, o algoritmo sempre resulta no mesmo resultado

Probabilístico Um algoritmo probabilístico utiliza ao menos uma vez um número (pseudo) aleatório para realizar uma decisão

Algoritmos Determinísticos

- Algoritmos determinísticos são utilizados quando pode-se efetuar decisões eficientemente a partir dos dados do problema
- Exemplos: busca em profundidade/largura, branch-and-bound, métodos gulosos, programação dinâmica

Algoritmos Probabilísticos

- Geralmente empregados em casos onde métodos determinísticos não são eficientes, devido ao tamanho do espaço de busca, por não haver uma relação clara entre função objetivo e as variáveis otimizadas, entre outros motivos.

- Exemplos: têmpera simulada, algoritmos genéticos, otimização por enxame de partículas

Espaço de Busca

- O espaço de busca consiste no domínio das variáveis do problema de otimização tratado
- Métodos de otimização iterativos realizam buscas dentro deste espaço

Espaço de Busca

- Alguns problemas, como o do caixeiro viajante, não possuem soluções exatas conhecidas a não ser em força bruta (ex.: enumerar soluções candidatas e verificar a melhor, ou montar uma árvore de busca procurando cortar sub-árvores inúteis)
- O número de soluções candidatas pode ser demasiadamente grande, impossibilitando buscas de força bruta

Exemplo de Espaço de Busca

Problema: designar uma tarefa para cada pessoa de forma a minimizar o custo total (Exemplo 2 de problema combinatorial).

Exemplo:

	T_1	T_2	T_3	T_4
P_1	15	10	8	12
P_2	10	12	10	15
P_3	10	12	8	20
P_4	5	15	20	10

Exemplo de Espaço de Busca

- Existem $4! = 4.3.2.1 = 24$ combinações possíveis, logo pode ser resolvido em tempo aceitável pela verificação de todas as combinações

- Mas, se houvessem 30 pessoas e 30 trabalhos, qual seria o número de combinações possíveis?

Exemplo de Espaço de Busca

$30! = 30.29.28.27... =$

Exemplo de Espaço de Busca

$30! = 30.29.28.27... = 265252859812191058636308480000000$ combinações possíveis!

Exemplo de Espaço de Busca

Assumindo que um computador consiga gerar e verificar 1 bilhão de combinações por segundo, seria necessário mais do que 1 trilhão de anos de computação para computar a resposta ótima!

Exemplo de Espaço de Busca

- Existe um algoritmo $O(n^3)$ para a solução ótima para o problema de designação: o algoritmo húngaro

- Tempo necessário para solução pelo algoritmo húngaro: menos de $1\mu s$

Espaço de Busca

- Conclusão: determinadas instâncias não podem ser resolvidas por força bruta
- Alguns problemas possuem algoritmos conhecidos com complexidade polinomial

Espaço de Busca

- Quando não se conhece algoritmo em tempo polinomial, pode-se tentar o uso de técnicas gerais como métodos meta-heurísticos e programação por restrições para tornar o problema factível

- Técnicas como programação por restrições garantem o resultado ótimo, mas podem não ser suficientemente eficientes para alguns problemas

Espaço de Busca

- Técnicas como algoritmos genéticos e otimização por enxame de partículas não garantem a determinação de uma solução ótima, mas comumente podem encontrar soluções aproximadas em tempo factível

- Logo, em alguns casos deve-se abrir mão da solução ótima para tornar o problema factível

Função Objetivo

- A Função Objetivo é uma relação direta das variáveis otimizadas e uma métrica sobre a solução avaliada
- No problema de minimização do custo das alocações de trabalhadores, a função objetivo poderia dizer o custo total em função da designação de cada trabalhador
- Neste caso, algoritmos de otimização realizariam a minimização do valor da Função Objetivo

Função Heurística

- Uma Função Heurística provê informações extras sobre o problema que podem guiar a otimização, com o objetivo de acelerar o processo
- Difere do conceito de Função Objetivo

Função Heurística

- Uma Função Heurística é admissível se ela nunca superestima o custo para chegar ao objetivo
- Uma Função Heurística é consistente se ela for monotônica e admissível
- Algoritmos como o A* requerem em certas condições que a heurística utilizada seja consistente para garantir o resultado ótimo

Ótimo Local

Ótimo Local

Um ótimo local é uma solução que tem o melhor valor da Função Objetivo dentre sua vizinhança.

Ótimo Global

Ótimo Global

Um ótimo global é uma solução que tem o melhor valor da Função Objetivo dentre todas as possíveis soluções.

Exemplo: Mínimo Global

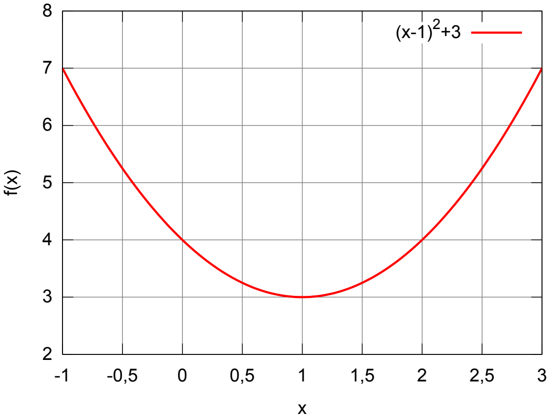


Figura : Exemplo de problema com exatamente um mínimo global.

Exemplo: Mínimos Globais

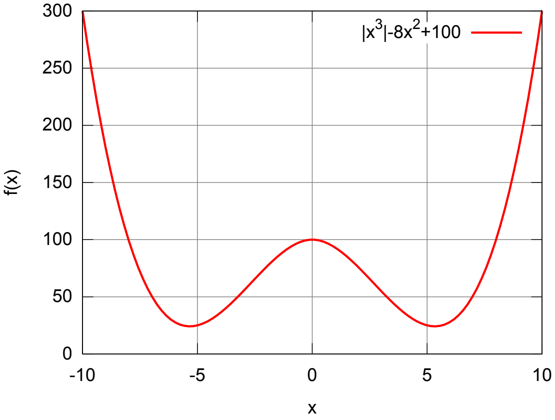


Figura : Exemplo de problema com dois mínimos globais.

