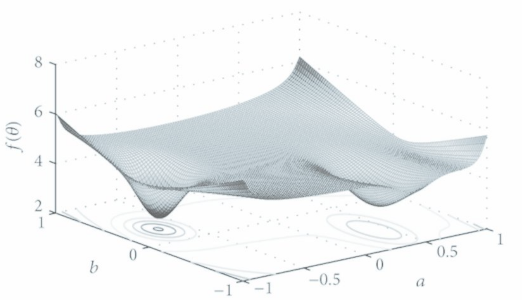


Aula 5

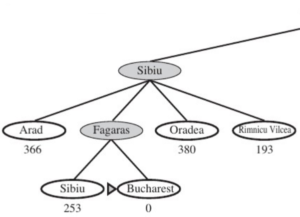
Busca Local e Problemas de Otimização

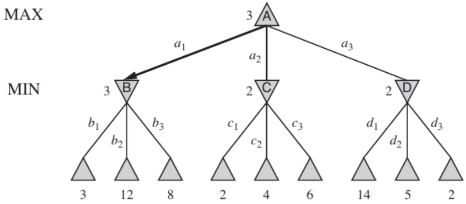
Rafael Geraldeli Rossi

Introdução

- Os algoritmos de busca que vimos até essa aula foram desenvolvidos para explorar o espaço de buscas **sistematicamente** → **mantendo um ou mais caminhos em memória e armazenando quais alternativas foram exploradas ao longo do caminho**
- Os algoritmos vistos até agora são denominados **algoritmos sistemáticos**
- No caso acima, quando um objetivo é encontrado, o caminho até o objetivo constitui a solução do problema

Introdução


Arad
Sibiu
Timisoara
329
Zerind
374
Arad
366
Fagaras
Oradea
380
Rimnicu Vilcea
193
Sibiu
253
Bucharest
0


MAX
MIN
3 A
a_1
a_2
a_3
3 B
2 C
2 D
b_1
b_2
b_3
c_1
c_2
c_3
d_1
d_2
d_3
3
12
8
2
4
6
14
5
2

Introdução

- Porém, em muitos problema, **o caminho até o objetivo é irrelevante**
  - **8 rainhas**: o que importa é a configuração final das rainhas, não a ordem em que elas foram adicionadas
  - **Outras aplicações**:
    - Design de chão de fábrica
    - Design de circuitos integrados
    - Escalonamento de jornadas de trabalho
    - Otimização de redes de telecomunicações
    - Distribuição de recursos
    - Otimização de funções em geral
    - ...

Introdução

- Se o caminho até o objetivo não importa, pode-se considerar uma classe diferente de algoritmos: **algoritmos de busca local**
- **Algoritmos de Busca Local operam considerando um único nó corrente** (ao invés de múltiplos caminhos) e **geralmente se movem apenas para o vizinho do nó**
- Tipicamente, **os caminhos seguidos pela busca local não são armazenados**

Introdução

- **Duas vantagem dos algoritmos de busca local**:

1. Eles **utilizam pouca memória** (inclusive em uma quantidade constante)

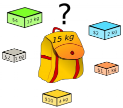
2. Eles geralmente **encontram soluções razoáveis em um espaço de estados grande ou infinito** (mesmo considerando estados contínuos), sendo que os algoritmos sistemáticos são inviáveis ou não apropriados para este tipo de problema

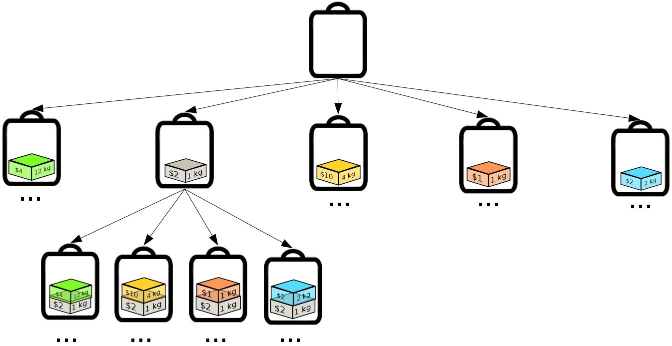
Introdução

- Além de encontrar objetivos, algoritmos de busca local são **úteis para resolver problemas de otimização pura**, os quais devem **encontrar um melhor estado de acordo com uma solução objetivo**
- Muitos problemas de otimização são parecidos com os problemas utilizados nas buscas apresentadas até essa aula: dada um estado, é possível se mover para um outro estado sendo que o número de estados a ser considerado é finito

