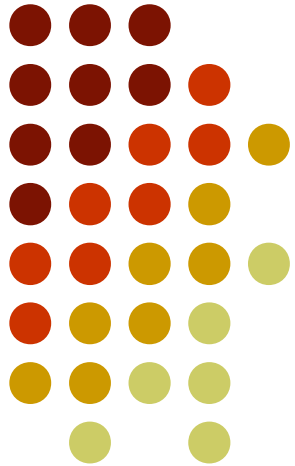
PGCA009 –Inteligência Computacional
Aula 9 Algoritmos de CE

Prof. Angelo Loula

Mestrado em Computação Aplicada (UEFS)



Computação Evolutiva

- Princípios da Computação Evolutiva
  - Reprodução com herança ‘genética’ de características
  - Variação ‘genética’
  - Seleção (para reprodução) favorável aos mais adaptados
  - Competição

Computação Evolutiva

- Um Algoritmo Evolutivo (AE)
  1. População inicial de indivíduos, representando por suas características possíveis soluções para um problema
  2. Indivíduos são selecionados segundo sua qualidade para solução do problema
  3. Reprodução dos selecionados com herança e variação de características gerando novos indivíduos
  4. Volta ao passo 2, se critério de parada não for atendido

Computação Evolutiva

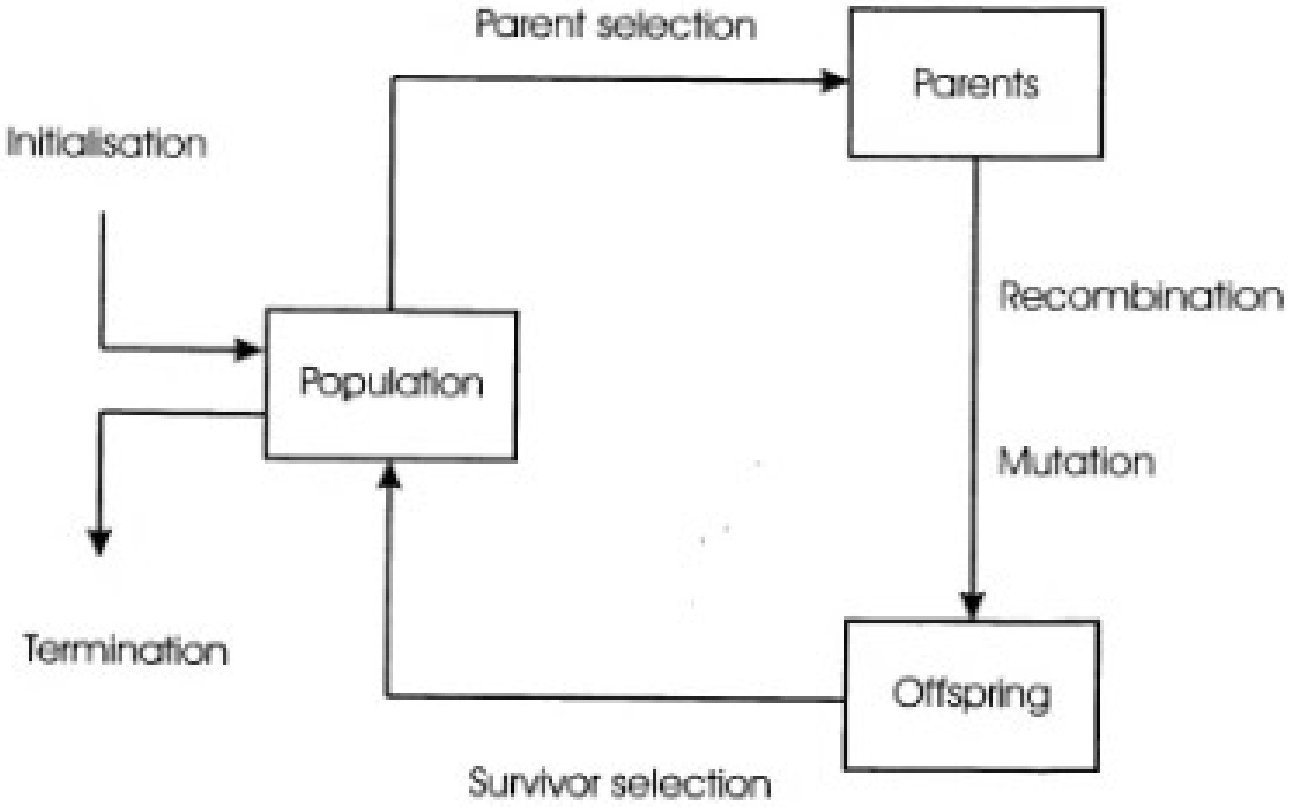


Fig. 2.2. The general scheme of an evolutionary algorithm as a flow-chart

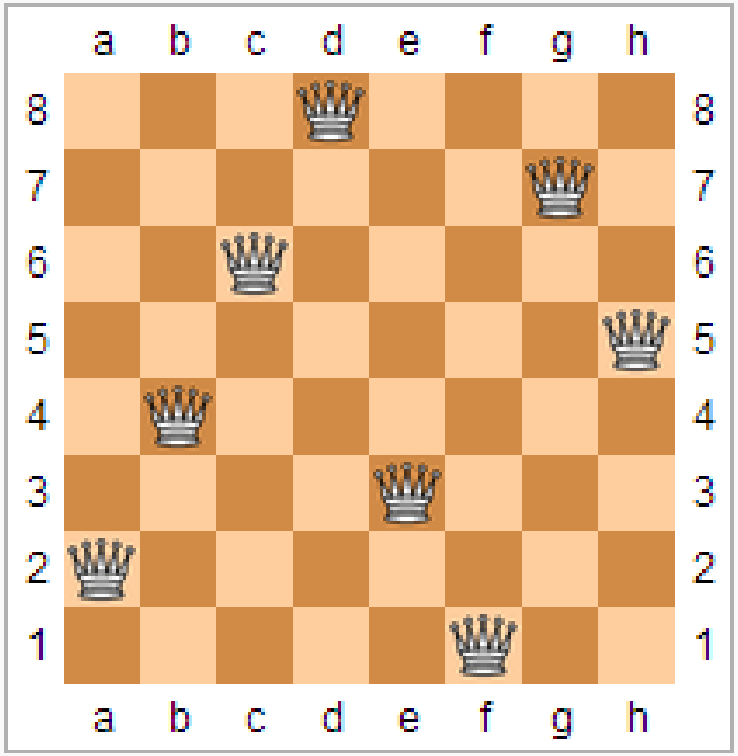
Fonte: Eiben & Smith (2003) Introduction to Evolutionary Computing

Computação Evolutiva

- Componentes de um Algoritmo Evolutivo
  - Representação (definição dos indivíduos)
  - População
  - Função de Avaliação
  - Procedimento de Seleção
    - Progenitores (reprodução)
    - Sobreviventes (composição da nova geração)
  - Operadores de variação (mutação e recombinação)

Computação Evolutiva

- Problema das oito-rainhas
  - Dispor oito rainhas no tabuleiro de xadrez sem possam se atacar



Uma das possíveis soluções

Computação Evolutiva

- Problema das oito-rainhas
  - Dispor oito rainhas no tabuleiro de xadrez sem possam se atacar
  - **Proponha**:
    - Representação
    - População
    - Função de Avaliação
    - Procedimento de Seleção: Progenitores e Sobreviventes
    - Operadores de variação

Computação Evolutiva

- Problema das oito-rainhas
  - Dispor oito rainhas no tabuleiro de xadrez sem possam se atacar
  - Como lidar com estas questões?
    - Configurações impossíveis
      - Duas rainhas na mesma casa
      - Mais ou menos de oito rainhas
    - Configurações que sabemos que não são factíveis
      - Rainhas na mesma linha ou coluna
      - Rainhas na mesma diagonal

Computação Evolutiva

- Problema das oito-rainhas
  - Possível AE:

Representation	Permutations
Recombination	"Cut-and-crossfill" crossover
Recombination probability	100%
Mutation	Swap
Mutation probability	80%
Parent selection	Best 2 out of random 5
Survival selection	Replace worst
Population size	100
Number of Offspring	2
Initialisation	Random
Termination condition	Solution or 10,000 fitness evaluation

Table 2.1. Description of the EA for the eight-queens problem

Fonte: Eiben & Smith (2003) Introduction to Evolutionary Computing

Computação Evolutiva

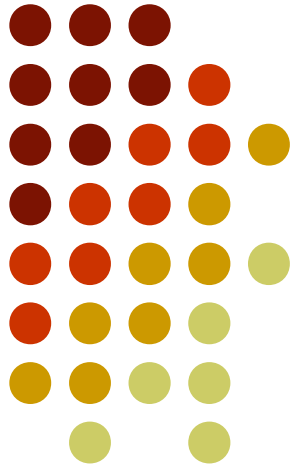
- Problema das oito-rainhas
  - Possível AE:

1. Select a random position, the crossover point, $i \in \{1, \ldots, 7\}$
2. Cut both parents in two segments after this position
3. Copy the first segment of parent 1 into child 1 and the first segment of parent 2 into child 2
4. Scan parent 2 from left to right and fill the second segment of child 1 with values from parent 2, skipping those that are already contained in it
5. Do the same for parent 1 and child 2

Fig. 2.3. "Cut-and-crossfill" crossover

Fonte: Eiben & Smith (2003) Introduction to Evolutionary Computing

Algoritmos Genéticos



Algoritmos Genéticos

- Proposta de John Holland
- Representação inicial por cadeias de bits
  - Mas surgiram outras representações posteriormente
- Há mutação, mas ênfase em recombinação
  - Recombinação considerada principal método de busca
  - Baixa mutação
- Seleção probabilística proporcional ao fitness

Algoritmos Genéticos

- Algoritmo Genético Simples ou Canônico
  - Representação: cadeia de bits
  - População: a definir
  - Função de Avaliação: a definir
  - Procedimento de Seleção: seleção para reprodução proporcional ao fitness, seleção para nova geração somente dos filhos
  - Operadores de Variação: Mutação por mudança de bit, Recombinação por crossover de 1 ponto, Probabilidade dos operadores a definir

Algoritmos Genéticos

- Algoritmo Genético Simples ou Canônico
  - Maximizar a função $f(x)=x^2$, $x \in [0,31]$
  - **Proponha um Algoritmo Genético Simples**

Algoritmos Genéticos

- Algoritmo Genético Simples ou Canônico
  - Maximizar a função $f(x)=x^2$, $x \in [0,31]$
  - **Proponha um Algoritmo Genético Simples**
    - Representação: string de 5 bits
    - Fitness f(x)
    - Operador de seleção dos pais: probabilístico = $f_i / \sum f_i$
    - Mutação: cada bit tem 1% chance de mudar
    - Crossover: seleção aleatória de pais e ponto de corte

Computação Evolutiva

- Algoritmo Genético Simples ou Canônico
  - Maximizar a função $f(x)=x^2$, $x \in [0,31]$

String no.	Initial population	x Value	Fitness $f(x) = x^2$	$Prob_i$	Expected count	Actual count
1	0 1 1 0 1	13	169	0.14	0.58	1
2	1 1 0 0 0	24	576	0.49	1.97	2
3	0 1 0 0 0	8	64	0.06	0.22	0
4	1 0 0 1 1	19	361	0.31	1.23	1
Sum			1170	1.00	4.00	4
Average			293	0.25	1.00	1
Max			576	0.49	1.97	2

Table 3.2. The x^2 example, 1: initialisation, evaluation, and parent selection

Fonte: Eiben & Smith (2003) Introduction to Evolutionary Computing

Computação Evolutiva

- Algoritmo Genético Simples ou Canônico
  - Maximizar a função $f(x)=x^2$, $x \in [0,31]$