Exemplo: Mínimo Local e Global

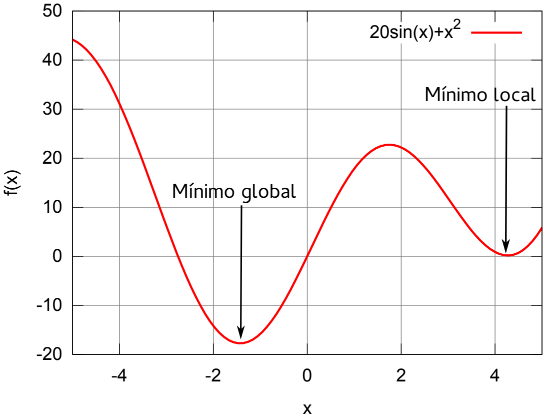


Figura : Exemplo de problema com mínimo local e global.

Ótimos Locais e Globais

- Determinados algoritmos de otimização podem ficar estagnados em ótimos locais

- Certos algoritmos utilizam estratégias para evitar a estagnação por meio de operadores que *sacudam* as soluções estagnadas ou que não parem a busca ao encontrar uma região localmente ótima

- Balanço entre o tamanho da região que deve ser explorada (tempo de execução) e a probabilidade de encontrar um ótimo global

Métodos Gulosos

- Algoritmos de otimização usualmente são compostos por iterações, onde são realizadas decisões
- Um algoritmo guloso sempre decide pel que for melhor para o momento (ótimo local), sem se preocupar diretamente com o futuro (ótimo global) [3]

Métodos Gulosos

- Para muitos problemas, escolhas gulosas não levam ao resultado ótimo, mas em alguns sim!
- Exemplos: seleção de atividades, árvore geradora mínima, caminho mínimo de Dijkstra

Métodos Gulosos

- Para muitos problemas, escolhas gulosas não levam ao resultado ótimo, mas em alguns sim!
- Exemplos: seleção de atividades, árvore geradora mínima, caminho mínimo de Dijkstra

Métodos Gulosos - Exemplo

Problema: Determinar a menor quantidade de notas para formar uma quantia de N reais utilizando notas de R\$ $1,00$[1], R\$ $10,00$ e R\$ $50,00$.

Exemplo: Para formar R\$ 113, a solução ótima é usar 2 notas de R\$ $50,00$, 1 nota de R\$ $10,00$ e 3 notas de R\$ $1,00$, totalizando 6 notas.

[1] ok, moeda de R\$ $1,00$!

Métodos Gulosos - Exemplo

Esta instância pode ser resolvida de forma gulosa, procurando sempre utilizar primeiro as notas de maior valor, então as de menor valor.

Métodos Gulosos - Exemplo

A estratégia gulosa funcionaria se as notas disponíveis fossem de R\$ $1,00$, R\$ $4,00$, R\$ $6,00$, e R\$ $9,00$?

Qual seria a menor quantidade de notas, neste caso, para compor R\$ $17,00$?

Métodos Gulosos - Exemplo

Solução gulosa: $1 \times 9 + 1 \times 6 + 2 \times 1$ = 4 notas

Solução ótima : $1 \times 9 + 2 \times 4$ = 3 notas

Métodos Gulosos - Conclusão

- Determinados problemas podem ser resolvidos (solução ótima) por métodos gulosos

- Alguns problemas não podem ser resolvidos de forma gulosa (apesar da falsa aparência). Logo, deve-se buscar a solução por outros algoritmos

- Métodos gulosos podem prover soluções aproximadas/iniciais para problemas de otimização

Programação Dinâmica

1. *Programação dinâmica* (PD) resolve problemas combinando a solução de subproblemas
2. Cada subproblema é computado somente uma vez pelo uso de memória adicional

Programação Dinâmica - Algoritmos Clássicos

- Cocke-Younger-Kasami (CYK)
- Floyd-Warshall
- Ordem de multiplicação de matrizes
- Maior subsequência comum
- Árvores binárias de busca ótimas

Programação Dinâmica - Conceitos

PD pode ser aplicada a problemas que exibem as propriedades de *sobreposição de subproblemas* e *subestrutura ótima*.

Programação Dinâmica - Conceitos

Sua definição é parecida com a do método da *divisão e conquista*.

Utiliza-se *divisão e conquista* quando os subproblemas são independentes, e PD quando estes não são independentes, isto é, compartilham subsubproblemas.

Sobreposição de Subproblemas

Diz-se que um problema tem subproblemas sobrepostos se este pode ser dividido em problemas menores que serão *reutilizados diversas vezes*, ao invés de gerar cada vez subproblemas distintos.

Subestrutura Ótima

Um problema apresenta a característica de subestrutura ótima quando a sua solução ótima pode ser computada a partir das soluções ótimas dos seus subproblemas.

Método de Solução

Identificadas as propriedades de **Sobreposição de Subproblemas** e **Subestrutura Ótima** pode-se escrever um algoritmo que compute a solução ótima de cada subproblema, utilizando as soluções já computadas para formar a solução dos outros subproblemas.

