Computação Evolutiva e Algoritmos Populacionais

Cleber Zanchettin

Tópicos

- Simulated Annealing
- Tabu Search
- Algoritmos Evolutivos
  - Algoritmos Genéticos
  - Programação Evolutiva
  - Estratégia Evolutiva
  - Programação Genética
- Inteligência de Enxames
  - Ant Colony Optimization
  - Particle Swarm Optimization

Simulated Annealing e Tabu Search

Cleber Zanchettin
UFPE - Universidade Federal de Pernambuco
CIn - Centro de Informática

Hill Climbing (Subida da Encosta)

- **Problema de Otimização:**
  - Seja *S* o conjunto de soluções possíveis, em que cada solução tem um *custo* associado.
  - Objetivo: Encontrar a solução com menor custo.
- **Método Hill Climbing:**
  - 1 - Escolhe aleatoriamente uma solução.
  - 2 - Gera uma nova solução (vizinha) a partir da atual.
  - 3 - Se *custo* (solução nova) < *custo* (solução atual),
    - Aceita solução nova.
  - Se não,
    - Não aceita solução nova (continua com a atual).
  - 4 - Repete 2 e 3 até terminarem as iterações permitidas.

Hill Climbing (Subida da Encosta)

- **Problema do Método Hill Climbing:**
  - Pode ficar **preso em mínimos locais** (pode estacionar em um mínimo local, sendo que todas as novas soluções geradas têm custo maior).
- **Simulated Annealing:**
  - Procura minimizar este problema, **podendo aceitar vizinhos piores com uma dada probabilidade**, que diminui ao longo da execução.
- **Tabu Search:**
  - Procura minimizar este problema, aceitando sempre os vizinhos, sejam melhores ou piores, **guardando a melhor solução visitada e evitando ciclos** na trajetória de busca.

Simulated Annealing

- **Idéia básica:**
  - Se a nova solução gerada for **melhor** que a atual (melhor = menor custo), esta nova solução **é aceita**.
  - Se for pior, a nova solução **pode ser aceita** com uma dada **probabilidade**.
  - Esta probabilidade é controlada por um parâmetro chamado de **temperatura**, que **diminui** ao longo das iterações.

Simulated Annealing

- Estrutura básica:
  - 1 - Escolhe aleatoriamente uma solução.
  - 2 - Gera uma nova solução (vizinha) a partir da atual.
  - 3 - Se *custo* (solução nova) < *custo* (solução atual),
    - Aceita solução nova.
  - Se não,
    - Aceita solução nova com probabilidade:

      p = *exp* [–(c*usto*(sol. nova) – *custo*(sol. atual)) / Temperatura]
  - 4 - Repete 2 e 3 até terminarem as iterações permitidas.
- Observação:
  - O parâmetro 'Temperatura' vai diminuindo a cada *N* iterações.

Simulated Annealing

– Probabilidade de aceitar a nova solução (se o custo aumentar):

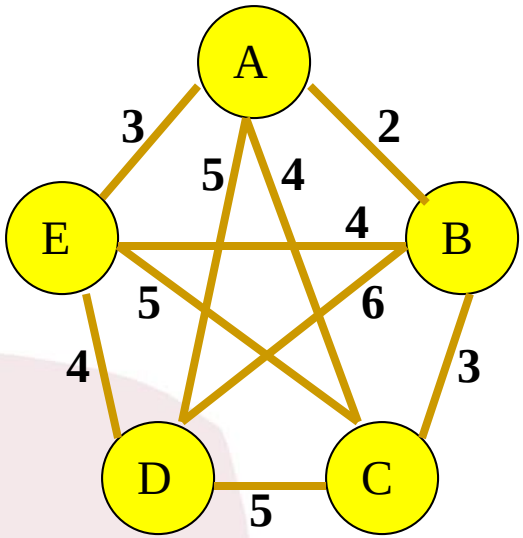
$p = exp$ [– (*custo*(solução nova) – *custo*(solução atual)) / Temperatura].

- **Mantendo fixo o aumento no custo, se a temperatura diminuir, a probabilidade diminui.**
  - Com o aumento das iterações, fica mais difícil aceitar soluções que pioram o custo.
- **Mantendo fixa a temperatura, se a variação no custo aumentar, a probabilidade diminui.**
  - Em uma mesma iteração, quanto maior o aumento no custo, mais difícil é aceitar a nova solução.

Simulated Annealing

- Implementação:
  - **Representação das soluções:**
    - Como as soluções serão consideradas no espaço de busca.
  - **Função de custo (ou avaliação):**
    - ‘Nota’ atribuída a cada solução.
  - **Operador (ou mecanismo de geração de vizinhos):**
    - Como novas soluções serão geradas a partir da atual.
  - **Esquema de esfriamento:**
    - Como a temperatura será reduzida ao longo da execução.
  - **Máximo número de iterações.**

Problema do Caixeiro Viajante



- **Representação das soluções:**
  - Seqüência de cidades do percurso.
  - Ex.: s = [B,D,E,C,A]
- **Função de custo (ou avaliação):**
  - Distância do percurso.
  - Ex.: custo(s) = 6+4+5+4+2 = 21
- **Operador (geração de vizinhos):**
  - Permutar 2 cidades consecutivas escolhidas aleatoriamente.
  - Ex.: s' = [B,E,D,C,A]
- **Esquema de esfriamento:**
  - Temperatura inicial: $T_0 = 1$
  - Regra de esfriamento: $T_{i+1} = 0.9\ T_i$
- **Máximo de 4 iterações.**

Problema do Caixeiro Viajante

Iteração	Temp.	Solução	Custo	Aleatório	Probabilidade	Aceita?
0	1	BDECA	21			
1	0.9	BDEAC	20			S
2	0.81	BDAEC	22	0.02	$e^{-(22-20)/0.81} = 0.085$	S
3	0.73	BADEC	19			S
4	0.66	BADCE	21	0.13	$e^{-(21-19)/0.66} = 0.047$	N