Introdução

Problema da Mochila
(maximizar o valor carregado na mochila dado um limite de peso)

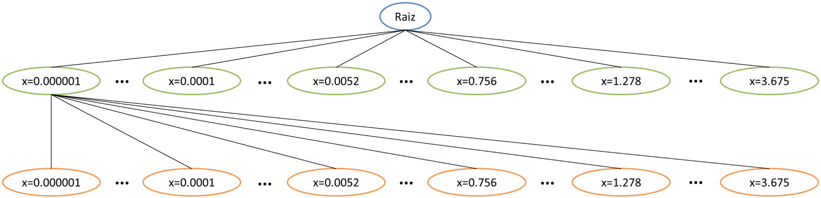




Introdução

- Porém, muitos dos problemas de otimização “padrão” **não são parecidos com os problemas apresentados nas buscas não informada, busca informada e busca com adversários $\rightarrow$ soluções com valores reais $\rightarrow$ espaço de busca infinito**

Encontre os valores de *x* e *y* para a equação: $x^2 + seno(x) - y^3 + \tan(y) = 5$

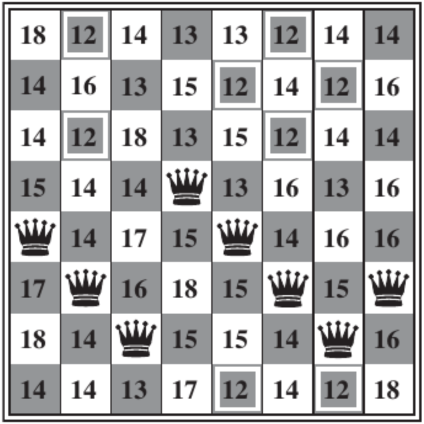


Introdução

- Para realizar e aproveitar os benefícios da busca local como definidos anteriormente, considera-se que **o agente está em um determinado ponto do espaço de estados e somente enxerga seus estados vizinhos**
- A ideia é **escolher para qual estado vizinho a busca deve seguir**
- Para isso, deve-se **medir o quão bom são os estados vizinhos**
- Geralmente escolhe-se **mover para o estado mais promissor** ou para um estado **melhor que o estado atual**

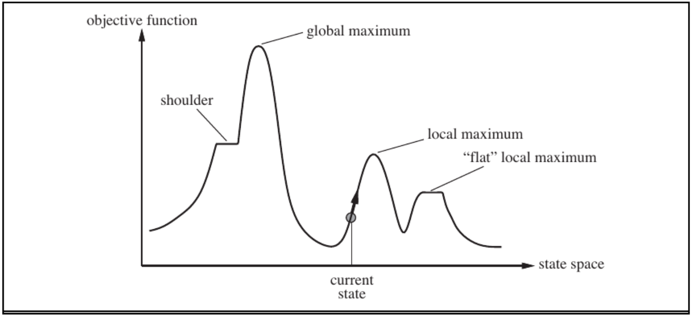
Introdução

Número de pares de rainhas que se atacam ao mover as rainhas nas colunas



Introdução

- O movimento do ponto no espaço de estados pode atingir os seguintes pontos:



- A variação dos valores da função objetivo de acordo com o espaço de estados é conhecida como **superfície de erro**

Introdução

- O objetivo é sempre encontrar o **máximo global** → **solução ótima**
- **OBSERVAÇÃO:** vale ressaltar que se o problema for **minimizar uma função objetivo**, o objetivo é encontrar o **mínimo global**
- Os algoritmos de busca locais visam explorar o espaço de busca em busca dos pontos de máximos (no caso de maximização de funções) ou mínimo no caso de minimização de funções
- Entretanto, alguns algoritmos só são capazes de encontrar os máximos / mínimos locais

Introdução

- A função objetivo é definida de acordo com o problema a ser tratado
  - Número de rainhas em ataque
  - Erro de uma função
  - Quantidade de espaço disponível em um circuito
  - ...