Programação Dinâmica - Exemplo

```
fibo(n)
    if n == 0 then return 0 end
    if n == 1 then return 1 end
    return fibo(n-1) + fibo(n-2)
end

seq = {0, 1, 1, 2, 3, 5, 8, 13, 21, ...}
```

Programação Dinâmica

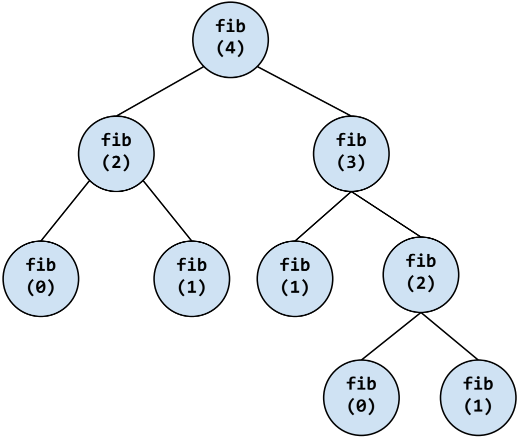


Figura : Árvore de chamadas

Programação Dinâmica - Exemplo

- A solução recursiva não é eficiente
- Complexidade exponencial: $\approx \Theta(1,6^n)$
- São feitas muitas chamadas (recursivamente) repetidas à função

Programação Dinâmica - Exemplo

Solução: memorizar as chamadas já computadas (PD top-down)!

Solução Recursiva + Memorização (PD top-down)

```
mapa[0] = 0
mapa[1] = 1

fibo(n)
    // evita repetição da computação
    if mapa[n] == vazio then
        mapa[n] = fibo(n-1) + fibo(n-2)
    end

    return mapa[n]
end
```

Programação Dinâmica

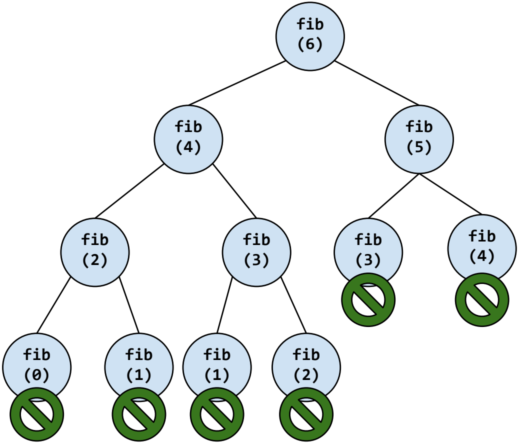


Figura : Árvore de chamadas sem repetição

Programação Dinâmica

Resumo:

- Programação dinâmica se aplica a problemas com a propriedade de sobreposição de subproblemas e subestrutura ótima

- Consiste em computar a solução ótima para cada subproblema e compor a solução de problemas maiores a partir das soluções dos menores já computados

- A chave está em identificar o **estado** que define cada subproblema

Programação Linear - Conceitos

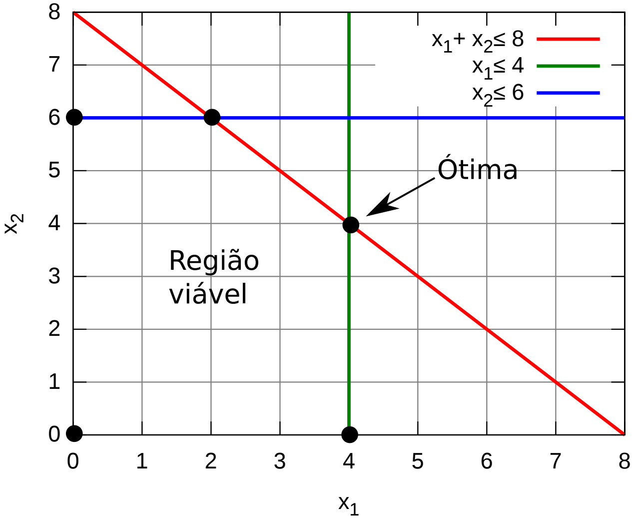
- Método para a determinação dos resultados ótimos em problemas de otimização que podem ser modelados por uma função objetivo linear e uma lista de restrições lineares (igualdades ou desigualdades)

- Primeiros algoritmos publicados na época da segunda guerra mundial, por George Dantzig e John von Neumann

Programação Linear - Exemplo

$$
\begin{aligned}
\underset{x_1, x_2}{\text{maximizar}} \quad & c = 2x_1 + x_2 \\
\text{sujeito a} \quad & x_1 + x_2 \leq 8 \\
& x_1 \leq 4 \\
& x_2 \leq 6 \\
& x_1, x_2 \geq 0
\end{aligned}
$$

Programação Linear - Exemplo



Programação Linear - Algoritmos

Simplex Algoritmo original desenvolvido por George Dantzig. Em casos especiais pode requerer um número exponencial de iterações em relação ao tamanho do problema

Elipsoide Um dos primeiros algoritmos com pior caso em tempo polinomial. Na prática, só era mais eficiente que o Simplex em casos especiais

Karmarkar Melhorou a complexidade do pior caso, sendo mais rápido que o Simplex em problemas típicos

Redes de Fluxo

- Uma rede de fluxo é um grafo onde as arestas representam a capacidade de fluxo entre dois vértices
- Pergunta possível: sabendo as capacidades de cada aresta, qual é o maior fluxo possível entre dois vértices?
- Analogia: água em um encanamento, corrente em um circuito

Redes de Fluxo

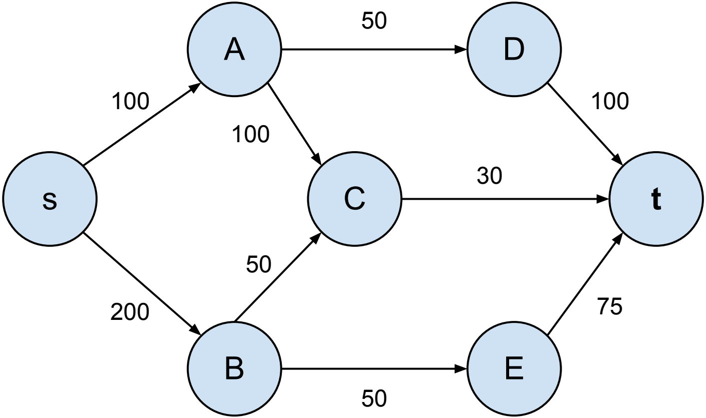


Figura : Exemplo de rede de fluxo.

