Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências de Computação
Rodrigo Fernandes de Mello
mello@icmc.usp.br

- Tópicos:
  - Introdução
  - Algoritmos Evolutivos:
    - Algoritmos Genéticos
    - Programação Evolutiva
    - Estratégia Evolutiva
    - Programação Genética
  - Inteligência de Enxames
    - Ant Colony Optimization
    - Particle Swarm Optimization

- Conceitos:
  - Com a evolução dos computadores:
    - Tornou-se possível a simulação e análise de problemas mais complexos
  - Baseada na Teoria Evolutiva de Darwin
    - Uma ou mais populações de indivíduos competindo por recursos
    - Há mudanças na população
      - Nascimentos e mortes
    - Conceito de *fitness* ou de aptidão
      - Reflete a habilidade de um indivíduo sobreviver e reproduzir no ambiente
    - Conceito da herança variacional
      - Filhos são similares aos pais, no entanto, não são idênticos

- Podemos propor um algoritmo simples:
- Etapas:
  - Como iremos representar os indivíduos?
  - Algoritmo:
    - Gerar uma população inicial com M indivíduos
    - Do forever:
      - Selecione um membro da população para ser o Pai
      - Utilize o Pai para produzir um filho
        - Similar, mas não igual
      - Selecione um membro da População para morrer

- Melhorando nosso algoritmo:
  - Gerar aleatoriamente uma população inicial de M indivíduos segundo uma distribuição uniforme
  - Computar a função de fitness para os M indivíduos
  - Do Forever:
    - Selecione um Pai de maneira uniforme
    - Utilize o Pai para produzir um Filho
      - Faça uma cópia idêntica do Pai
      - Então probabilisticamente altere (mutação) essa cópia
    - Calcule o fitness para o Filho
    - Selecione um indivíduo da população:
      - Compare seu fitness com o do Filho
      - Aquele com menor fitness (ou pior aptidão) morre

Perceba a estocasticidade do processo!

- A Estocasticidade:
  - Faz com que precisemos executar esse algoritmo por várias iterações
  - Executar para sempre?
    - Não, pois precisamos de uma solução
    - Quando chegamos em uma boa solução?
      - Podemos analisar se esse algoritmo está convergindo em termos de fitness
      - Para isso podemos **plotar o fitness médio da população e seu desvio para cada iteração**
        - Assim chegamos em um número razoável de iterações
        - Ou podemos colocar essa variação como critério de parada automático!
  - Façamos isso para nosso primeiro algoritmo

- Por exemplo:
  - Considere que precisamos otimizar o Caixeiro Viajante:
    - Precisamos definir como é uma solução
      - E quando uma solução será válida
      - Podemos gerar somente soluções válidas para aumentar o desempenho de nosso otimizador?
    - Como será nossa função de fitness?
      - Quanto maior o fitness, maior a adequação daquele indivíduo ao ambiente
    - Implementar
    - Avaliar a média e desvio padrão de fitness da população para cada iteração

- Tendências de custo
  - Tamanho do problema (PI), em que k é o número de cidades (exemplificando com k!)

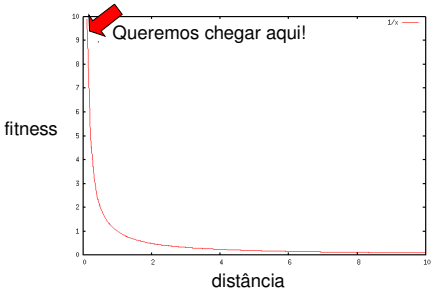
Capacidade do Computador (CC) em MIPS (2008)