- **OBSERVAÇÃO:** a função objetivo inclusive pode ser uma função heurística

Introdução

- **Otimalidade e completude**
  - Um algoritmo de busca local **COMPLETO** sempre encontra um solução se existir
  - Um algoritmo de busca local **ÓTIMO** sempre encontra o mínimos/máximos globais

Hill-Climbing

- O algoritmo de busca **Hill-Clibing** (ou **Subida de Encosta**) pode ser entendido como um *loop* que se move continuamente na direção de um valor crescente no espaço de estados
- **O algoritmo para quando encontra um "pico"**, no qual nenhum vizinho apresenta um modelo maior
- **O algoritmo não mantém uma árvore de busca** $\rightarrow$ a estrutura de dados requerida necessita armazena o estado atual e seu valor de acordo com uma função objetivo

Hill-Climbing

- O algoritmo Hill-Climbing não "olha" para além dos vizinhos imediatos
- Alguns autores falam que é como "tentar encontrar o topo do Everest em meio a uma neblina e sofrendo de amnésia"

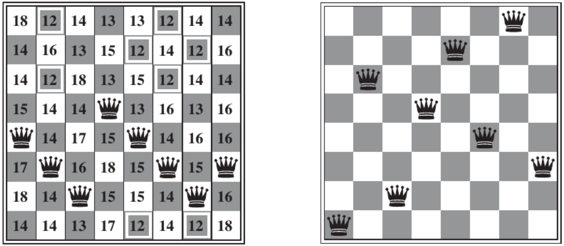
function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum

  current ← MAKE-NODE(*problem*.INITIAL-STATE)
  loop do
    neighbor ← a highest-valued successor of *current*
    if neighbor.VALUE ≤ current.VALUE **then return** *current*.STATE
    current ← *neighbor*

Hill-Climbing

- Para ilustrar o Hill-Climbing, vamos considerar o problema das 8 rainhas, considerando uma formulação completa dos estados $\rightarrow$ cada estado apresenta 8 rainhas no tabuleiro
- Os estados sucessores são todos os estados possíveis ao mover uma única rainha para um outro quadrado na mesma coluna $\rightarrow$ cada estado possui $8 * 7 = 56$ possíveis sucessores
- A função objetivo h é o número de pares de rainhas que atacam umas as outras
- O mínimo global é 0, o qual é alcançado quando nenhuma rainha ataca a outra

Hill-Climbing



(a) (b)

- (a) mostra um estado em que $h = 17$ e todos os valores dos estados sucessores (melhor estados têm $h = 12$)
- (b) mostra um mínimo local, no qual $h = 1$ mas não há sucessores com melhor custo

Hill-Climbing

- **OBSERVAÇÃO:** em caso de empates, o algoritmo Hill-Climbing típicamente escolhe aleatoriamente o sucessor (dentre os melhores)
- O algoritmo Hill-Climbing é também chamado de algoritmo **guloso** uma vez que que não considera as possibilidades futuras para realizar um movimento $\rightarrow$ escolhe somente a melhor opção imediata
- Apesar da busca gulosa ser considerada como um dos "sete pecados capitais", **a performance dos algoritmos gulosos são geralmente boas**

Hill-Climbing

- No problema das 8 rainhas, com apenas 5 movimentos saímos de um estado com $h = 17$ e vamos para um estado com $h = 1$
- Infelizmente o algoritmo Hill-Climbing pode ficar preso pelas seguintes razões
  - **Máximo local**: um pico cujo valor é maior do que qualquer estado vizinho mas menor que um máximo global (só inverter caso seja um mínimo local)
  - **Platô/Planalto**: é uma área plana no espaço de estados (vizinhos apresentam o mesmo valor)
  - **Ficar alternando entre dois pontos em torno de um vale ou cume**

Stochastic Hill-Climbing

- **Escolhe "aleatoriamente" o estado vizinho**
- Entretanto, **vizinhos com maior inclinação** (maiores valores da função objetivo) **têm maior probabilidade de serem escolhidos**
- Usualmente converte (acha a solução) **mais vagarosamente do que Hill-Climbing** original, mas **pode encontrar melhores soluções**

Random-Restart Hill-Climbing

- Implementa uma técnica muito famosa: "Se não funcionou, reinicie"
- **Esta versão realiza uma série de buscas Hill-Climbing gerando estados iniciais aleatoriamente, e retorna a melhor solução encontrada considerando todos os reinícios**
- **OBSERVAÇÃO:** para o problema das 8 rainhas, o *Random-Restart Hill Climbing* é bastante eficiente → mesmo para 3 milhões de de rainhas, esta abordagem pode encontrar a solução em menos de um minuto