Fluxo Máximo / Corte Mínimo

Algoritmos

- Ford-Fulkerson
- Edmonds-Karp
- Dinitz
- Push-Relabel

Aplicações

- Maior fluxo entre vértices de um grafo
- Maior casamento em grafos bipartidos
- Casamento em grafos bipartidos de maior menor/maior valor (método Húngaro)
- Maior número de caminhos disjuntos em grafos
- Escalonamento
- Segmentação de imagens

Análise Numérica - Otimização

- Métodos voltados à otimização numérica
- Minimizar / maximizar uma função / determinação de raízes
- Usualmente dependem do cálculo do gradiente da função, mesmo que este seja aproximado
- Usualmente são métodos iterativos

Descida de Gradiente

- Utiliza diretamente o gradiente (ou uma aproximação) da função como informação para a otimização [4]
- Uso em problemas de minimização, maximização, determinação de raízes e outros derivados destes

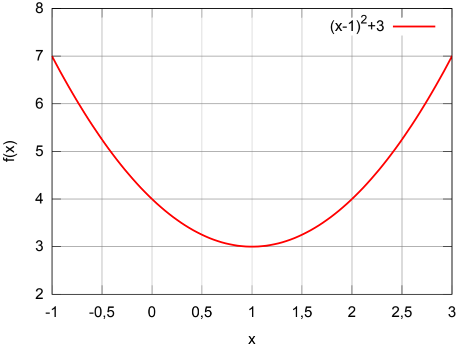
Descida de Gradiente

Para minimização:

$$x_{n+1} = x_n - \gamma \nabla f(x)$$

Descida de Gradiente - Exemplo

Objetivo: Determinar o valor de x que minimize a função $f(x) = (x-1)^2 + 3$.



Descida de Gradiente - Exemplo

Algoritmo:

1. $x_{n+1} = x_n - \gamma \nabla f(x)$
2. Se $|x_n - x_{n-1}| > \varepsilon$ então volte ao passo 1

Neste exemplo utilizaremos $\gamma = 0.4$ e partiremos de $x = 10$.

Exemplo - Resultado

```
x_1 = 10,00; f(x_1) = 6,240000
x_2 = 2,80 ; f(x_2) = 3,129600
x_3 = 1,36 ; f(x_3) = 3,005184
x_4 = 1,07 ; f(x_4) = 3,000207
x_5 = 1,01 ; f(x_5) = 3,000008
x_6 = 1,00 ; f(x_6) = 3,000000
```

Descida de Gradiente - Limitações

- Convergência lenta (passos pequenos ao se aproximar de um ótimo)
- Pode percorrer o espaço em *zig-zag* em determinados problemas
- O uso de um γ adaptativo ajuda, mas em alguns casos pode-se utilizar métodos de segunda ordem para acelerar o processo

Gauss-Newton

- Usado em problemas de mínimos quadrados [4]
- Para acelerar a convergência utiliza-se a *Hessiana*, além do gradiente de erro
- Iteração: $X_{n+1} = X_n + H^{-1}G$, onde H é a matriz *Hessiana* e G é o gradiente de erro, calculados a partir da *Jacobiana*

Gauss-Newton

- Usualmente converge mais rapidamente do que o método de descida de gradiente [4]
- Problemas lineares necessitam somente de uma iteração
- Problemas mal condicionados ou com estimativas iniciais distantes de um ótimo podem não convergir nem para um ótimo local

Levenberg-Marquardt

- Ponderação entre os métodos de descida de gradiente e Gauss-Newton [5, 4]

- Á escolha da participação de cada método é realizada sistematicamente para aproveitar a convergência rápida do Gauss-Newton em situações favoráveis, recorrendo à descida de gradiente caso contrário

Levenberg-Marquardt

- Iteração: $X_{n+1} = X_n + (H + \mu I)^{-1} G$
- Se o erro diminuiu com os novos parâmetros X_{n+1}, então diminue-se o valor de μ, caso contrário aumenta-se o valor de μ e descarta-se a última iteração
- A redução do μ leva ao aumento da influência do Gauss-Newton, enquanto que seu aumento acarreta na redução da influência do Gauss-Newton

Levenberg-Marquardt - Exemplo

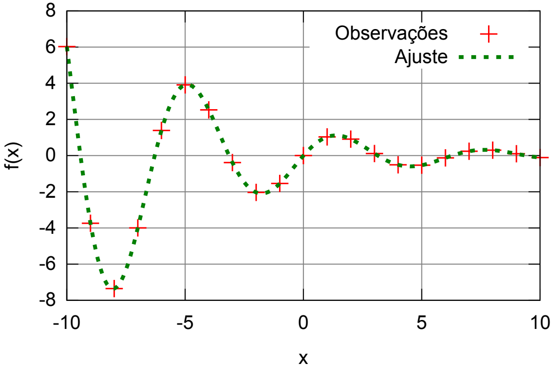


Figura : Ajuste dos parâmetros da função $f(x) = p_1 \sin(x) e^{p_2 x}$ aos pontos, utilizando o método de Levenberg-Marquardt (mínimos quadrados)

Programação por Restrições

- Uma técnica com origem nos anos 80
- Pouco uso no Brasil, tanto na indústria como academia
- Bem conhecida mundo afora
- Vantagens: rápida prototipação, pesquisas bem consolidadas, muitas ferramentas, buscas completas
- Desvantagens: a monotonicidade, simetria de respostas, explicações de respostas (mas melhor que as técnicas evolutivas)