PI (k)/CC	1,000	5,000	10,000	50,000	100,000	$20 \cdot 10^9$
$k = 20, 4 \cdot 10^{19}$	$4 \cdot 10^{10}$ s	$9 \cdot 10^{9}$ s	$4 \cdot 10^{9}$ s	$9 \cdot 10^{8}$ s	$4 \cdot 10^{8}$ s	$2 \cdot 10^{3}$ s
$k = 30, 7 \cdot 10^{33}$	$1 \cdot 10^{23}$ m	$2 \cdot 10^{22}$ m	$1 \cdot 10^{22}$ m	$2 \cdot 10^{21}$ m	$1 \cdot 10^{21}$ m	$6 \cdot 10^{15}$ m
$k = 40, 3 \cdot 10^{49}$	$8 \cdot 10^{36}$ h	$1 \cdot 10^{36}$ h	$8 \cdot 10^{35}$ h	$1 \cdot 10^{35}$ h	$8 \cdot 10^{34}$ h	$4 \cdot 10^{29}$ h
$k = 50, 1 \cdot 10^{66}$	$4 \cdot 10^{49}$ y	$9 \cdot 10^{48}$ y	$4 \cdot 10^{48}$ y	$9 \cdot 10^{47}$ y	$4 \cdot 10^{47}$ y	$2 \cdot 10^{42}$ y
$k = 60, 4 \cdot 10^{83}$	$1 \cdot 10^{66}$ d	$3 \cdot 10^{65}$ d	$1 \cdot 10^{65}$ d	$3 \cdot 10^{64}$ d	$1 \cdot 10^{64}$ d	$7 \cdot 10^{58}$ d
$k = 70, 8 \cdot 10^{101}$	10^{84} c	10^{84} c	10^{83} c	10^{83} c	10^{82} c	10^{77} c

Processor	IPS	Year
Pencil and paper (for comparison)	0.0119 IPS	1892
Intel 8080	640 kIPS at 2 MHz	1974
Intel 286	2.66 MIPS at 12 MHz	1982
Intel 386DX	8.5 MIPS at 25 MHz	1988
Intel 486DX	54 MIPS at 66 MHz	1992
Intel Pentium Pro	541 MIPS at 200 MHz	1996
Intel Pentium III	1,354 MIPS at 500 MHz	1999
Pentium 4 Extreme Edition	9,726 MIPS at 3.2 GHz	2003
Intel Core 2 Extreme QX6700	49,161 MIPS at 2.66 GHz	2006
Intel Core 2 Extreme QX9770	59,455 MIPS at 3.2 GHz	2008
Intel Core i7 Extreme 965EE	76,383 MIPS at 3.2 GHz	2008

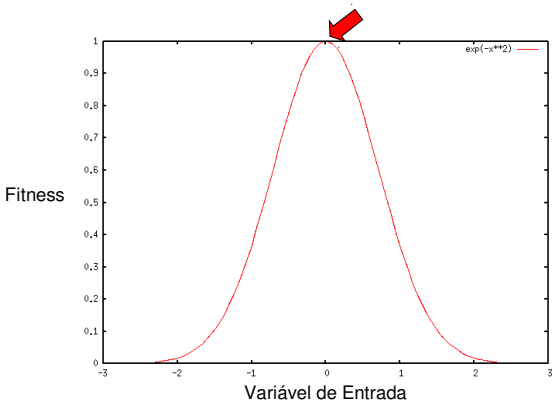
Obtido de: http://en.wikipedia.org/wiki/Instructions_per_second

- Nossa função de fitness (fitness landscape)
  - Considera o inverso da distância percorrida



  - Sendo assim:
    - Se a distância percorrida é pequena, o fitness é alto
    - Se a distância percorrida é grande, o fitness é baixo
  - Um dos principais problemas que temos é definir a função de fitness para nosso problema!!!

- Por exemplo, consideremos outra função de fitness



- Agora que temos uma noção do que se trata:
  - Sewell Wright (1930s)
    - Estudou **superfícies de fitness** e como um sistema evolutivo poderia **explorar tal função**
    - Essa exploração da superfície em busca de uma melhor solução fez com que a área de Computação Evolutiva estivesse muito ligada à **Otimização**
  - Friedman (1956)
    - Podemos ver a Computação Evolutiva como área que permite **modelar sistemas que evoluem ao longo do tempo (dinâmicos)**
    - Assim podemos modelar esses sistemas dinâmicos:
      - Ex: Comportamento do crescimento da população humana
    - Ou até mesmo utilizar a área para controle adaptativo:
      - Ex: existe um sistema A que observamos, coletamos seus dados e utilizamos Computação Evolutiva para adaptar parâmetros desse sistema A

- Nos anos 1960s:
  - Cresceu o interesse por modelar aspectos evolutivos de maneira computacional
    - Isso se deve ao aumento na produção e acesso a computadores digitais
  - Rechenberg (1965)
    - Começa a estudar a solução de **problemas complexos de otimização** usando Computação Evolutiva
  - Fogel et al (1966)
    - Uso de agentes inteligentes
      - Máquinas de estado finito
    - Desenvolve a **programação evolutiva**
  - Holland (1967)
    - Estudou o emprego de Computação Evolutiva para modelar sistemas adaptativos
    - Desenvolveu conceitos que levaram, mais tarde, aos **algoritmos genéticos mais simples**