Solução final: BADEC – Custo: 19

Tabu Search

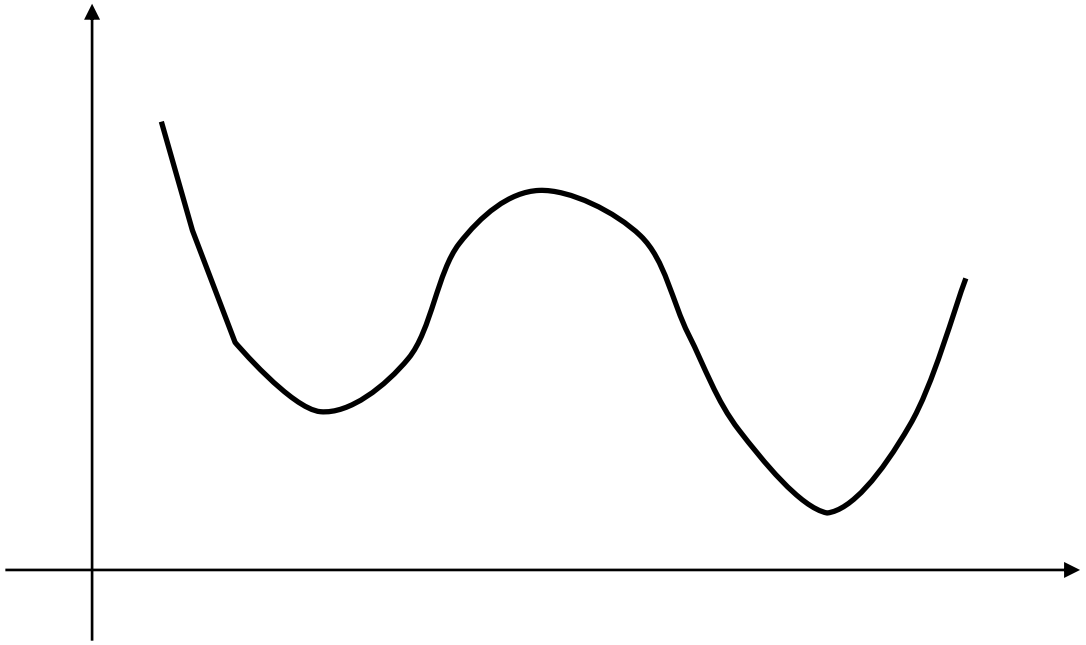
- Idéia básica:
  - A partir da solução atual, gerar um **conjunto de novas soluções.**
  - Aceitar sempre a **melhor solução deste conjunto.**
    - Pode ser melhor ou pior que a solução atual.
    - Pode haver ciclos na trajetória (aceitar soluções que já foram visitadas).
  - Guardar na memória:
    - A **melhor solução encontrada** desde o início da execução.
    - Uma **Lista Tabu**, contendo as ***K* soluções mais recentemente visitadas**. Estas soluções são proibidas (para evitar ciclos).
  - A **solução final** dada pelo algoritmo é a **melhor solução encontrada** desde o início da execução, e não a última.

Tabu Search

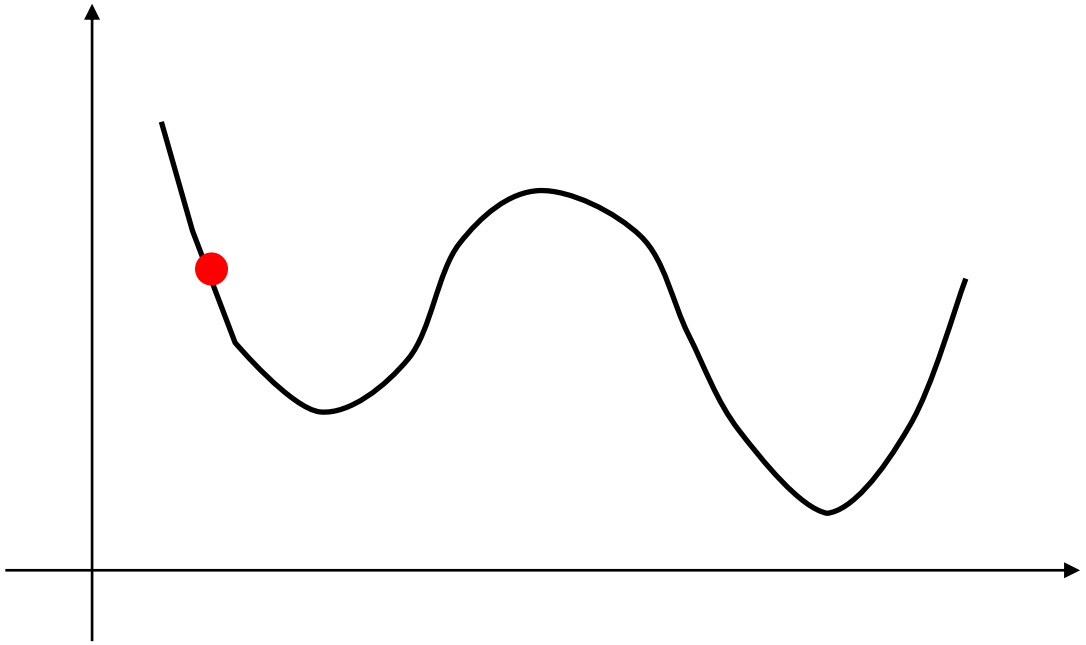
- Estrutura básica:
  - 1 - Escolhe aleatoriamente uma solução.
  - 2 - Guarda a solução em *melhor solução* e na *lista tabu*.
  - 3 - Gera um conjunto de *N* soluções vizinhas à atual.
  - 4 - Aceita a solução de menor custo entre os *N* vizinhos (que não esteja na *lista tabu*).
  - 5 - Atualiza *melhor solução* e insere a nova solução na *lista tabu*.
  - 6 - Repete 3 a 5 até terminarem as iterações permitidas.
  - 7 - Retorna *melhor solução.*
- Observação:
  - A *lista tabu* guarda as *K* soluções mais recentemente visitadas.

Busca Tabu

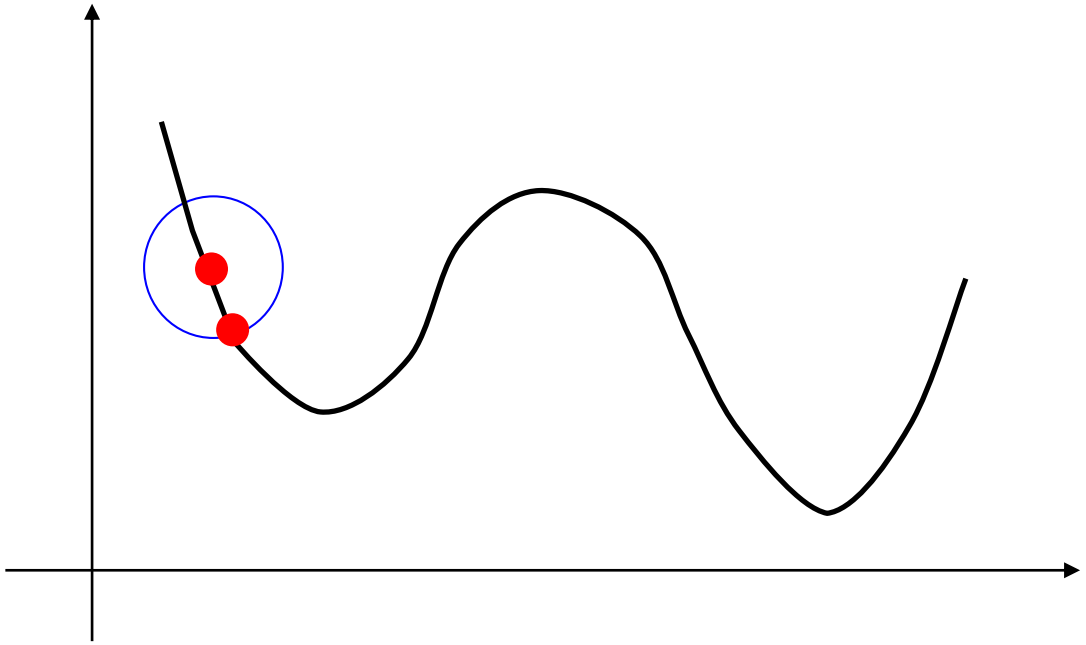
Fred Glover (1986) & Pierre Hansen (1986)