Modelagem

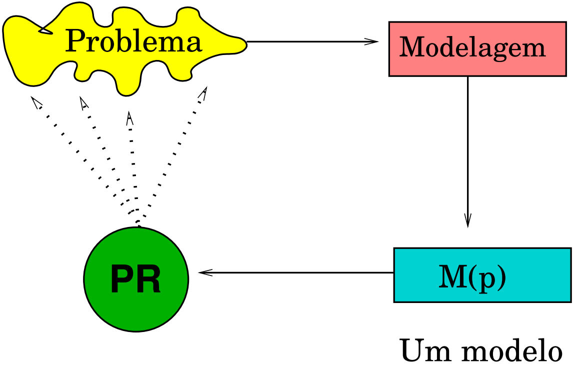


Figura : Metodologia da construção de Modelos da PR

Fluxo de Cálculo

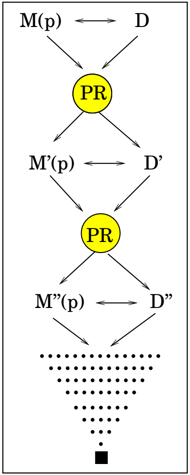
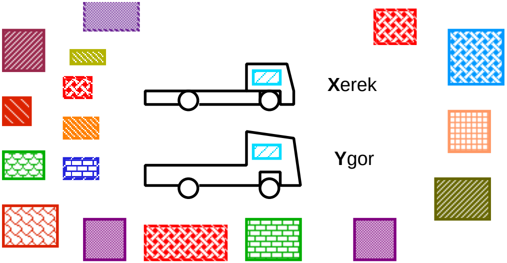


Figura : Dinâmica recursiva do cálculo da PR

Exemplo



Demanda de Transporte:

- Há caixas diversas (1..N), para **X**erek e **Y**gor, mas:
- $3 < C_{Xerek} < 9$
- $2 < C_{Ygor} < 5$
- Felizmente temos estes limites!

Um espaço de estado possível:

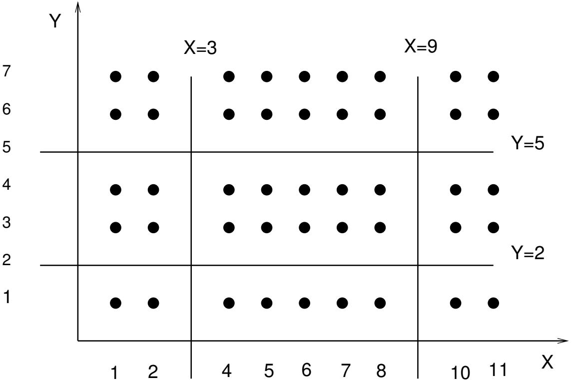


Figura : Encontrar valores segundo as demandas

continuando ...

A Figura 13 é a intersecção das seguintes condições:

- $3 < X < 9$
- $2 < Y < 5$

Quanto aos domínios: $D_X = \{1, 2, ..., 11\}$ e $D_Y = \{1, 2, ..., 7\}$

Codificando este modelo:

Postando estas restrições em código de um *solver*, tem-se:

```
:- lib(ic). %% Declarando uma biblioteca

quad_1 :-
   [X] :: 1..11,      %%% Dominios
   [Y] :: 1..7,
  X #> 3,          %%% Restricoes
  X #< 9,
  Y #> 2,
  Y #< 5,
  %%% Fase da propagacao e busca
   search([X,Y], 0, anti_first_fail,
                     indomain_middle,
                     complete, []),
   printf("\n Possiveis solucoes: X: %d  Y:%d", [X,Y])
   .
  %%% Um ponto ao termino da clausula
```

Cuja saída é:

```
?- quad_1
Possiveis solucoes: X: 6  Y:3
Possiveis solucoes: X: 6  Y:4
Possiveis solucoes: X: 7  Y:3
Possiveis solucoes: X: 7  Y:4
Possiveis solucoes: X: 5  Y:3
Possiveis solucoes: X: 5  Y:4
Possiveis solucoes: X: 8  Y:3
Possiveis solucoes: X: 8  Y:4
Possiveis solucoes: X: 4  Y:3
Possiveis solucoes: X: 4  Y:4
```

Significado em termos reais:

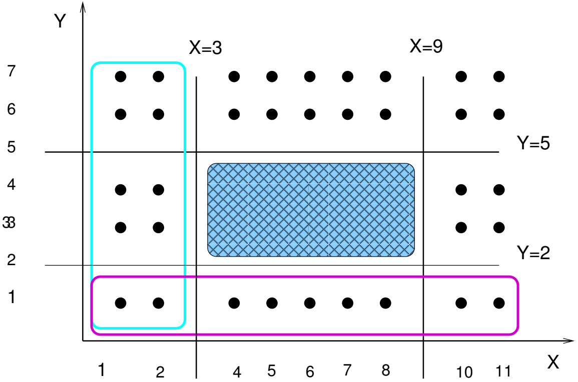


Figura : As áreas a serem explorada no *quadrado*

Espaço de Busca:

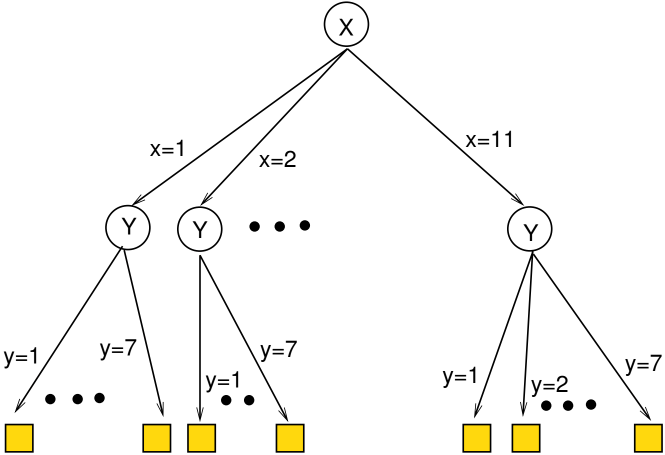


Figura : Espaço completo de busca

Filtragem e Propagação:

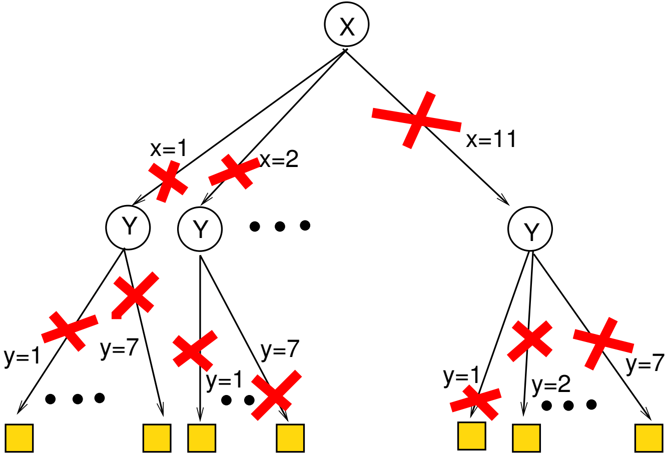


Figura : As restrições/filtragem

Efetivamente uma busca reduzida:

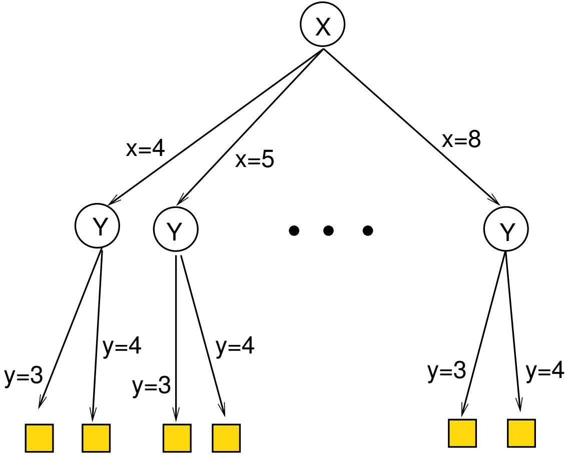


Figura : A busca (e expansão) efetiva sobre esta árvore

Resumindo a PR:

- Temos variáveis que precisam serem instanciadas
- Há valores que precisam serem atribuídos as estas variáveis
- Ai entra PR, com estratégias sobre as escolhas de variáveis do problema e visitas em seus domínios
- Tudo isto visa exclusivamente a redução do EE na árvore de busca
- A PR é *completa*, difere das buscas evolutivas
- *Gargalos* e pesquisas da PR:
  - Ausência de decisões e respostas aproximadas. Ex: *fuzzy logic*
  - *Descartes* inteligentes
  - Combinar com métodos de buscas globais Ex: *particle swarm* e PR

Resumindo o que a PR faz:

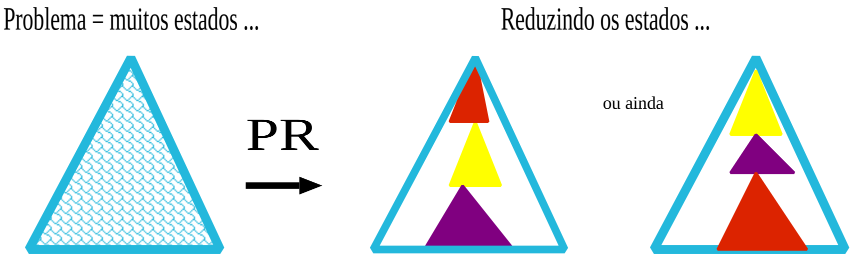


Figura : O *mar de estados* e a PR

Métodos Meta-Heurísticos

- São métodos que realizam a melhoria iterativa de um conjunto de soluções candidatas em relação a uma função objetivo
- Precisam de pouca informação sobre o problema alvo (não necessitam do gradiente, por exemplo)
- Não há garantia de encontrar uma solução ótima

Métodos Meta-Heurísticos

- Grande parte dos métodos meta-heurísticos utiliza (pseudo) aleatoriedade durante a melhoria das soluções
- Considerando simultaneamente todos os infinitos problemas de otimização, nenhum método de otimização é melhor que o outro
- Considerando somente os problemas práticos, existem algoritmos mais adequados que outros
- Problemas específicos podem ser resolvidos de maneira mais eficiente com a aplicação dos métodos adequados

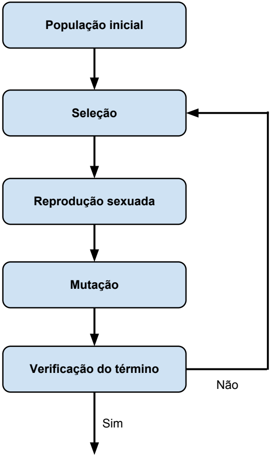
Exemplo de Métodos Meta-Heurísticos

- Algoritmos Genéticos
- Otimização por Enxame de Partículas
- Têmpera Simulada
- Busca Tabu
- Subida de Colina
- Otimização por Colônias de Formigas

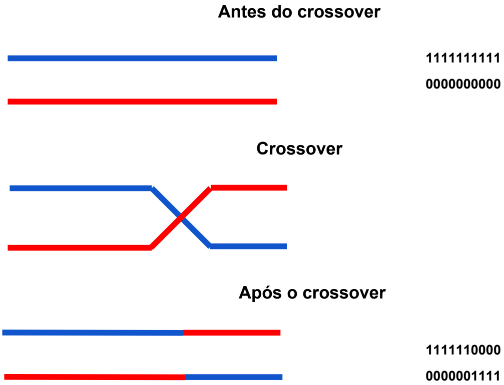
Algoritmos Genéticos

- Técnica de otimização e busca baseado nos princípios da genética e da seleção natural [6]
- Consiste na evolução de uma população de indivíduos (soluções candidatas) sujeitos a regras de seleção
- Objetivo: seleção, combinação e mutação de indivíduos com o objetivo de maximizar uma função objetivo