Computação Evolutiva

- Nos anos 1970s:
  - **Programação Evolutiva:**
    - Considerou essencialmente populações de tamanho fixo (N)
    - Em que cada indivíduo (Pai) produziria somente um Filho
    - Os 2N indivíduos seriam ordenados pelo fitness:
      - Somente os N mais aptos sobreviveriam
    - Fogel et al (1966)
      - Sugere reprodução:
        - Assexuada (mutação)
        - Sexuada (recombinação)

Computação Evolutiva

- Nos anos 1970s:
  - **Estratégia Evolutiva:**
    - Cada indivíduo da população geraria outros k Filhos
    - O melhor dos k+1 indivíduos seria mantido na próxima população
    - Reprodução assexuada (mutação)
      - Observaram que **excesso de mutações** poderia gerar indivíduos muito aleatórios
        - Levando à queda do fitness ao longo das iterações
        - Ou fazendo com que fitness varie demais

Computação Evolutiva

- Nos anos 1970s:
  - **Algoritmos Genéticos:**
    - Objetivo foi o de propor um algoritmo independente do problema abordado (meta-heurística)
    - Representação de indivíduos por meio de strings
      - Strings binárias são as mais comuns nesse período
        - Mutação:
          - Bit-flip
        - Recombinação:
          - 1-point crossover
    - Em primeiros estudos:
      - N Pais geravam N Filhos
      - Filhos substituiam, incondicionalmente, Pais
    - Outros estudos perceberam que se deveria manter parte dos Pais
      - Pais que apresentam maior fitness seriam mantidos (ou teriam maior probabilidade de serem mantidos) para repassar suas características

Computação Evolutiva

- Nos anos 1980s para frente:
  - **Melhorias constantes na forma de:**
    - Gerar populações
    - Definir funções de fitness mais adequadas para determinados problemas
    - Operadores de mutação e recombinação

Computação Evolutiva: Nosso Primeiro Algoritmo

- Algoritmo **Versão 1 ou Modelo Steady State**:
  - Gerar aleatoriamente uma população inicial de M indivíduos segundo uma distribuição uniforme
  - Computar a função de fitness para os M indivíduos
  - Do Forever:
    - Selecione um Pai de maneira uniforme
    - Utilize o Pai para produzir um Filho
      - Faça uma cópia idêntica do Pai
      - Então probabilisticamente altere (mutação) essa cópia
    - Calcule o fitness para o Filho
    - Selecione um indivíduo da população:
      - Compare seu fitness com o do Filho
      - Aquele com menor fitness morre

Computação Evolutiva: Nosso Primeiro Algoritmo

- Algoritmo **Versão 2 ou Modelo em Batch ou de Geração**:
  - Gerar aleatoriamente uma população inicial de M indivíduos segundo uma distribuição uniforme
  - Computar a função de fitness para os M indivíduos
  - Do Forever:
    - Para k Filhos
      - Selecione um Pai de maneira uniforme
      - Utilize o Pai para produzir um Filho
        - Faça uma cópia idêntica do Pai
        - Então probabilisticamente altere (mutação) essa cópia
      - Calcule o fitness para o Filho
      - Mantenha o Filho em uma População Separada
    - Para i de 1 até k:
      - Force o Filho i a competir com um indivíduo aleatório da População Pai
      - Compare seu fitness com o do Filho
      - Aquele com menor fitness morre

Computação Evolutiva: Nosso Primeiro Algoritmo

- Implementação:
  - Comparar a média e desvio padrão do fitness dos dois algoritmos para um dado problema

Computação Evolutiva: Programação Evolutiva

- Uma questão relativa a nosso primeiro algoritmo:
  - Como um Pai é selecionado de maneira aleatória (uniforme) para reprodução, pode ser que **algum desses Pais nunca seja selecionado**
    - Pior, pode ser que esse **Pai tenha alto fitness e poderia gerar um bom filho**
  - De maneira similar, a seleção aleatória de um Pai para competir com um Filho pode fazer com que um Pai nunca seja selecionado
    - Pior, pode ser que esse **Pai tenha baixo fitness e não seja removido da próxima população**
- Podemos, portanto, deixar nosso primeiro **algoritmo mais determinístico**:
  - Assumir que a população de Pais e Filhos tem mesmo tamanho
  - Logo...