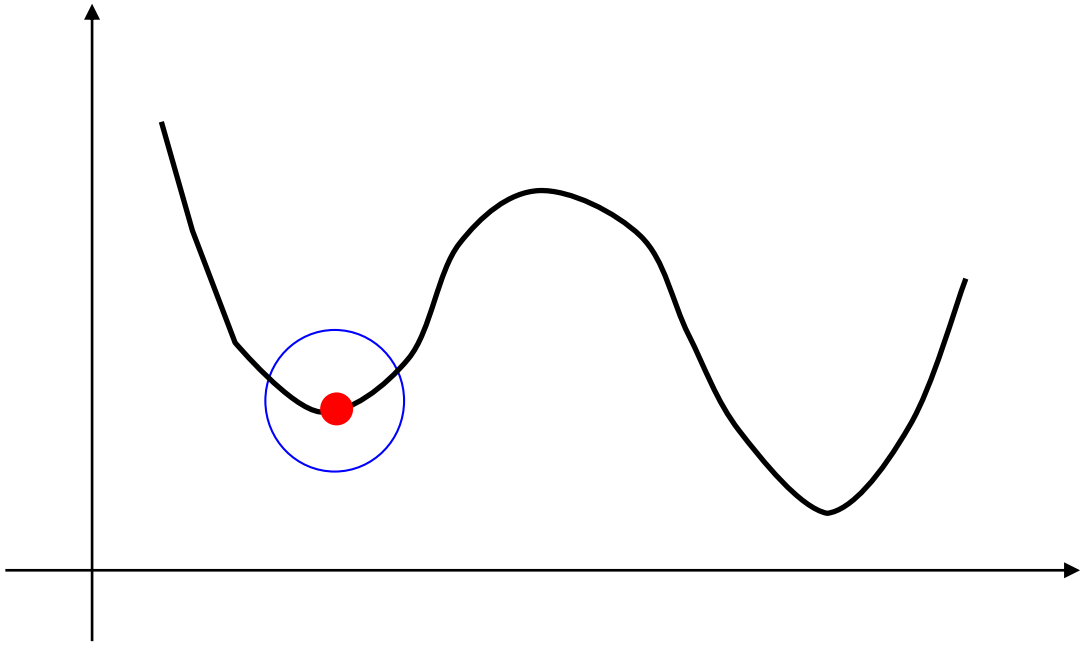
1º Princípio: Mover para o melhor vizinho



1º Princípio: Mover para o melhor vizinho

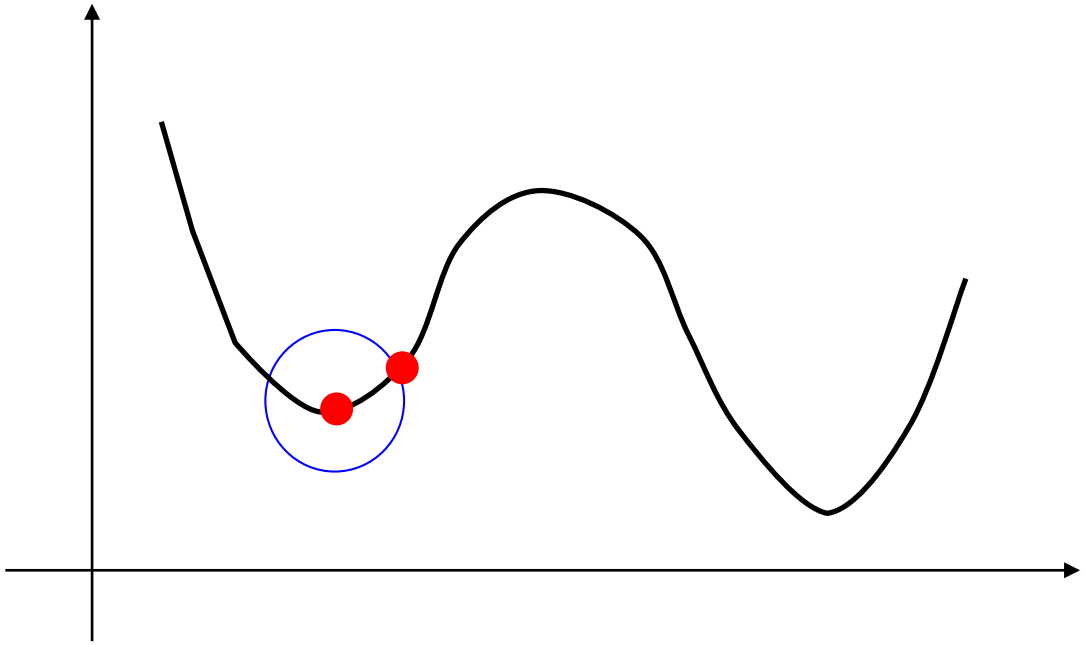


1º Princípio: Mover para o melhor vizinho



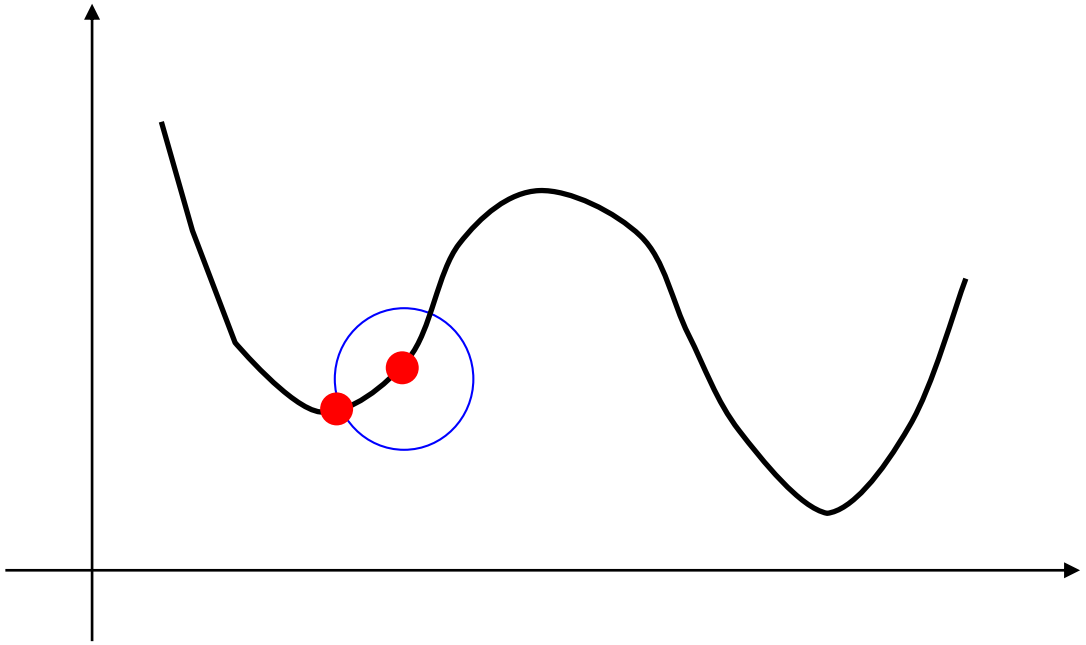
Heurística de descida: Fica-se preso no primeiro ótimo local