Hill-Climbing

- **O sucesso do algoritmo Hill-Climbing depende muito do formato do espaço de busca**
- Se existirem pouco máximos locais e platôs, o *Random-Restart Hill-Climbing* irá encontrar uma boa solução rapidamente
- Entretanto, muitos problemas, a superfície de erro parece como uma “família de porcos espinhos espalhadas em um chão plano

Hill-Climbing

- Problemas NP completos tipicamente tem um número exponencial de máximos locais
- Apesar disso, máximo locais razoáveis podem ser obtidos após uma pequeno número de reinícios
- Máximos locais geralmente resolvem bem o problema

Simmulated Annealing

- O algoritmo Hill-Climbing nunca faz movimento "ladeira abaixo"
- Movimentos apenas "ladeira acima" **não garantem o algoritmo ser ótimo** $\rightarrow$ pode ficar preso em máximos locais
- Por outro lado, **movimentos puramente aleatórios podem ser extremamente ineficientes**
- Porém, parece ser razoável combinar o algoritmo **Hill-Climbing com uma caminhada aleatória para garantir otimalidade e completude**

Simmulated Annealing

- **A abordagem Simmulated Annealing (ou têmpera simulada) realiza a combinação de Hill-Climbing com caminhadas aleatórias**

- Na metalurgia, a têmpera é um processo utilizada para tornar vidros e metais mais duros por submeter os produtos inicialmente à altas temperaturas e diminuir a temperatura gradualmente [Wikipedia, 2016]

Simmulated Annealing

- Um exemplo muito utilizado para explicar o *Simmulated Annealing*, imagine que a tarefa seja mandar uma bolinha de ping-pong para a cavidade mais profunda em uma superfície irregular
  - Se deixarmos apenas a bola rolar, a bola ira imediatamente para o mínimo local, se e o mínimo local for a primeira superfície mais profunda próxima a bolinha de ping-pong
  - Se chacoalharmos a superfície, podemos mover a bolinha para outro ponto além do mínimo local
  - O truque é chacoalhar forte basta para afastar a bolinha dos mínimos locais mas não longe o suficiente do mínimo global
- Na abordagem *Simmulated Annealing*, a ideia é começar chacoalhando forte (alta temperatura) e gradualmente reduzir a intensidade da chacoalhada

Simmulated Annealing

Exemplo de funcionamento da abordagem *Simmulated Annealing*

`https://en.wikipedia.org/wiki/File:Hill_Climbing_with_Simulated_Annealing.gif`

Simmulated Annealing

Pseudocódigo da abordagem *Simmulated Annealing*

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  inputs: *problem*, a problem
    schedule, a mapping from time to "temperature"

  current ← MAKE-NODE(*problem*.INITIAL-STATE)
  for $t = 1$ **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow$ *next*.VALUE – *current*.VALUE
    if $\Delta E > 0$ **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{\Delta E/T}$

Simmulated Annealing

- Algoritmo
  - O *loop* mais externo do algoritmo é semelhante ao da abordagem *Hill-Climging*
  - Se o movimento melhora a situação atual, ele é sempre aceito; caso contrário o pode aceitar o movimento de acordo com uma probabilidade
  - A probabilidade diminui exponencialmente de acordo com o quão "ruim" é o movimento
  - A probabilidade também diminui conforme a "temperatura" T diminui
  - Com isso, "piores" movimentos são mais prováveis de acontecer nas primeiras do que nas últimas iterações
  - Se a função *schedule* diminuir vagarosamente o suficiente, o algoritmo iŕa encontrar o ótimo global

Simmulated Annealing

- O *Simmulated Annealing* pode ser visto como uma versão estocástica do algoritmo *Hill-Climbing* no qual alguns movimentos em direção à um "pior" estado são permitidos

- Movimentos piores possuem maiores probabilidade de acontecer no começo da têmpera e menos frequentes ao decorrer do tempo

- A função $schedule(t)$ determina o valor da têmpera T em função do tempo t

Local Beam Search

- Manter apenas um nó em memória parece ser uma solução extrema para o problema de limitação de memória
- O algoritmo **Local Beam Search** mantém k estados ao invés de apenas um
- k estados são aleatoriamente gerados
- Em cada passo, todos os sucessores do todos os k estados são gerados
- Se um dos estados atingirem o objetivo, o algoritmo é congelado e retorna a solução encontrada; caso contrário, os k estados sucessores mais promissores são gerados e os passos são repetidos