Algoritmo Geral

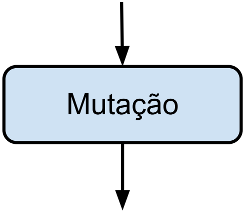


Reprodução Sexuada



Mutação

100100100100111001001001101



100101100100111001001001001

Algoritmos Genéticos - Vantagens

1. Uma implementação simples pode produzir resultados surpreendentes
2. Um resultado satisfatório pode ser alcançado rapidamente
3. Permite facilmente a computação paralela

Algoritmos Genéticos - Desvantagens

1. A exemplo das redes neurais, seus resultados são difíceis de serem explicados
2. Um ótimo global pode não ser alcançado
3. Um estado particular pode não ser alcançado
4. A *controlabilidade* (por onde encaminhar a busca) é baixa

Otimização por Enxame de Partículas

- Técnica meta-heurística de otimização, similar à técnicas como algoritmos genéticos [7]

- Consiste na evolução iterativa de um grupo de partículas (soluções), que estão em movimento em um ambiente *n*-dimensional, onde *n* é o número de parâmetros

- Utiliza uma função objetivo (fitness) que avalia a qualidade das soluções

Otimização por Enxame de Partículas

- Dispensa conhecimento exato do modelo matemático do problema otimizado
- Logo, permite a otimização de sistemas ”caixa-preta”, nos quais não se sabe o funcionamento interno, tendo somente a relação de entradas e saídas
- O tamanho do espaço de busca altera a convergência do algoritmo. Espaços de busca demasiadamente grandes resultam uma menor cobertura por partículas

Otimização por Enxame de Partículas

Algoritmo:

1. Inicialização da população de partículas
2. Reinicialização de partículas aleatórias (busca global, proposta)
3. Avaliação das partículas
4. Atualização da velocidade e posição das partículas

onde as etapas 2 a 4 são repetidas até que um critério de parada arbitrário seja satisfeito.

Otimização por Enxame de Partículas

$$v_{id} = w \times v_{id} + c_1 \times rand() \times (p_{id} - x_{id}) + c_2 \times rand() \times (p_{nd} - x_{id}) \quad (1)$$

onde

v_{id}	Velocidade da partícula i na dimensão d
w	Taxa de inércia, na faixa de [0,1)
$c1$ e $c2$	Taxas de aprendizagem, usualmente na faixa de [1,3]
$rand()$	Função que retorna um número aleatório na faixa de [0,1]
p_{id}	Melhor posição visitada pela partícula
p_{nd}	Melhor posição visitada pelas partículas da vizinhança

Otimização por Enxame de Partículas

$$v_{id} = w \times v_{id} + c_1 \times rand() \times (p_{id} - x_{id}) + c_2 \times rand() \times (p_{nd} - x_{id}) \quad (1)$$

onde

v_{id}	Velocidade da partícula *i* na dimensão *d*
w	Taxa de inércia, na faixa de [0,1)
$c1$ e $c2$	Taxas de aprendizagem, usualmente na faixa de [1,3]
$rand()$	Função que retorna um número aleatório na faixa de [0,1]
p_{id}	Melhor posição visitada pela partícula
p_{nd}	Melhor posição visitada pelas partículas da vizinhança

Otimização por Enxame de Partículas

$$v_{id} = w \times v_{id} + c_1 \times rand() \times (p_{id} - x_{id}) + c_2 \times rand() \times (p_{nd} - x_{id}) \quad (1)$$

onde

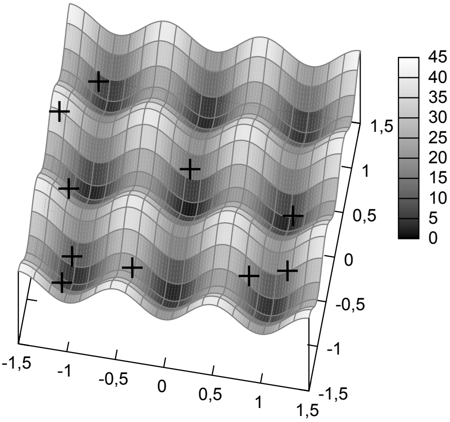
v_{id}	Velocidade da partícula i na dimensão d
w	Taxa de inércia, na faixa de [0,1)
$c1$ e $c2$	Taxas de aprendizagem, usualmente na faixa de [1,3]
$rand()$	Função que retorna um número aleatório na faixa de [0,1]
p_{id}	Melhor posição visitada pela partícula
p_{nd}	Melhor posição visitada pelas partículas da vizinhança

Exemplo

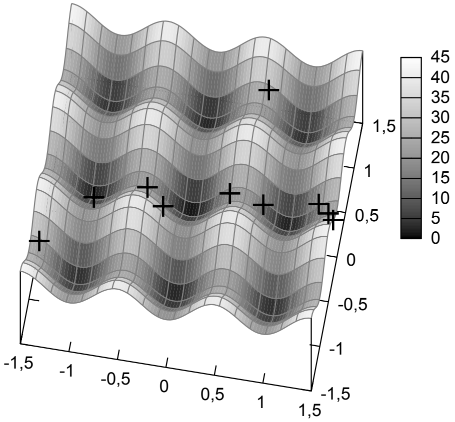
Exemplo: visualização do movimento (evolução) das partículas pelo espaço de busca na minimização da função de Rastrigin:

$$z = 20\left[x^2 - 10\cos\left(2\pi x\right)\right] + \left[y^2 - 10\cos\left(2\pi y\right)\right] \quad (2)$$