1º Princípio: Mover para o melhor vizinho



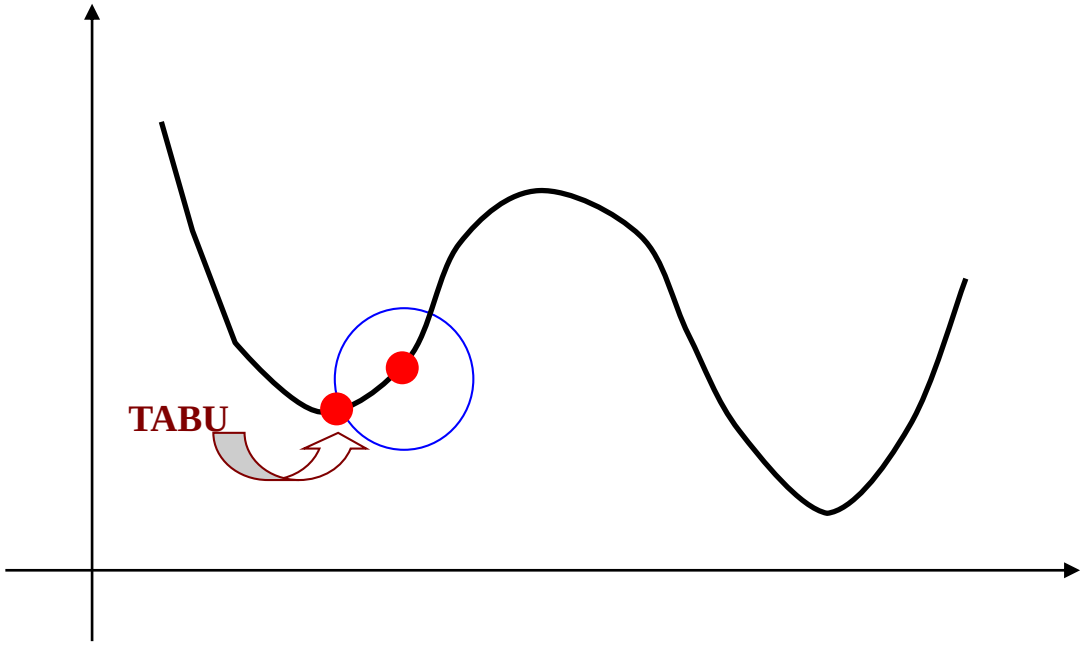
O melhor vizinho pode ser de piora!

1º Princípio: Mover para o melhor vizinho

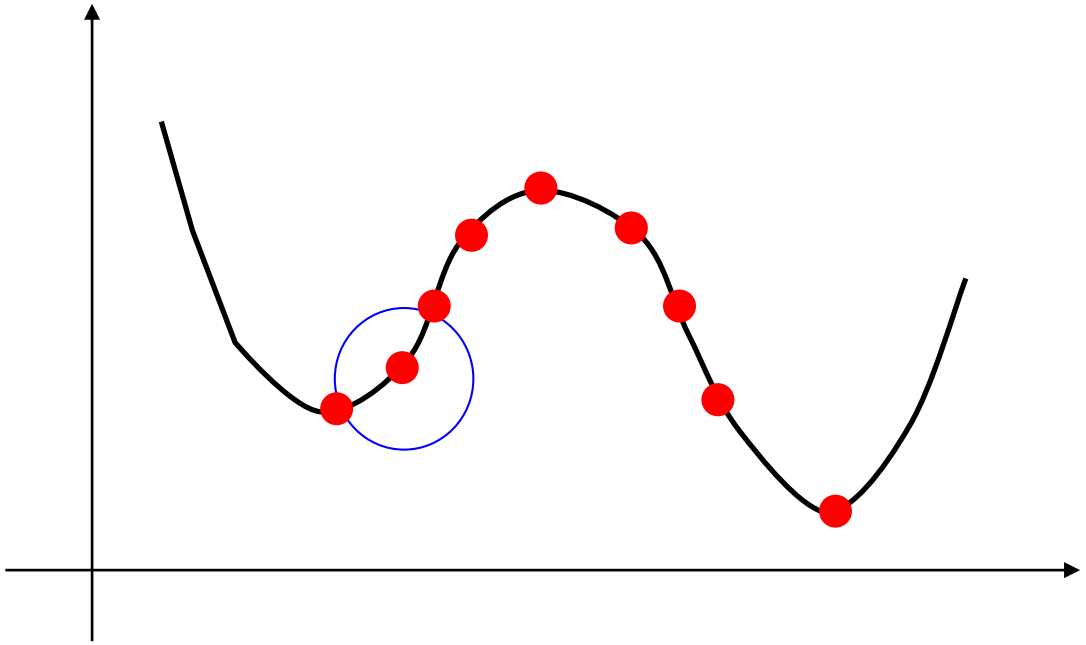


Problema: Ciclagem

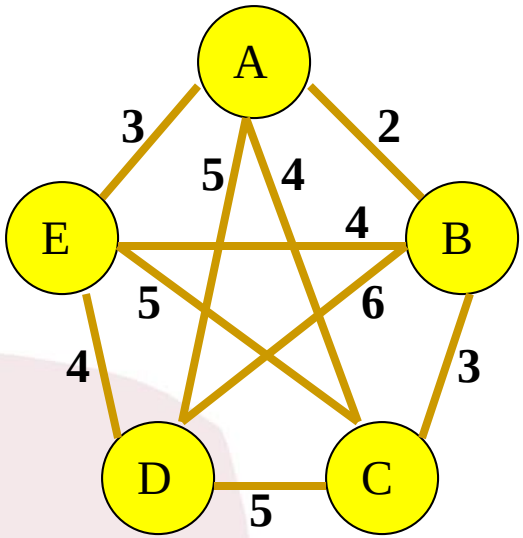
2º Princípio: Criar Lista Tabu



2º Princípio: Criar Lista Tabu



Problema do Caixeiro Viajante



- **Representação das soluções:**
  - Seqüência de cidades do percurso.
  - Ex.: s = [B,D,E,C,A]
- **Função de custo (ou avaliação):**
  - Distância do percurso.
  - Ex.: custo(s) = 6+4+5+4+2 = 21
- **Operador (geração de vizinhos):**
  - Permutar 2 cidades consecutivas, gerando 5 vizinhos por iteração.
  - Ex.: s_1 = [D,B,E,C,A]
    s_2 = [B,E,D,C,A]
    s_3 = [B,D,C,E,A]
    s_4 = [B,D,E,A,C]
    s_5 = [A,D,E,C,B]
- **Máximo de 2 iterações.**