Computação Evolutiva: Programação Evolutiva

- Algoritmo (**Primeiro Algoritmo de Programação Evolutiva**, Fogel et al 1966):
  - Gerar aleatoriamente uma população inicial de M indivíduos segundo uma distribuição uniforme
  - Computar a função de fitness para os M indivíduos
  - Do Forever:
    - Para i de 1 até M
      - Selecione o Pai i e o utilize para produzir um Filho
        - Faça uma cópia idêntica do Pai
        - Então probabilisticamente altere (mutação) essa cópia
      - Calcule o fitness para o Filho
      - Mantenha o Filho em uma População Separada
    - Unifique as populações:
      - Ordene os 2M indivíduos pelo fitness
      - Mantenha apenas os M indivíduos mais aptos

Computação Evolutiva: Programação Evolutiva

- Qual a diferença entre os algoritmos?
  - Programação Evolutiva:
    - **Determinístico** no sentido de que cada Pai gera, obrigatoriamente, um filho
    - Converge mais rapidamente
      - Pois emprega um conceito mais forte de **"elitismo"**
        - Elitismo é o conceito de sobrevivência do mais apto
  - Essa técnica:
    - Tem menor aleatoriedade
    - Portanto, para determinados problemas, **pode tender a um mínimo local**
      - Depende do formato da função de fitness
  - Logo:
    - **O equilíbrio entre o componente de variação populacional** (por exemplo, por mutação) **versus o componente de redução de variação** (seleção de indivíduos para uma próxima população) é um fator importante a ser considerado no projeto de Algoritmos Evolutivos

Computação Evolutiva: Estratégias Evolutivas

- Os algoritmos anteriores:
  - Produzem um pequeno número de Filhos
  - Pode ser que deixemos de explorar possibilidades com isso
- Algoritmos de Estratégia Evolutiva:
  - Produzem maior número de Filhos
  - Podemos simplesmente mudar os algoritmos anteriores
    - Surgem, no entanto, duas questões:
      - Quantos Filhos cada Pai deveria produzir?
      - Dado o fato que os Pais são mais utilizados ou melhor explorados para produzir Filhos, não poderíamos reduzir a população de Pais?

Computação Evolutiva: Estratégias Evolutivas

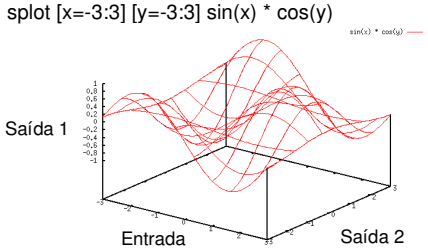
- Implementação:
  - Que tal experimentar a Versão 2 de nosso primeiro algoritmo com:
    - Uma população inicial de 1 Pai gerando 10 Filhos?
    - Observemos a média e desvio padrão de fitness
  - Como mantemos somente um indivíduo:
    - Alguns podem pensar que esse algoritmo, por ser ainda mais elitista, convergiria mais rapidamente
    - No entanto:
      - A população tem uma grande importância na busca por melhores soluções (auxilia a explorar melhor o espaço de possíveis soluções ao longo das iterações)
      - Pode ser que um indivíduo da população esteja mais próximo de uma boa solução
        - Enquanto tendo apenas um indivíduo, ele demora para convergir!

Computação Evolutiva: Estratégias Evolutivas

- Perceberam que técnica poderia considerar a seguinte situação:
  - Se mutações estão trazendo bons resultados
  - Então deixemos mais mutações ocorrerem
- Rechenberg (1965)
  - Propõe a seguinte técnica chamada 1:5
    - **Se mais de 20% dos indivíduos mutados vencem na competição, então o percentual de mutação utilizado é muito conservativo**
    - **Se menos de 20% dos indivíduos mutados vencem na competição, então a variação causada pela mutação é negativa**
      - Muito explorativa
      - Taxa de mutação deveria ser reduzida
  - Vamos implementar a Versão 2 de nosso primeiro algoritmo usando esse conceito adaptativo de mutação
    - Verifiquemos os resultados de média e desvio padrão do fitness
    - Espera-se convergir mais rapidamente!