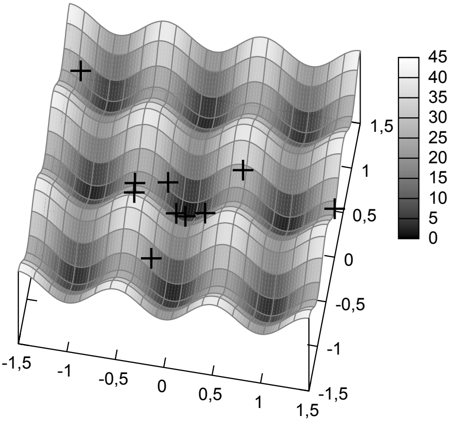
Exemplo - Estado Inicial



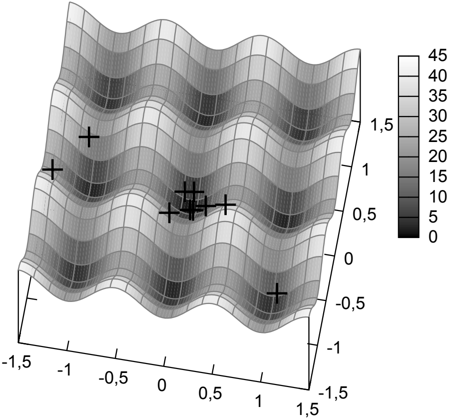
Exemplo - Iteração 5



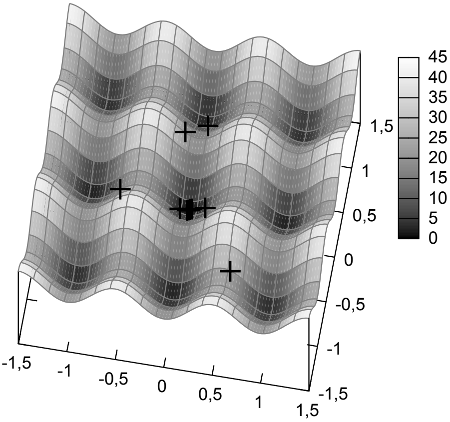
Exemplo - Iteração 10



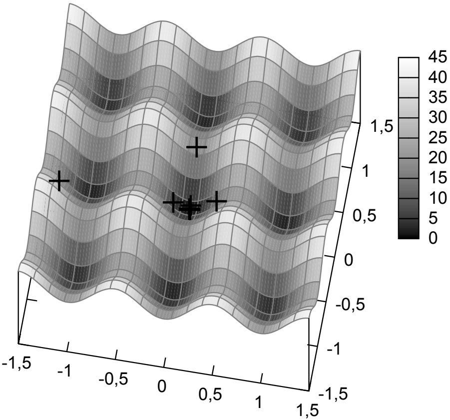
Exemplo - Iteração 15



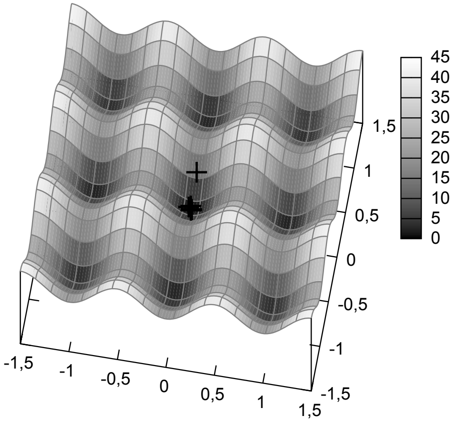
Exemplo - Iteração 20



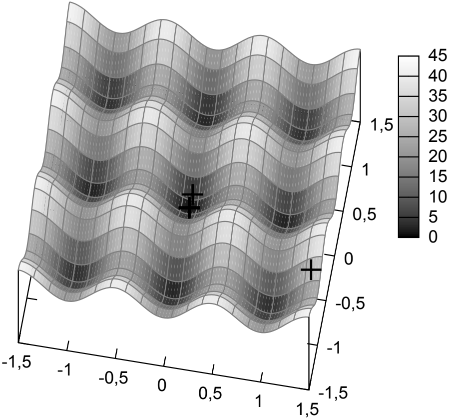
Exemplo - Iteração 30



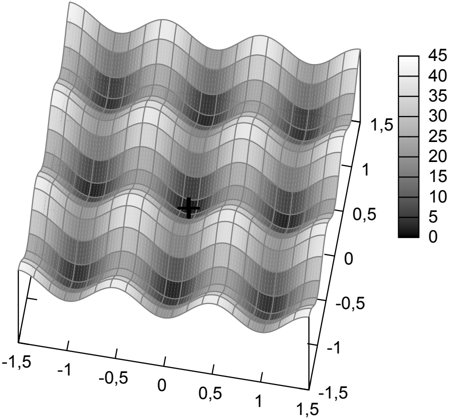
Exemplo - Iteração 40



Exemplo - Iteração 50



Exemplo - Iteração 100



Conclusões Finais

- Estamos cercados de problemas de otimização
- Tempo de computação limitado torna necessário o uso de métodos eficientes
- Grande variedade de métodos e abordagens
- Escolha do(s) método(s) depende do problema: não há algoritmo universal

Notas Finais

Agradecimentos

Muito grato!

Contato

lucashnegri@gmail.com
claudio@colmeia.udesc.br

Referências I

Thomas Weise.
Global optimization algorithms - theory and application, 2008.

Christos H. Papadimitriou and Kenneth Steiglitz.
Combinatorial optimization: algorithms and complexity.
Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1982.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein.
Introduction to Algorithms, Third Edition.
The MIT Press, 3rd edition, 2009.

Referências II

William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery.
Numerical recipes in C (2nd ed.): the art of scientific computing.
Cambridge University Press, New York, NY, USA, 1992.

K. Levenberg.
A method for the solution of certain non-linear problems in least squares.
Quarterly Journal of Applied Mathmatics, II(2):164–168, 1944.

Randy L. Haupt and Sue Ellen Haupt.
Practical genetic algorithms.
John Wiley & Sons, Inc., New York, NY, USA, 1998.

Referências III

Russell Eberhart.
A new optimizer using particle swarm theory.
In *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pages 19–43, 1995.