String no.	Mating pool	Crossover point	Offspring after xover	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 | 1	4	0 1 1 0 0	12	144
2	1 1 0 0 | 0	4	1 1 0 0 1	25	625
2	1 1 | 0 0 0	2	1 1 0 1 1	27	729
4	1 0 | 0 1 1	2	1 0 0 0 0	16	256
Sum					1754
Average					439
Max					729

Table 3.3. The x^2 example, 2: crossover and offspring evaluation

Fonte: Eiben & Smith (2003) Introduction to Evolutionary Computing

Computação Evolutiva

- Algoritmo Genético Simples ou Canônico
  - Maximizar a função $f(x)=x^2$, $x \in [0,31]$

String no.	Offspring after xover	Offspring after mutation	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 0	1 1 1 0 0	26	676
2	1 1 0 0 1	1 1 0 0 1	25	625
2	1 1 0 1 1	1 1 0 1 1	27	729
4	1 0 0 0 0	1 0 1 0 0	18	324
Sum				2354
Average				588.5
Max				729

Table 3.4. The x^2 example, 3: mutation and offspring evaluation

Fonte: Eiben & Smith (2003) Introduction to Evolutionary Computing

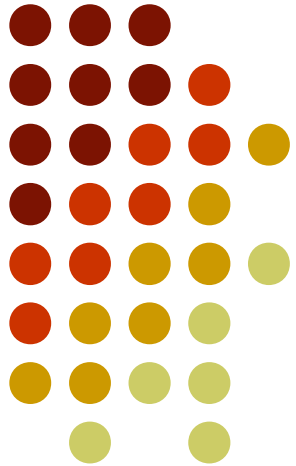
Algoritmos Genéticos

- Recombinação considerada principal método de busca
- Schema theorem
  - O que evolui são os genes na população e não os indivíduos
  - Blocos (partes) de bits são repassados entre indivíduos para formar indivíduos melhores
  - <1*010****> ou <*****11> podem ser excelentes esquemas para algum problema e serem repassados na população, aumentando frequência

Algoritmos Genéticos

- Schema theorem
  - Algoritmos genéticos exploram pelo crossover esquemas curtos, de baixa ordem, com alto fitness
  - Mas pode ocorrer decepção (deception): esquemas curtos quando se juntam podem formar indivíduos não tão bons
  - <1*010****> e <*******11> alto fitness, mas <1*010**11> com baixo fitness

Estratégias Evolutivas



Estratégias Evolutivas (EE)

- Proposta por Rechenberg e Schwefel
- Vetores de números em ponto flutuante
  - Problemas de otimização com variáveis contínuas
- Mutações com distribuição normal
  - Adiciona-se um valor aleatório
  - Parâmetros da mutação mudam durante execução
- Usa recombinação, com igual relevância para busca

Estratégias Evolutivas

- Parâmetros do algoritmo fazem parte da representação
  - Auto-adaptação
- Seleção determinística
- Tamanho da população de progenitores e de descendentes pode ser distinto

Estratégias Evolutivas

- Representação
  - Otimização de funções de $\Re^n$ em $\Re$
    - Representação pelo vetor $<x_1,...,x_n>$
  - Incluem parâmetros da mutação para desvio padrão da distribuição normal
    - $<x_1,...,x_n, \sigma>$ ou $<x_1,...,x_n, \sigma_1,...,\sigma_n>$
  - Pode incluir também parâmetros que indicam quantidade de iterações para modificar σ

Estratégias Evolutivas

- Mutação
  - $x_i' = x_i + N(0,\sigma_i')$
  - $N(0,\sigma_i')$: valor aleatório obtido de uma gaussiana com média zero e desvio padrão σ_i'
  - σ_i' é o desvio padrão mutado de σ_i
  - Há co-evolução de σ_i junto com x_i
  - Ajustes em σ_i podem ajudar em momentos diferentes da busca ou em regiões diferentes do espaço

Estratégias Evolutivas

- Mutação
  - Para um único σ, para todo x_i:
    - $x_i' = x_i + \sigma'.N_i(0,1)$
    - $\sigma' = \sigma.e^{\tau.N(0,1)}$