Local Beam Search

- A primeira vista, o algoritmo **Local Beam Search** pode ser visto como nada mais que k execuções paralelas do algoritmo *Hill-Climbing Random-Restart* rodando em paralelo
- Porém, os dois algoritmos são bem diferentes
  - No *Hill-Climbing Random-Restart*, cada processo roda independentemente dos outros
  - No *Local Beam Search*, as informações dos estados mais promissores são passadas entre as *threads*
    - Estados que geram os melhores sucessores "atraem" os outros estados
    - O algoritmo rapidamente abandona estados não promissores e foca apenas nos estados mais promissores atualmente

Local Beam Search

- Em sua versão mais simples o *Local Beam Search* pode sofre de falta de diversidade, uma vez que os k **estados podem rapidamente ficar concentrados em uma pequena região do espaço de estados** $\rightarrow$ torna-se apenas uma busca mais dispendiosa que o *Hill-Climbing*
- Uma variante chamada *Stochastic Beam Search* ajuda a aliviar o problema da diversidade
  - Ao invés de escolher os melhor k sucessores, a busca estocástica escolhe k sucessores "aleatoriamente" $\rightarrow$ maior probabilidade de seleção é dada ao melhores sucessores
  - Lembra o processo de seleção natural $\rightarrow$ sucessores mais adaptados ao meio possuem mais chance de sobreviverem e se reproduzirem

Algoritmos Genéticos

- Baseado na evolução natural das espécies (Charles Darwin, 1859)
- Cada indivíduo se adapta de uma forma ao ambiente (uns melhores, outros piores)
    - Capacidade de adaptação é hereditária
    - Melhores indivíduos têm mais chances de reprodução
    - Indivíduos mais adaptados geram mais descendentes

Algoritmos Genéticos

- Pode ser considerado como uma variante da abordagem *Stochastic Beam Search* no qual estados sucessores são gerados por combinar dois estados "pais" ao invés de modificar um único estados
- A analogia da seleção natural é a mesma do *Stochastic Beam Search*, exceto que os AGs lidam com reproduções sexuadas ao invés de reproduções assexuadas

Algoritmos Genéticos

- O problema a ser resolvido é representado por um conjunto de indivíduos
  - **Cada indivíduo é uma solução potencial do problema**
  - **Um cromossomo é utilizado para representar um indivíduo**
- É repetido um ciclo de "operações genética" para resolução de um problema até **obter um conjunto de indivíduos adequado ao problema**
  - Seleção
  - Reprodução
  - Mutação

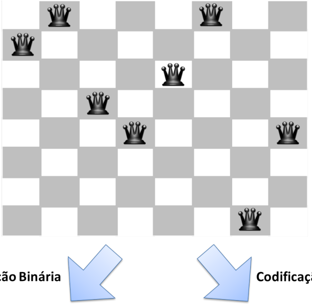
Indivíduo / População

- Indivíduos possuem uma "carga genética"
  - Cada indivíduo tem um cromossomo
  - Cada cromossomo é uma possível solução do problema
- A representação mais comum é a binária
  - Cromossomo é uma sequência de 0's e 1's
  - Indica a presença ou ausência de determinada característica
- Outras representações podem ser geradas de acordo com o problema

Indivíduo / População

- Exemplos de cromossomos de indivíduos para o problema das 8 rainhas
  1. **Binário**: cada cromossomo é representado por $8 \times \log_2 8 = 24$ *bits* → cada $log_2 8$ *bits* representa a posição de uma rainha na coluna em uma coluna do jogo
  2. **String numérica**: 8 posições em que cada posição representa um coluna e o valor da posição representa a posição da rainha na coluna

Indivíduo / População



Codificação Binária

0	0	1	0
0	0	0	1
0	1	0	0
0	1	0	1
0	0	1	1
0	0	0	1
1	0	0	0
0	1	0	1

Codificação Alternativa

2	1	3	4	3	1	8	5

Indivíduo / População

- Assim como na abordagem *Beam Search*, nos AGs são gerados k indivíduos aleatoriamente
- Todos os indivíduos gerados em um determinado instante do processo de busca do algoritmo genético constitui uma **POPULAÇÃO**
- Tamanho da população
  - Grande o suficiente para explorar bem o espaço de busca
  - Muito grande $\rightarrow$ processo lento

Função de Fitness

- Como avaliar o quão bom um indivíduo (adaptado) à um problema?
- **FUNÇÃO DE FITNESS!!**
- A função de *fitness* corresponde à uma função objetivo
  - Função de erro (problema de otimização de funções)
  - Função heurística
  - ...
- A função de *fitness* é definida de acordo com o problema