Computação Evolutiva: Estratégias Evolutivas

- Rechenberg (1965)
  - Há alguns problemas:
    - Estamos tratando funções de fitness unidimensionais
      - Pois somente um valor é utilizado para verificar se um indivíduo é mais apto que outro
    - Como alterar a mutação para casos multidimensionais?
      - Exemplo: uma superfície em que no eixo x definimos a entrada e obtemos duas saídas (y e z)
      - splot [x=-3:3] [y=-3:3] sin(x) * cos(y)



Computação Evolutiva: Algoritmos Genéticos

- Algoritmo Genético Versão 1:
  - Gerar uma população de M indivíduos
    - Cada indivíduo é também chamado de cromossomo
    - Os elementos do indivíduo são chamados genes
  - Repetir:
    - Computar e manter o fitness u(i) para cada indivíduo i na população
    - Para gerar M Filhos:
      - Selecionar, aleatoriamente, 2 Pais para a reprodução
        - Cada Pai tem probabilidade proporcional a seu fitness u(i) de ser selecionado
      - Cruzar Pais (usando crossover) e realizar mutação no filho segundo certa probabilidade
    - Manter somente os Filhos na população seguinte

- Permite combinar boas características dos indivíduos usando recombinação (do inglês crossover)
  - Implementar com crossover de 1 ponto
    - Comparar com os demais
  - Pode-se implementar também com crossover de n pontos

Computação Evolutiva: Algoritmos Genéticos

- Pode-se gerar novas versões para o Algoritmo Original:
  - Outros **métodos de seleção** para operação de mutação ou crossover:
    - **Proporcional ou Roleta** - indivíduos são selecionados de acordo com o fitness proporcional. Pode-se criar uma roleta dividida em **n** partes (sendo **n** o número de indivíduos). Cada parte com um tamanho relativo ao fitness do cromossomo. Utiliza-se um gerador de números aleatórios e compara-se com a roleta
    - **Ranking** - cromossomos são primeiramente ordenados de acordo com o fitness, depois cada cromossomo recebe um novo valor de fitness de acordo com sua posição na lista ordenada. Depois aplica-se um procedimento que seleciona indivíduos de acordo com suas posições no ranking. O procedimento deve dar maior chance de seleção para aqueles que estão no topo do ranking.
      - Geralmente utiliza-se um ranking que reduza as diferenças entre o fitness do melhor e do pior indivíduo
      - Assim, este método pode prevenir que indivíduos muito aptos dominem o processo de seleção. Assim outros indivíduos também têm chances.
    - **Torneio** - um torneio de tamanho k é definido (exemplo: k=2). Então k indivíduos são sorteados e seus fitness comparados. Aquele com melhor fitness é selecionado. Esses passos podem ocorrer por diversas vezes.

Computação Evolutiva: Algoritmos Genéticos

- Pode-se gerar novas versões para o Algoritmo Original:
  - Pode-se manter alguns Pais na população seguinte
    - Os melhores
    - Ou escolha aleatória, por exemplo

Computação Evolutiva: Algoritmos Genéticos

- Questões relevantes:
  - Há problemas em que os operadores de mutação e crossover devem ser adaptados
    - Pode ser que um indivíduo não seja representado por um vetor binário, mas sim por um vetor de inteiros ou caracteres, ou até mesmo por uma matriz
    - Formas mais complexas de codificação de indivíduos também têm sido propostas
      - Codificar usando árvore, por exemplo
  - Pode-se propor novos operadores
    - Isso tem se tornado comum para resolver problemas específicos
  - Há ainda problemas multiobjetivo
    - Podemos simplificar vários objetivos em uma única função de fitness
    - Ou podemos utilizar dois ou mais objetivos em conjunto

Computação Evolutiva: Inteligência de Enxames

- Inteligência de Enxames
  - Explora comportamento coletivo e descentralizado
  - Baseado no conceito de sistemas emergentes
    - Partes atuam e favorecem algo em comum
      - Ex: Colônias de Formigas, Abelhas, formação do vôo de pássaros, etc.
- Expressão introduzida por Beni e Wang (1989)
- Agentes independentes interagem localmente

