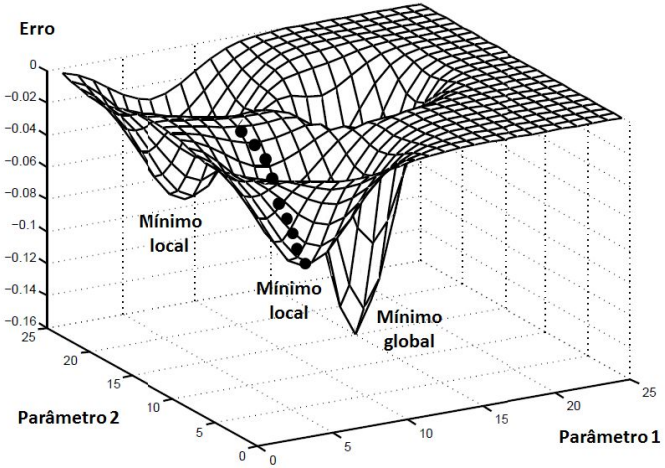
Inteligência Artificial

Busca por melhoria iterativa (parte 2)

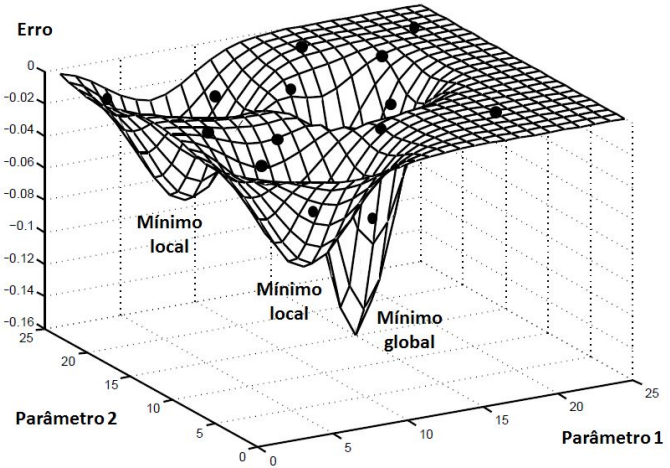
Profa. Debora Medeiros

Recapitulando

- Busca local
  - Uma solução candidata sofre alterações para melhorar sua qualidade
- Busca baseada em população
  - Uma coleção de soluções candidatas evoluem de maneira colaborativa



(a) Busca local



(b) Busca baseada em população

Recapitulando

- Exemplos:
  - Busca local: Hill climbing
  - Busca baseada em população: Algoritmos genéticos

Nesta aula

- Busca local
  - *Simulated annealing*
- Busca baseada em população
  - *Particle swarm optimization*

Simulated annealing

- Escolha do próximo estado aleatória
  - Se esse estado for melhor, é aceito
  - Senão, ele substitui o atual com probabilidade $exp(-\Delta E/T)$
    - onde ΔE é a diferença entre a função-objetivo desse estado e o estado atual
    - e T é um valor é reduzido a cada iteração.
- Comportamento inicial
  - Ser mais permissivo quanto a piora do estado
- Com o passar das iterações
  - Tende a aceitar apenas estados que apresentam melhoras

Simulated annealing

```
def SimulatedAnnealing(problema):
    s0 = solucaoInicial()
    T = 100
    while T > eps:
        s = estadoVizinhoAleatorio(s0)
        if f(s) > f(s0):
            s0 = s
        else:
            if random() <= exp(-(f(s)-f(s0))/T):
                s0 = s
        T = reduz(T)
```

Particle swarm optimization

- População de soluções candidatas
  - Chamadas neste contexto de partículas
- A cada iteração, cada partícula tem seus valores atualizados
  - De acordo com uma “velocidade”
    - Valor de deslocamento em cada uma de suas coordenadas
- Cada partícula tem a memória da melhor posição que já assumiu
  - Melhor local
- O algoritmo mantém a memória da melhor posição já encontrada até o momento
  - Melhor global