Problema do Caixeiro Viajante

Iteração	Solução – Custo	Lista Tabu	Melhor Solução – Custo	Vizinhos – Custo
0	BEDCA – 19	BEDCA	BEDCA – 19	EBDCA – 22
				BDECA – 21
				BECDA – 21
				BEDAC – 20
				AEDCB – 17 ←
1	AEDCB – 17	BEDCA	AEDCB – 17	EADCB – 20
		AEDCB		ADECB – 19 ←
				AECDB – 21
				AEDBC – 20
				BEDCA – 19 X

Problema do Caixeiro Viajante

Iteração	Solução – Custo	Lista Tabu	Melhor Solução – Custo	Vizinhos – Custo
1	AEDCB – 17	BEDCA	AEDCB – 17	EADCB – 20
		AEDCB		ADECB – 19 ←
				AECDB – 21
				AEDBC – 20
				BEDCA – 19 X
2	ADECB – 19	BEDCA	AEDCB – 17	DAECB – 22
		AEDCB		AEDCB – 17 X
		ADECB		ADCEB – 21
				ADEBC – 20 ←
				BDECA – 21

Solução final: AEDCB – 17

Observações Finais

- **Lista Tabu:**
  - Para problemas em que a solução é uma **seqüência binária**, é comum armazenar na lista tabu **apenas a posição do bit que foi modificado**, em vez de guardar toda a solução.
- **Esforço computacional por iteração:**
  - Uma iteração de Tabu Search exige **mais esforço computacional** do que uma iteração de Simulated Annealing (mais operações).
  - Porém, em geral, Tabu Search precisa de menos iterações para convergir, pois avalia um **conjunto de vizinhos** a cada iteração.
- **Variações dos algoritmos:**
  - Diversas melhorias foram propostas para Simulated Annealing e Tabu Search; o que foi visto nesta aula são apenas as noções básicas destes algoritmos.

- Applet

http://www.heatonresearch.com/aifh/vol1/tsp_anneal.html
http://www.staff.science.uu.nl/~beuke106/anneal/anneal.html

http://siebn.de/other/tabusearch/
http://prof.if.ktu.lt/~jonas.mockus/tabu/tsp.html

Bibliografia

– D.T. Pham and D. Karaboga, “Introduction”, Intelligent Optimisation Techniques (Edited by D.T. Pham and D. Karaboga), pp. 1-50, Springer-Verlag, 2000.

- **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?

- 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**

- 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**?

- 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 a cada iteração**
        - Assim chegamos em um número **razoável de iterações**
        - Ou podemos colocar essa **variação** como critério de **parada** automático!

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: 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?

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

- 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 *conservative***
    - **Se menos de 20% dos indivíduos mutados vencem na competição, então a variação causada pela mutação é *negative***
      - Muito *explorative*
      - **Taxa de mutação** deveria ser reduzida

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*)

Problema 1

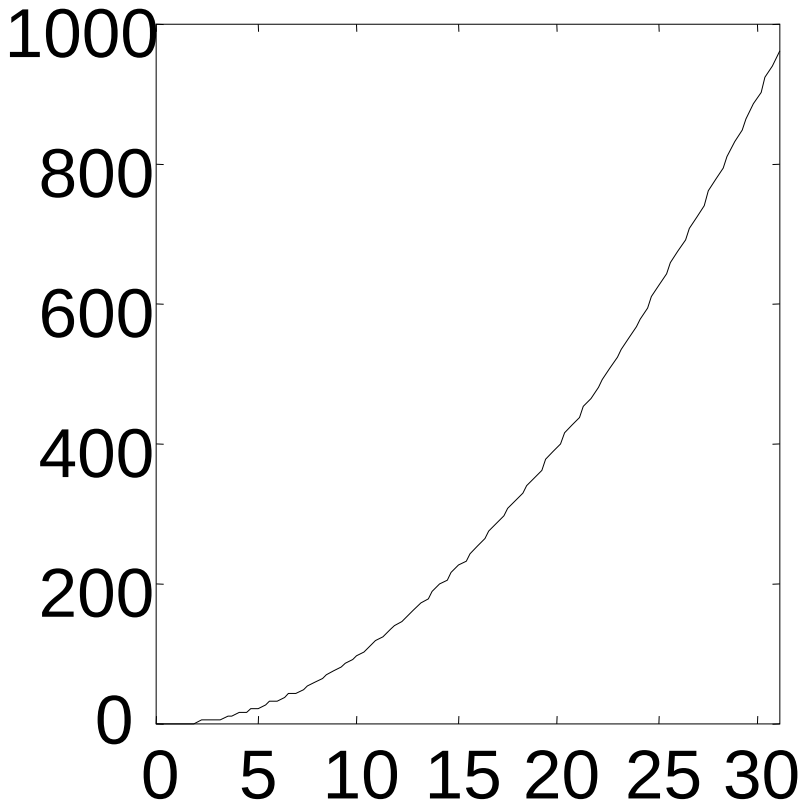
Problema: Use um AG para encontrar o ponto máximo da função:

$$f(x) = x^2$$

com x sujeito as seguintes restrições:

$$0 \leq x \leq 31$$

x é inteiro



Indivíduo

- Cromossomo
  - Estrutura de dados que representa uma possível solução para o problema.
  - Os parâmetros do problema de otimização são representados por cadeias de valores.
  - Exemplos:
    - Vetores de reais, (2.345, 4.3454, 5.1, 3.4)
    - Cadeias de bits, (111011011)
    - Vetores de inteiros, (1,4,2,5,2,8)
    - ou outra estrutura de dados.

Individuo (II)

- Aptidão
  - Nota associada ao indíviduo que avalia quão boa é a solução por ele representada.
- Aptidão pode ser:
  - Igual a função objetivo (raramente usado na prática).
  - Resultado do **escalonamento** da função objetivo.
  - Baseado no **ranking** do indíviduo da população.

Cromossomo do Problema 1

- Cromossomos binários com 5 bits:
  - 0 = 00000
  - 31 = 11111
- Aptidão
  - Neste problema, a aptidão pode ser a própria função objetivo.
  - Exemplo:

$$\text{aptidão}(00011) = f(3) = 9$$

Seleção

- Seleção
  - **Imitação** da seleção natural.
  - Os **melhores indivíduos** (maior aptidão) são selecionados para gerar filhos através de *crossover* e mutação.
  - **Dirige** o AG para as melhores regiões do espaço de busca.
- Tipos mais comuns de seleção
  - **Proporcional a aptidão**.
  - **Torneio**.