    τ é fixo e pré-definido, proporcional a $1/\sqrt{n}$

    $N_i(0,1)$ é sorteado a cada i e $N(0,1)$ somente uma vez
    - limitação do valor mínimo de σ

Estratégias Evolutivas

- Mutação
  - Para n parâmetros σ_i, para todo x_i:
    - $x_i' = x_i + \sigma_i'.N(0,1)$
    - $\sigma_i' = \sigma_i.e^{\tau'.N(0,1) + \tau.N_i(0,1)}$

    τ' é proporcional a $1/\sqrt{(2n)}$ e τ é proporcional a $1/\sqrt{(2\sqrt{n})}$
    - limitação do valor mínimo de σ

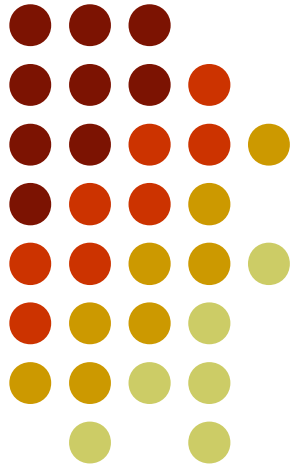
Estratégias Evolutivas

- Recombinação
  - Discreta ou intermediária
  - Discreta
    - Cada alelo de cada pai é aleatoriamente escolhido
  - Intermediária
    - É feita média do valor dos alelos dos pais
  - O número de pais pode ser >2
  - Tipicamente, discreta para x_i e intermediária para σ_i

Estratégias Evolutivas

- Seleção
  - Progenitores: aleatória com distribuição uniforme gerando λ
    - Não proporcional ao fitness
    - λ > μ, típico razão 1/7
  - Sobreviventes:
    - Dentre descentes λ (ou progenitores e descentes μ + λ), os μ indivíduos de melhor fitness sobrevivem de forma determinística
      - Seleção (μ, λ) ou (μ + λ)
      - Preferencialmente em EE, (μ, λ)

Programação Evolutiva



Programação Evolutiva

- Formalizada por FOGEL et al. (1966)
- Originalmente desenvolvida para evoluir máquinas de estado finito
- Somente mutação, não há recombinação: evolução de indivíduos e não população
- Problemas de otimização de valores reais: mutações com distribuição normal e estende o processo evolutivo ao espaço de parâmetros
- Operador de seleção: probabilístico
- Representação em aplicações atuais: vetores de números em ponto flutuante

Programação Evolutiva

- Evolução máquinas de estado finito (FSM)
  - Sistema dinâmico a eventos discretos
  - Estados e transições entre estados
  - Símbolos de entrada geram transições que fornecem símbolos de saída
  - Podem ser representadas por uma tabela de transições, indicando para cada estado e cada símbolo de entrada, qual o estado seguinte e o símbolo de saída

Programação Evolutiva

- Evolução máquinas de estado finito (FSM)

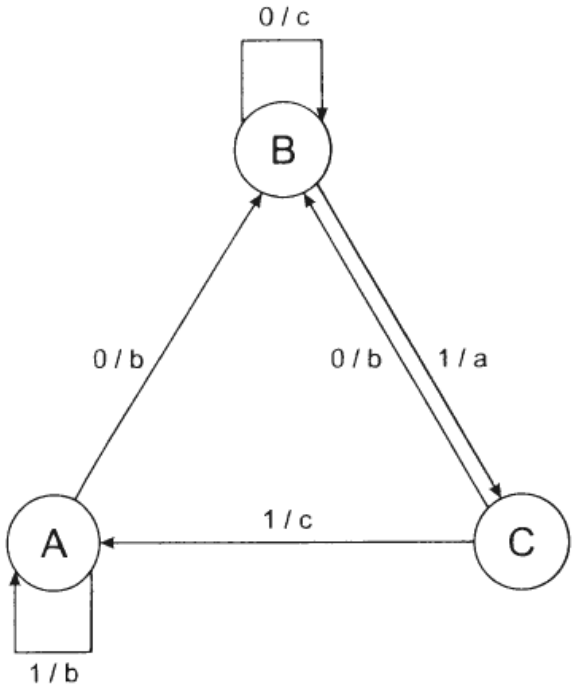


Fig. 5.1. Example of a finite state machine consisting of three states: A, B, and C. The input alphabet is $I = \{0, 1\}$, and the output alphabet is $O = \{a, b, c\}$. The FSM's transition function $\delta : S \times I \rightarrow S \times O$ that transforms the input stream to the output stream is specified by the arrows and their labels indicating the input/output of the given transition