A população inicial é gerada aleatoriamente

- Função de fitness tem grande importância dentro dos algoritmos genéticos
- **Indivíduos mais aptos (maior valor retornado pela função de fitness) têm maior probabilidade de serem selecionados para reprodução**
- Se os pais são boas soluções, filhos tendem a ser boas soluções também
- Como selecionar os indivíduos mais aptos?
  - Várias técnias
  - Mais utilizada: **técnica da roleta**

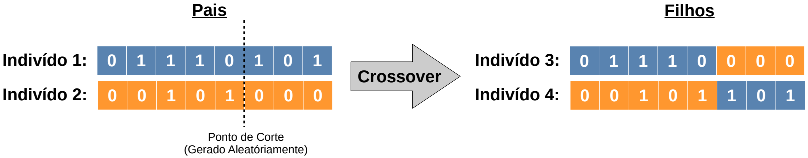
Técnica da Roleta

- Cada indivíduo é dono de uma porção da roleta
- Com isso, indivíduos melhores adaptados (maior função de fitness) possuem maior chance de serem escolhidos para reprodução

Indivíduo	Nro. Indivíduo	Aptidão	Participação na roleta
	1	4	5%
	2	20	25%
	3	16	20%
	4	24	30%
	5	8	10%
	6	8	10%

Crossover

- Dados dois indivíduos selecionados para reprodução, como gerar seus filhos?
- **Crossover (recombinação) → filho recebe parte das características de cada pai**
- **É necessário definir um ponto em que os cromossomos serão quebrados para recombinação (normalmente é aleatório)**



Crossover

- A operação de crossover **pode** gerar um estado distante de seus pais $\rightarrow$ diversidade na população

- Geralmente na execução de um algoritmo genético
    - A população tende a ser diversa no início do processos $\rightarrow$ similar ao Simmulated Annealing que dá grandes passos no espaço de estados nas iterações iniciais
    - Na final do processo a população tende a ser mais homogênea $\rightarrow$ novamente similar ao Simmulated Annealing gerando menores passos no espaço de estados ao longo do tempo

Mutação

- Taxa de Mutação
  - Evita contornar mínimos locais/globais
  - Muito alto $\rightarrow$ busca aleatória!
  - Cada gene de um cromossomo tem um probabilidade sofrer mutação
  - Essa probabilidade de mutação é definida pelo usuário

Critério de Parada

- **Tempo de execução**
- **Número de gerações**
- **Falta de diversidade** → variação entre o fitness das geração é semelhante
- Encontrou o máximo/mínimo global

Pseudocódigo

```
function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
  inputs: population, a set of individuals
          FITNESS-FN, a function that measures the fitness of an individual

  repeat
     new_population ← empty set
     for i = 1 to SIZE(population) do
        x ← RANDOM-SELECTION(population, FITNESS-FN)
        y ← RANDOM-SELECTION(population, FITNESS-FN)
        child ← REPRODUCE(x, y)
        if (small random probability) then child ← MUTATE(child)
        add child to new_population
     population ← new_population
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to FITNESS-FN
```

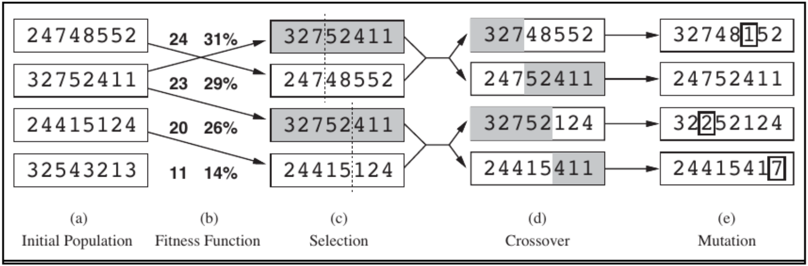
```
function REPRODUCE(x, y) returns an individual
  inputs: x, y, parent individuals

  n ← LENGTH(x); c ← random number from 1 to n
  return APPEND(SUBSTRING(x, 1, c), SUBSTRING(y, c + 1, n))
```