Computação Evolutiva: Inteligência de Enxames

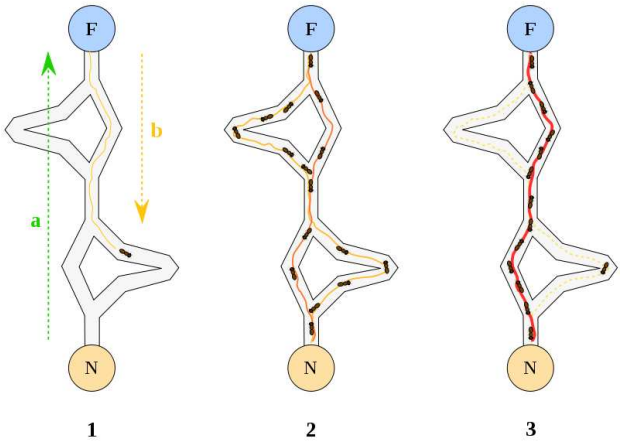
- Ant Colony Optimization:
  - Meta-heurística proposta por Dorigo (1992)
    - Não dá garantias de encontrar a melhor solução
  - Técnica probabilística para resolver problemas computacionais definidos sobre grafos
  - Mimetiza o comportamento de formigas em busca de alimento
  - Diversos estudos sobre formigas foram feitos anteriomente
    - Dorigo se baseou nesses estudos para propor ACO
  - Assim como as demais meta-heurísticas:
    - NÃO garante solução ideal
    - Pode convergir para mínimo local!

Computação Evolutiva: Inteligência de Enxames

- Ant Colony Optimization:
  - Funcionamento:
    - Inicialmente, formigas caminham aleatoriamente sobre o grafo até encontrar uma fonte de alimento
    - Recuperam o alimento e tentam voltar para a colônia ou ninho
    - Formigas deixam feromônio por onde passam
      - Feromônio dá resposta positiva para outras formigas escolherem certo caminho
      - Feromônio evapora ao longo do tempo (resposta negativa)

Computação Evolutiva: Inteligência de Enxames

- Ant Colony Optimization:



http://en.wikipedia.org/wiki/File:Aco_branches.svg

Computação Evolutiva: Inteligência de Enxames

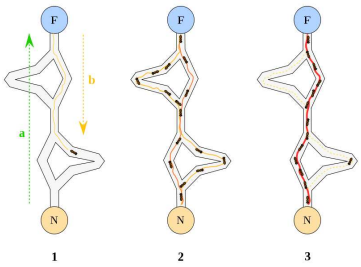
- Ant Colony Optimization:
  - Estigmergia é o nome dado ao fato das formigas utilizarem o meio ambiente para se comunicar (por meio de feromônio)
    - Outros animais fazem o mesmo:
      - Cupins
      - Cachorros
  - ACO é baseado em:
    - Feedback positivo – feromônio deixado em um caminho
    - Feedback negativo – evaporação de feromônio evita que formigas “viciem” em um caminho
      - Ou seja, evita mínimos locais

Computação Evolutiva: Inteligência de Enxames

- Ant Colony Optimization:
  - Tem vantagens sobre demais técnicas de otimização, tais como Algoritmos Genéticos e Simulated Annealing:
    - Pois continuamente evolui
    - Grafo pode mudar dinamicamente, que formigas se adaptam
- Aplicações:
  - Roteamento de pacotes na Internet
  - Caixeiro Viajante e variações
  - Roteamento de veículos
  - Escalonamento de processos
    - Recursos (alimento)
  - Demais problemas que podem ser mapeados em grafos

Computação Evolutiva: Inteligência de Enxames

- Algoritmo Ant Colony System
  - Escolha do caminho é probabilística:
    - Quantidade de feromônio no caminho entre os vértices x e y τ_{xy}
    - Parâmetro que controla a influência do feromônio $\alpha \geq 0$
    - Parâmetro que controla o desejo da formiga em seguir o caminho: $\beta \leq 1$
      - Esse desejo é, geralmente, o inverso da distância η_{xy}^{β}
    - Usualmente o inverso da



$$p_{xy}^{k} = \frac{(\tau_{xy}^{\alpha})(\eta_{xy}^{\beta})}{\sum(\tau_{xy}^{\alpha})(\eta_{xy}^{\beta})}$$

Computação Evolutiva: Inteligência de Enxames

- Algoritmo Ant Colony System
  - A atualização de feromônio ocorre ao longo das iterações:

$$\tau_{xy}(t+1) = (1-\rho)\tau_{xy}(t) + \Delta\tau_{xy}$$

    - Rho é o coeficiente de evaporação
    - O termo delta se refere à quantidade de feromônio deixada no caminho xy naquela determinada iteração:
      - $\Delta\tau_{xy}$ é igual a 0 se nenhuma formiga utilizou aquele caminho ou
      - Igual a Q/L, em que Q é uma constante que representa a quantidade de feromônio deixada e L é o custo do caminho, tipicamente sua distância