Fonte: Eiben & Smith (2003) Introduction to Evolutionary Computing

Programação Evolutiva

- Evolução de vetores de números reais
  - Otimização de funções de $\Re^n$ em $\Re$
    - Representação pelo vetor $<x_1,...,x_n>$
  - Ênfase na representação mais natural e imediata possível
  - Inclusão na representação de parâmetros de auto-adaptação na representação
    - meta-EP

Programação Evolutiva

- Evolução de vetores de números reais
  - Mutação de $<x_1,...,x_n,\sigma_1,...,\sigma_n>$:
    - $x_i' = x_i + \sigma_i'.N(0,1)$
    - $\sigma_i' = \sigma_i.(1+\alpha.N(0,1))$,

    N(0,1) – saída de gaussiana de média 0 e desvio padrão 1

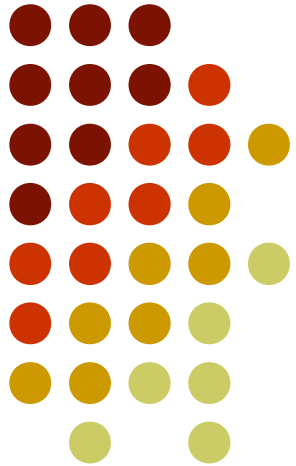
    $\alpha \sim= 0.2$

    limitação do valor mínimo de σ_i
  - Sem recombinação

Programação Evolutiva

- Evolução de vetores de números reais
  - Seleção de pais: cada indivíduo gera um filho
  - Seleção de sobreviventes em μ + λ
    - torneio de cada indivíduo *a* com *q* (típico, 10) outros escolhidos aleatoriamente, comparando fitness com demais
    - *a* ganha pontos a cada comparação ganha
    - indivíduos com mais pontos sobrevivem, mantendo tamanho da população

Programação Genética



Programação Genética

- Extensão dos algoritmos genéticos feita por Koza (1992)
- Espaço de busca são programas computacionais
- Representação: árvores
- Operadores de crossover e mutação adaptados para estruturas do tipo árvore
- Seleção probabilística e proporcional ao fitness.

Programação Genética

- Exemplo: Classificador de Crédito

Customer Id	No. of children	Salary	Marital status	Creditworthiness
Id-1	2	45.000	Married	0
Id-2	0	30.000	Single	1
Id-3	1	40.000	Married	1
Id-4	2	60.000	Divorced	1
...	...	...	...	...
Id-10000	2	50.000	Married	1