Características Gerais

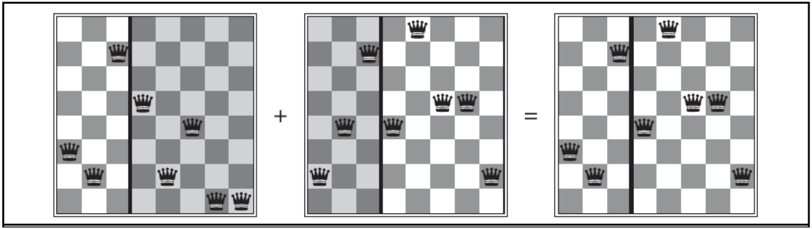
- Assim como a estratégia *Stochastic Beam Search*, os algoritmos genéticos combinam uma estratégia de subida de encosta com uma exploração aleatória
- Além disso, ocorre troca de "informações" entre as *threads* de buscas paralelas
- Uma das vantagens dos algoritmos genéticos em relação aos demais algoritmos de busca e a operação de *crossover* $\rightarrow$ aumentam o nível de granularidade dos indivíduos $\rightarrow$ maior diversidade no espaço de busca
- A mutação também é responsável por aumentar a diversidade no espaço de busca

Algoritmo Genético para o Problema das 8 raínhas



Algoritmo Genético para o Problema das 8 raínhas

Exemplo da operação de crossover para o problema das 8 raínhas



Busca Local em Espaços Contínuos

- **Muitos problemas do mundo real apresentam variáveis numéricas no problema de otimização**
- Os algoritmo de busca apresentados nesta aula foram instanciados como considerando espaços discretos
- Algumas considerações precisam ser feitas para considerar espaços contínuos

Busca Local em Espaços Contínuos

- **Um dos fatores mais importantes para se realizar uma busca em espaços numéricos é a definição do quão bom é um estado**
- Com esse tipo de informação / função podemos aplicar os algoritmos apresentados anteriormente
- Para isso, basta definirmos um valor de variação das variáveis (para cima ou para baixo, i.e., + ou -)
- No caso específico dos algoritmos genéticos, é necessário uma codificação do cromossomo para representar soluções numéricas

Busca Local em Espaços Contínuos

- Exemplo de codificação genética para valore numéricos
  - Considere o problema de minimizar a função $f(x, y) = |x * y * sen(y * \pi/4)$ com x e y pertencendo ao intervalo de 0 a 15
  - Cromossomos dos indivíduos
    - Indivíduo 1: 01000011 (x=0100) e (y=0011) [x=4 e y=3]
    - Indivíduo 2: 11101001 (x=1110) e (y=1001) [x=14 e y=5]

- **OBSERVAÇÃO:** para representar valores com casa decimais a ideia é a mesma → uma parte do cromossomo irá representar a parte inteira e outra parte irá representar as casas decimais

Busca Local em Espaços Contínuos

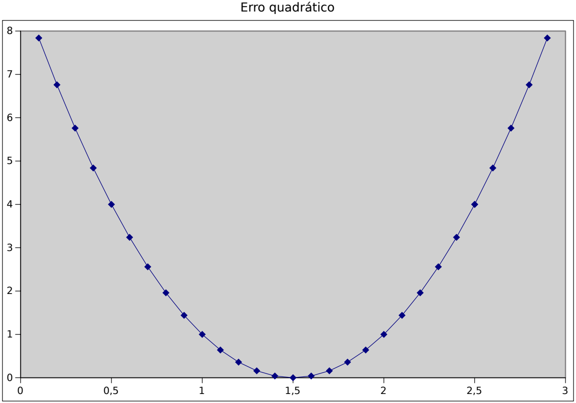
- Quando se existem valores de referência, uma função muito objetivo muito utilizada para guiar a busca em um espaço numérico é

$$\frac{1}{2}(Erro)^2$$

- O erro quadrático é a diferença entre o valor para uma função objetivo obtido pelo estado atual e o valor desejável, na qual a diferença é elevada ao quadrado
- Por exemplo, considere a equação $2x = 3$
  - Se consideramos $x = 0$, $2x = 0$
  - Porém, o valor que queremos obter com a função é 3
  - Portanto o erro quadrático é $(0 - 3)^2 = 9$
  - De forma geral: $Erro^2 = (valor\_obtido - valor\_esperado)^2$