População Inicial do Problema 1

É **aleatória** (mas quando possível, o conhecimento da aplicação pode ser utilizado para definir a população inicial)

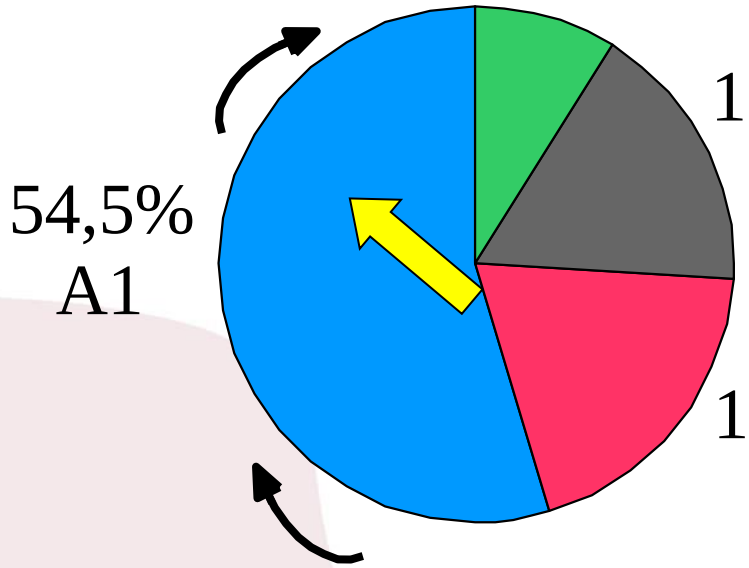
Pop. inicial

cromossomos	x	$f(x)$	Prob. de seleção
$A_1 = 1\ 1\ 0\ 0\ 1$	25	625	54,5%
$A_2 = 0\ 1\ 1\ 1\ 1$	15	225	19,6%
$A_3 = 0\ 1\ 1\ 1\ 0$	14	196	17,1%
$A_4 = 0\ 1\ 0\ 1\ 0$	10	100	8,7%

Probabilidade de seleção proporcional a aptidão

$$p_i = \frac{f(x_i)}{\sum_{k=1}^{N} f(x_k)}$$

Seleção proporcional a aptidão (Roleta)



Pais selecionados

A1 = 1 1 0 0 1

A2 = 0 1 1 1 1

A2 = 0 1 1 1 1

A1 = 1 1 0 0 1

Seleção por Torneio

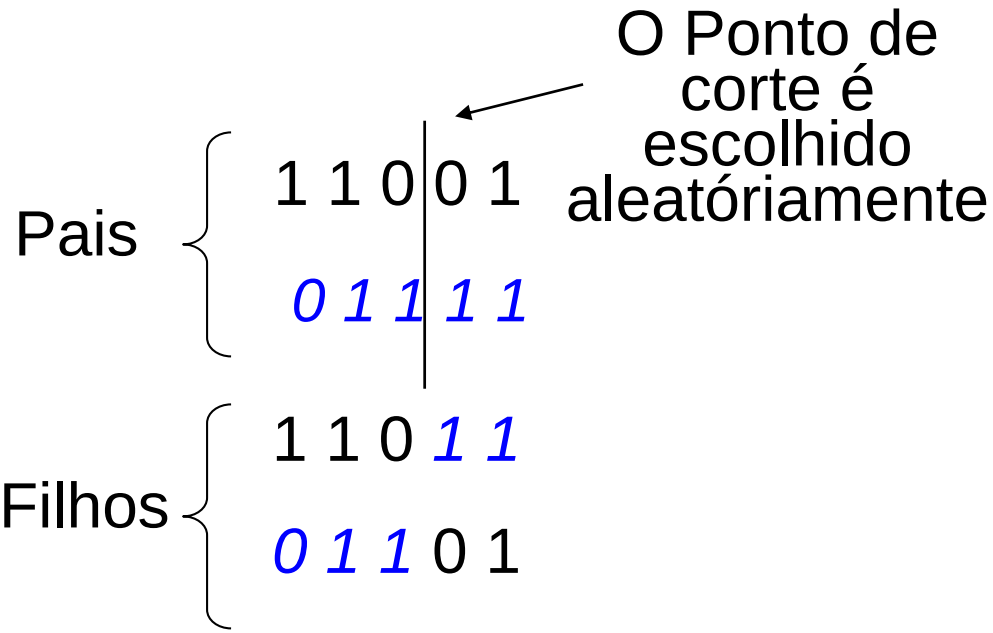
- Escolhe-se *n* (tipicamente 2) indivíduos aleatoriamente da população e o melhor é selecionado.

Crossover e Mutação

- **Combinam** pais selecionados para produção de filhos.
- Principais **mecanismos de busca** do AG.
- Permite **explorar** áreas **desconhecidas** do **espaço de busca**.

Crossover de 1 ponto

O crossover é aplicado com uma dada probabilidade denominada *taxa de crossover* (60% a 90%)



Se o crossover é aplicado os pais trocam suas caldas gerando dois filhos, caso contrário os dois filhos serão cópias exatas dos pais.

Mutação

Mutação inverte os valores dos bits.

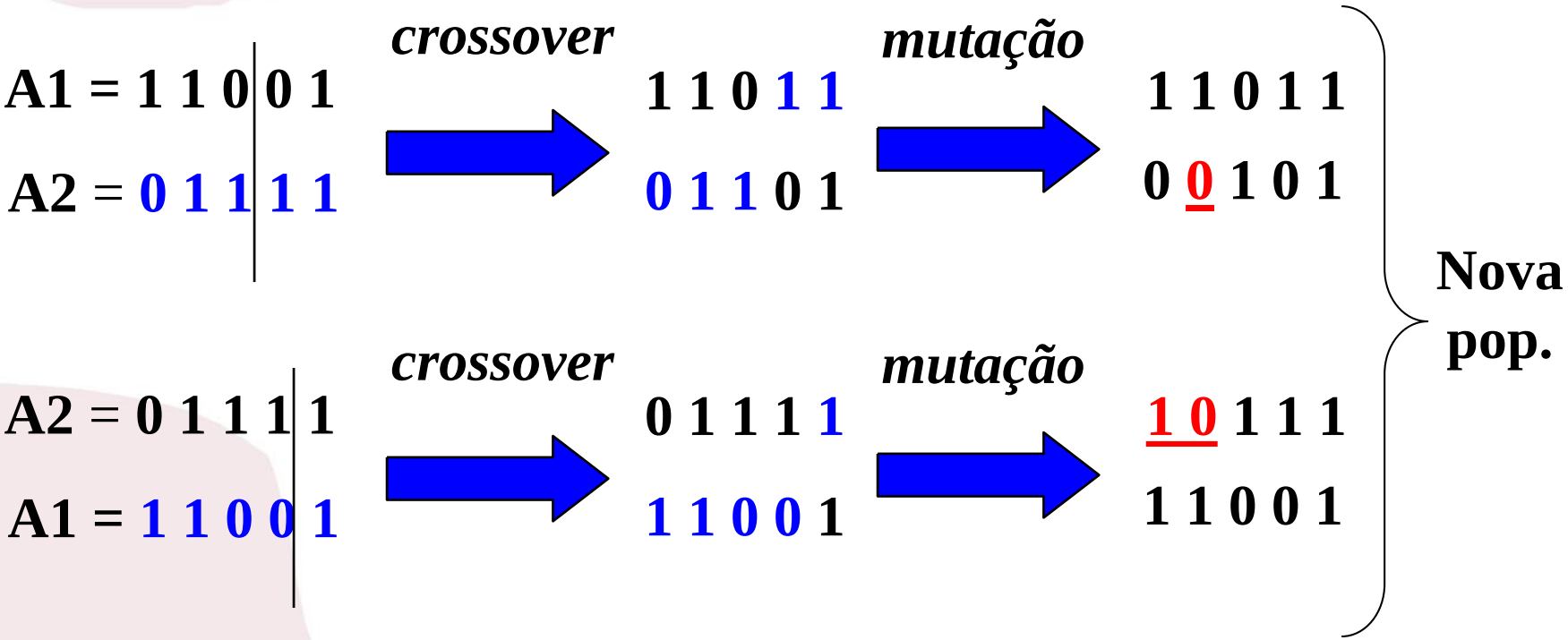
A mutação é aplicada com dada probabilidade, denominada *taxa de mutação* (~1%), em cada um dos bits do cromossomo.