Particle swarm optimization

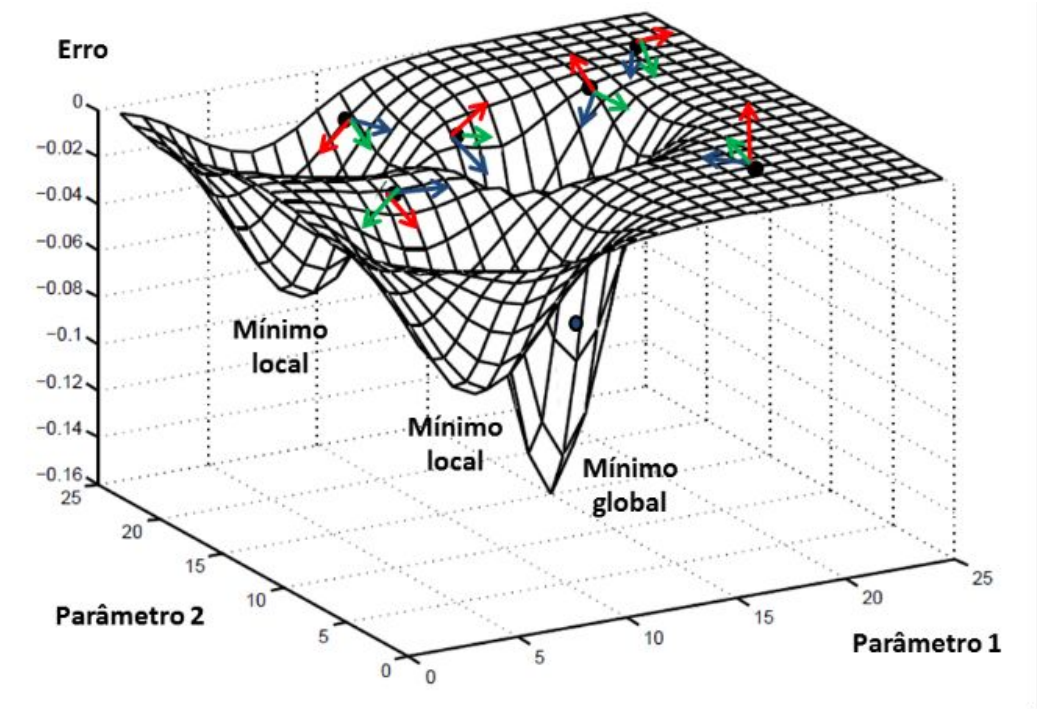
- A velocidade é atualizada (v_{i+1}) a cada iteração em função da:
  - velocidade atual (v_i)
  - coordenadas da partícula (x_i)
  - melhor local (p_i)
  - melhor global (g)

$$v_{i+1} = w * v_i + r_1 * c_1 * (g - x_i) + r_2 * c_2 * (p_i - x_i)$$

- Então, a posição de cada partícula é atualizada a cada iteração de acordo com sua velocidade

$$x_{i+1} = x_i + v_i$$

Particle swarm optimization



$$v_{i+1} = w * v_i + r_1 * c_1 * (g - x_i) + r_2 * c_2 * (p_i - x_i)$$

Particle swarm optimization

- Algoritmo

inicialize a nuvem de partículas
repita
  para $i = 1$ **até** m
    se $f(\mathbf{x}_i) < f(\mathbf{p}_i)$ **então**
      $\mathbf{p}_i = \mathbf{x}_i$
      se $f(\mathbf{x}_i) < f(\mathbf{g})$ **então**
        $\mathbf{g} = \mathbf{x}_i$
      fim se
    fim se
    para $j = 1$ **até** n
      $r_1 = \text{rand}()$, $r_2 = \text{rand}()$
      $v_{ij} = wv_{ij} + c_1 r_1 (p_i - x_{ij}) + c_2 r_2 (g_j - x_{ij})$
    fim para
    $\mathbf{x}_i = \mathbf{x}_i + \mathbf{v}_i$
  fim para
até satisfazer o critério de parada

Particle swarm optimization

- Melhoramento: redução linear de *w* (Shi & Eberhart, 1998)
  - Permite ajuste fino nas iterações finais

$$w^{k+1} = w_{\max} - k\left(\frac{w_{\max} - w_{\min}}{k_{\max}}\right)$$

  - Sendo k_{max} o número máximo de iterações
  - Os autores propuseram:
    - $w_{max} = 0{,}9$
    - $w_{min} = 0{,}4$
    - $c_1 = c_2 = 2$

Referências

- Material de:
  - Fabrício Olivetti e Denis Fortunato (UFABC)
  - Marcone Souza (UFOP)
- Artigos:
  - J. Kennedy and R. Eberhart. Particle swarm optimization. In Neural Networks, 1995. Proceedings., IEEE International Conference on, volume 4, pages 1942–1948 vol.4, Nov 1995.
  - Y. Shi and R. Eberhart. A modified particle swarm optimizer. In Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on, pages 69–73, May 1998.