Computação Evolutiva: Inteligência de Enxames

- Algoritmo Ant Colony System
  - Implementação

Computação Evolutiva: Inteligência de Enxames

- Particle Swarm Optimization:
  - Meta-heurística iterativa atribuída a Kennedy e Eberhart (1995) e Eberhart e Shi (1998) para abordar problemas de otimização
    - NÃO dá garantias de encontrar a melhor solução
  - Um conjunto de soluções candidatas é proposto
    - Cada solução é representada por uma partícula
    - Partículas se movem sobre a função de custo
      - Função a ser otimizada
    - Para que se movam partículas têm:
      - Uma posição inicial
      - Uma velocidade

Computação Evolutiva: Inteligência de Enxames

- Particle Swarm Optimization:
  - Iterativamente mantemos:
    - A melhor posição de uma partícula
    - A melhor posição obtida por todas partículas
  - Baseada no comportamento de:
    - Formação de vôo de pássaros
    - Formação de peixes
  - A função de custo a ser minimizada não precisa ser diferenciável:
    - Gradiente descendente requer

Computação Evolutiva: Inteligência de Enxames

- Particle Swarm Optimization:
  - Algoritmo:
    - Para cada partícula i=1,2,...,S
      - Inicialize a posição $\mathbf{x}_i$ da partícula aleatoriamente (usando uma distribuição uniforme) dentro do intervalo de interesse U(lower, upper)
      - Inicialize a melhor posição $\mathbf{p}_i$ para a partícula até o momento como sua própria posição inicial
      - Se o custo f($\mathbf{p}_i$) < f($\mathbf{g}$), atualize a melhor posição global $\mathbf{g}$ do sistema
      - Inicialize a velocidade $\mathbf{v}_i$ da partícula no intervalo U(- | upper - lower | , | upper - lower |)

Computação Evolutiva: Inteligência de Enxames

- Particle Swarm Optimization:
  - Algoritmo:
    - Enquanto o critério de parada não for atendido (número de iterações, qualidade mínima da solução ou variação de qualidade da solução global):
      - Para cada partícula i=1,2,...,S
        - Gerar dois números aleatórios r_p e r_g ~ U(0,1)
        - $v_i(t+1) = \omega v_i(t) + \phi_p r_p(p_i - x_i) + \phi_g r_g(g - x_i)$
          - phis e omega são definidos pelo projetista
        - Atualizar $x_i(t+1) = x_i(t) + v_i(t+1)$
        - Se f($\mathbf{x}_i$) < f($\mathbf{p}_i$) então $\mathbf{p}_i = \mathbf{x}_i$
          - Se f($\mathbf{p}_i$) < f($\mathbf{g}$) então $\mathbf{g} = \mathbf{p}_i$
      - Finalmente, melhor solução estará em g

Computação Evolutiva: Inteligência de Enxames

- Particle Swarm Optimization:
  - Algoritmo:
    - Parâmetros:
      - Omega – define o quanto manteremos da velocidade anterior
      - Phis – pode alterar o tamanho do passo
        - Influencia no aumento ou redução da velocidade
        - Isso influencia no tamanho do passo que daremos para uma próxima solução
    - Se queremos o máximo de uma função, basta definir a função de custo como h(.) = -f(.)

Computação Evolutiva: Inteligência de Enxames

- Implementação:
  - Primeiramente:
    - Definir problema
    - Definir função de custo
    - Implementação do algoritmo
    - Definição de parâmetros
      - Observar comportamento quando há variação nos parâmetros

Referências

- Henneth A. de Jong, Evolutionary Computation: A Unified Approach, MIT, 2006
- M. Dorigo, Optimization, Learning and Natural Algorithms, PhD thesis, Politecnico di Milano, Italie, 1992
- Kennedy, J.; Eberhart, R. (1995). "Particle Swarm Optimization". Proceedings of IEEE International Conference on Neural Networks. IV. pp. 1942–1948. doi:10.1109/ICNN.1995.488968.
- Shi, Y.; Eberhart, R.C. (1998). "A modified particle swarm optimizer". Proceedings of IEEE International Conference on Evolutionary Computation. pp. 69–73.
- Kennedy, J.; Eberhart, R.C. (2001). Swarm Intelligence. Morgan Kaufmann. ISBN 1-55860-595-9.