Antes da mutação	0 1 1 0 1
Depois	0 0 1 0 1

Aqui, apenas o 2o.bit passou no teste de probabilidade

A taxa de mutação não deve ser nem alta nem baixa, mas o suficiente para assegurar a diversidade de cromossomos na população.

A primeira geração do Problema 1



A primeira geração do Problema 1 (II)

cromossomos		x	$f(x)$	prob. de seleção
1	1 1 0 1 1	27	729	29,1%
2	1 1 0 0 1	25	625	24,9%
3	1 1 0 0 1	25	625	24,9%
4	1 0 1 1 1	23	529	21,1%

As demais gerações do Problema 1

Segunda Geração

		x	$f(x)$
1	1 1 0 1 1	27	729
2	1 1 0 0 0	24	576
3	1 0 1 1 1	23	529
4	1 0 1 0 1	21	441

Terceira Geração

		x	$f(x)$
1	1 1 0 1 1	27	729
2	1 0 1 1 1	23	529
3	0 1 1 1 1	15	225
4	0 0 1 1 1	7	49

As demais gerações do Problema 1 (II)

Quarta Geração

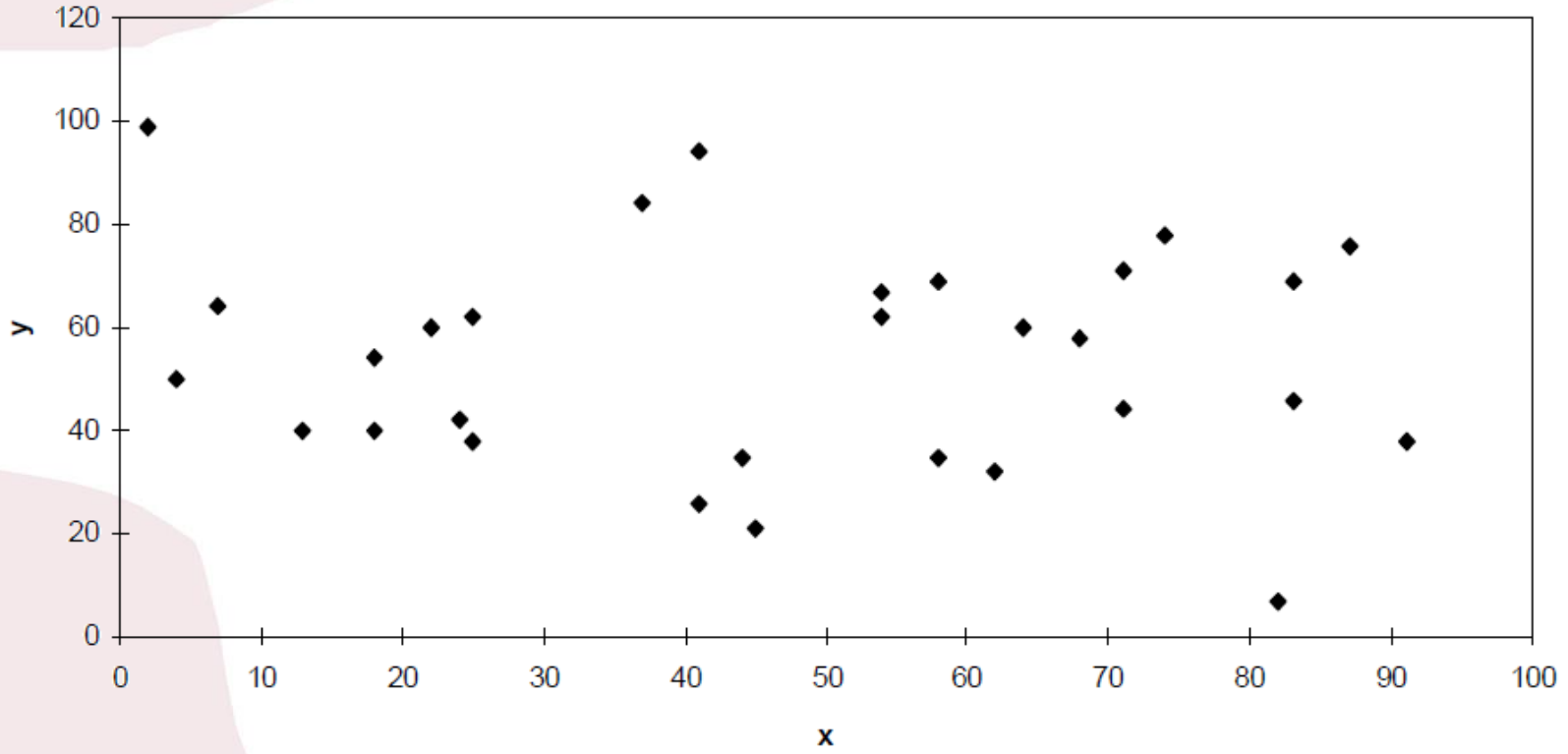
		x	$f(x)$
1	1 1 1 1 1	31	961
2	1 1 0 1 1	27	729
3	1 0 1 1 1	23	529
4	1 0 1 1 1	23	529