Table 6.2. Data for the credit scoring problem

IF *formula* THEN *good* ELSE *bad*

Fonte: Eiben & Smith (2003) Introduction to Evolutionary Computing

Programação Genética

- Exemplo: Classificador de Crédito

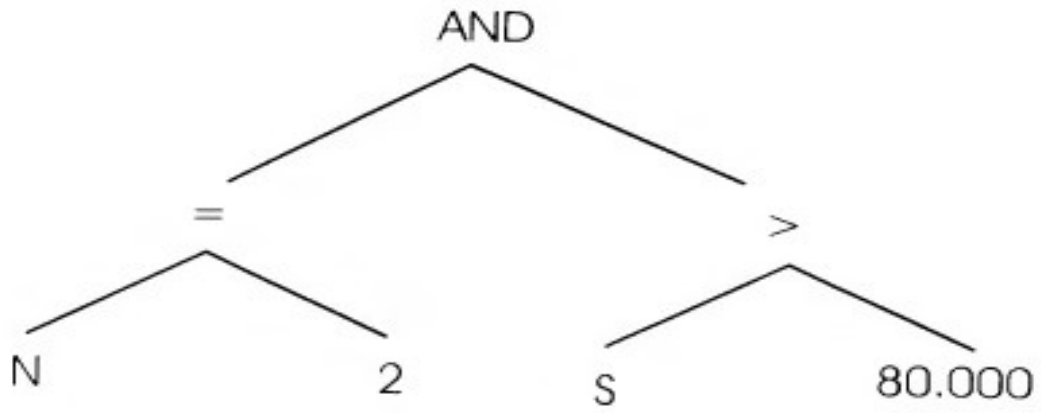


Fig. 6.1. Parse tree

IF $(No.\ children = 2)$ AND $(Salary > 80000)$ THEN $good$ ELSE bad

Fonte: Eiben & Smith (2003) Introduction to Evolutionary Computing

Programação Genética

- Representação
  - Árvores especificando
    - fórmulas aritméticas
    - expressões lógicas
    - programas de computador
  - É preciso definir sintaxe, símbolos terminais, símbolos não terminais, gramática

Programação Genética

- Mutação
  - Mais comum, trocar uma sub-árvore por outra
  - Nova sub-árvore pode ter tamanho diferente
- Recombinação
  - Crossover de sub-árvore
    - Troca de sub-árvores entre pais
    - Podem ter tamanhos diferentes

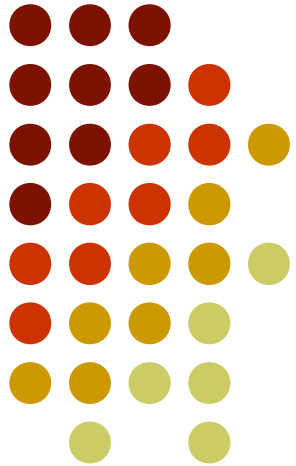
Programação Genética

- Seleção de Progenitores
  - Proporcional ao fitness
  - Usual, 80% vem dos x% melhores e 20% dos demais
- Seleção dos sobreviventes
  - Somente os descendentes sobrevivem
- Inicialização
  - Usual, altura máxima pré-definida

Programação Genética

- Controle da altura das árvores
  - Cromossomos de tamanho variável
  - Há tendência das árvores aumentarem de altura ao longo das gerações
  - Uma solução: tamanho máximo permitido, impedir operações que façam ultrapassar
  - Outra solução, penalidade no fitness por tamanho

Sistemas Classificadores



Sistemas Classificadores

- Learning Classifier Systems (LCS)
- Proposta de John Holland
- Descoberta de uma base de regras que realizam certa funcionalidade
- Maximiza recompensas apresentando saídas com base em entradas
- Evolução de uma população de regras e não regras individuais

Sistemas Classificadores

- Representação
  - Regras no formato, Se X (entrada) então Y (saída)
  - X e Y são cadeias com 0, 1
    - X também pode ter # (don't care)
  - Associado a cada regra, há uma recompensa R
    - X:Y → R
  - Recompensas vindas do ambiente são atribuídas as regras utilizadas
    - A recompensa associadas as regras representam uma recompensa *esperada*

Sistemas Classificadores

- Funcionamento
  - Ciclo de avaliação de regras
    - Cada regra é examinada para saber se corresponde a entrada recebida
    - Várias regras podem ter correspondência
    - Regras são agrupadas conforme saída proposta
    - O grupo com maior soma de recompensa define a saída
    - A recompensa (positiva ou negativa) é atribuída ao grupo vencedor

Sistemas Classificadores

- Funcionamento
  - Ciclo de descoberta de regras
    - Executa-se um algoritmo genético baseado na recompensa associada às regras
    - Recombinação e mutação de regras
    - Descendentes recebem a média das recompensas dos progenitores
    - Sobrevivem as regras com maior recompensa

Sistemas Classificadores

- Funcionamento
  - Alguns problemas atribuem recompensa para cada entrada-saída
  - Outros, somente após uma sequência de entradas e saídas
    - Recompensas atribuídas às regras por Bucket Brigade
  - Existem muitas variações e extensões
    - Holland, XCS, ZCS