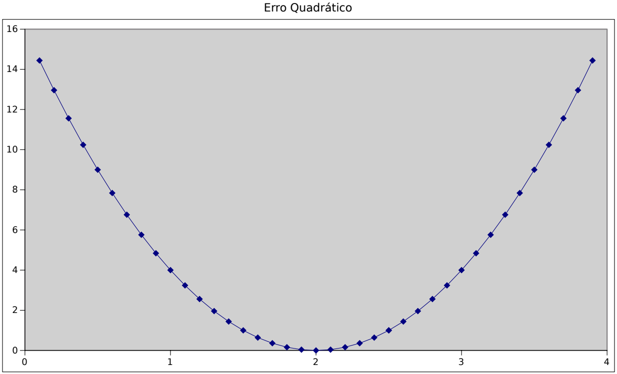
Busca Local em Espaços Contínuos

- Ex: encontrar o valor de x para equação $2x = 3$



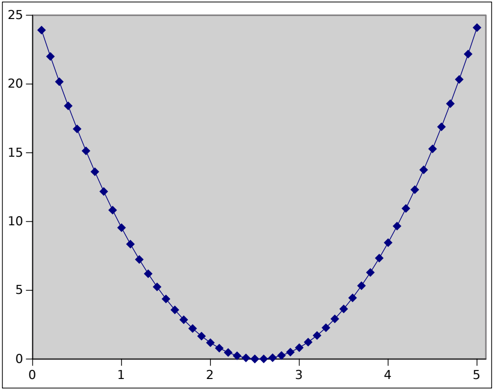
Busca Local em Espaços Contínuos

- Ex: encontrar o valor de x para equação $2x + x^2 = 8$



Busca Local em Espaços Contínuos

- Ex: encontrar o valor de x para equação $3x - seno(x) = 5.09$



Busca Local em Espaços Contínuos

- A derivada de uma função aponta para o ponto máximo de uma função
- A derivada do erro quadrático é

$$(valor\_obtido - valor\_esperado)$$

- No caso de utilizar uma abordagem do tipo *Hill Climbing* e o objetivo for:
  - Maximizar: caminhar em direção ao ponto máximo da função (considerar + erro)
  - Minimizar: caminhar em oposto ao ponto máximo da função (considerar − erro)

Busca Local em Espaços Contínuos

- No caso então de querer minimizar uma função $2x = 3$, a partir de um valor inicial de x, devemos atualizar x iterativamente até minimizar o erro quadrático

$$x = x - erro$$

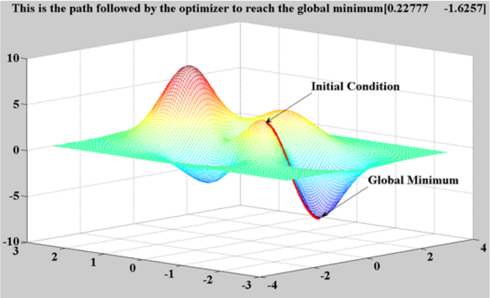
- Normalmente utiliza-se uma constante η para poderar a taxa com erro é atualizado

$$x = x - \eta erro$$

- Útil para que o algoritmo não dê grandes saltos no espaço de busca e para que o algoritmo não fique preso em torno de um mínimo global/local

Busca Local em Espaços Contínuos

- **OSERVAÇÃO:** métodos de busca local podem ser facilmente extendidos para resolver sistemas de equações lineares
- Espaço busca é um pouco mais complicado nesse tipo de situação



Material Complementar

- Descida de Gradiente e Têmpera Simulada (Simulated Annealing)

  https://www.youtube.com/watch?v=AQqDT2ioUok

- Método do gradiente

  https://pt.wikipedia.org/wiki/M%C3%A9todo_do_gradiente

- ALGORITMOS GENÉTICOS : CONCEITOS BÁSICOS E EXTENSÕES VINCULADAS AO PROBLEMA DE MINIMIZAÇÃO DE PERDAS ftp://ftp.dca.fee.unicamp.br/pub/docs/vonzuben/theses/pvargas_mest/arq_11.pdf

Material Complementar

- História, Conceito e Aplicações dos ALGORITMOS GENÉTICOS

  `https://www.youtube.com/watch?v=x7oHgs52BAI`

- Resolução do Problema da Mochila por Algoritmo Genético

  `https://www.youtube.com/watch?v=sXzFIrSt11o`

Imagem do Dia

Inteligência Artificial
http://lives.ufms.br/moodle/

Rafael Geraldeli Rossi
rafael.g.rossi@ufms.br

Slides baseados em [Russell and Norvig, 2010] e nos slides Prof. Ricardo Marcacini (algoritmos genéticos)

Referências Bibliográficas I

Russell, S. and Norvig, P. (2010).
Artificial Intelligence: A Modern Approach, Global Edition.
Pearson Education, Limited, 3rd edition.

Wikipedia (2016).