Quinta Geração

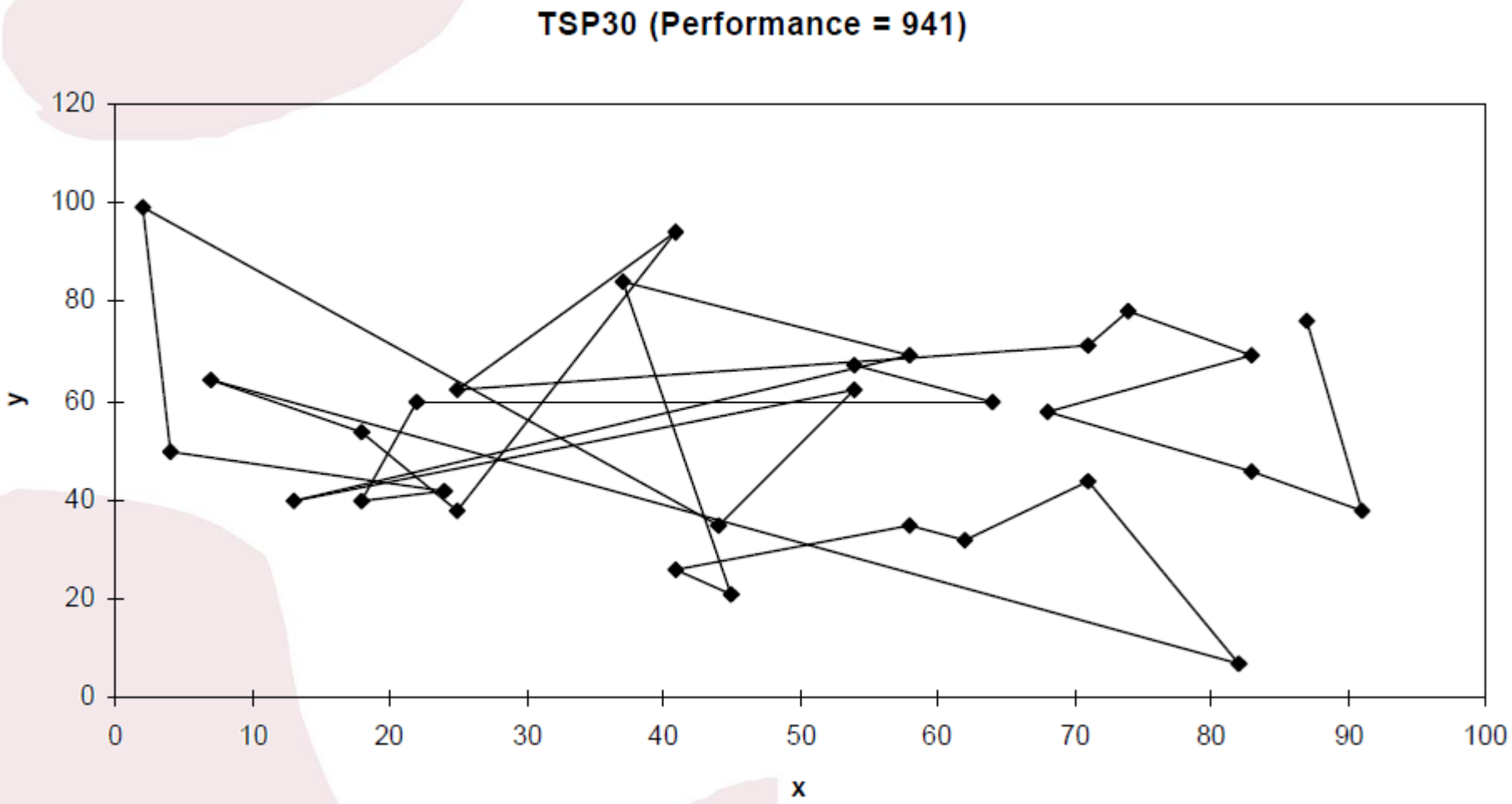
		x	$f(x)$
1	1 1 1 1 1	31	961
2	1 1 1 1 1	31	961
3	1 1 1 1 1	31	961
4	1 0 1 1 1	23	529

Problema 2 – TSP para 30 cidades

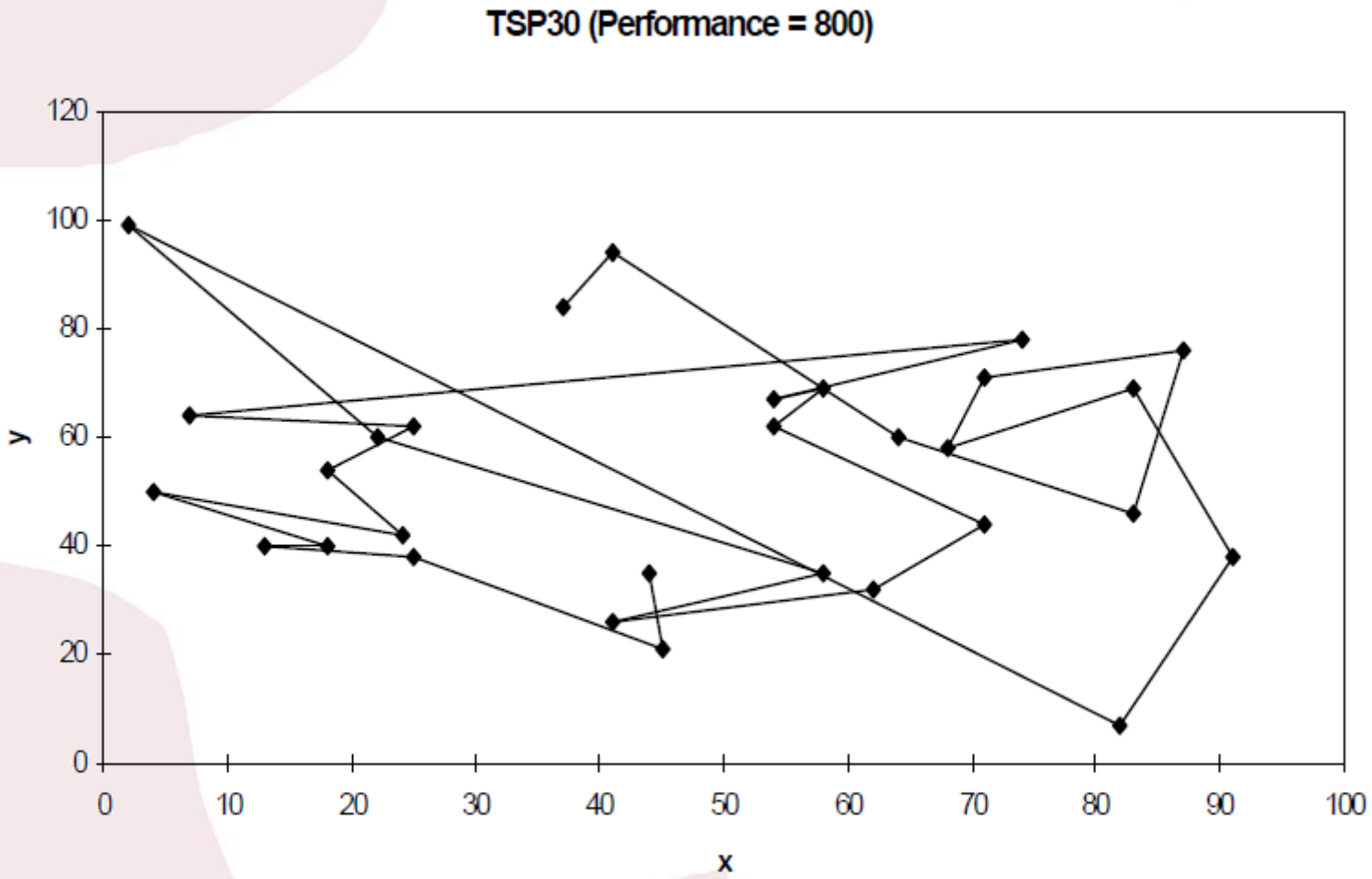
- Distribuição das cidades no mapa



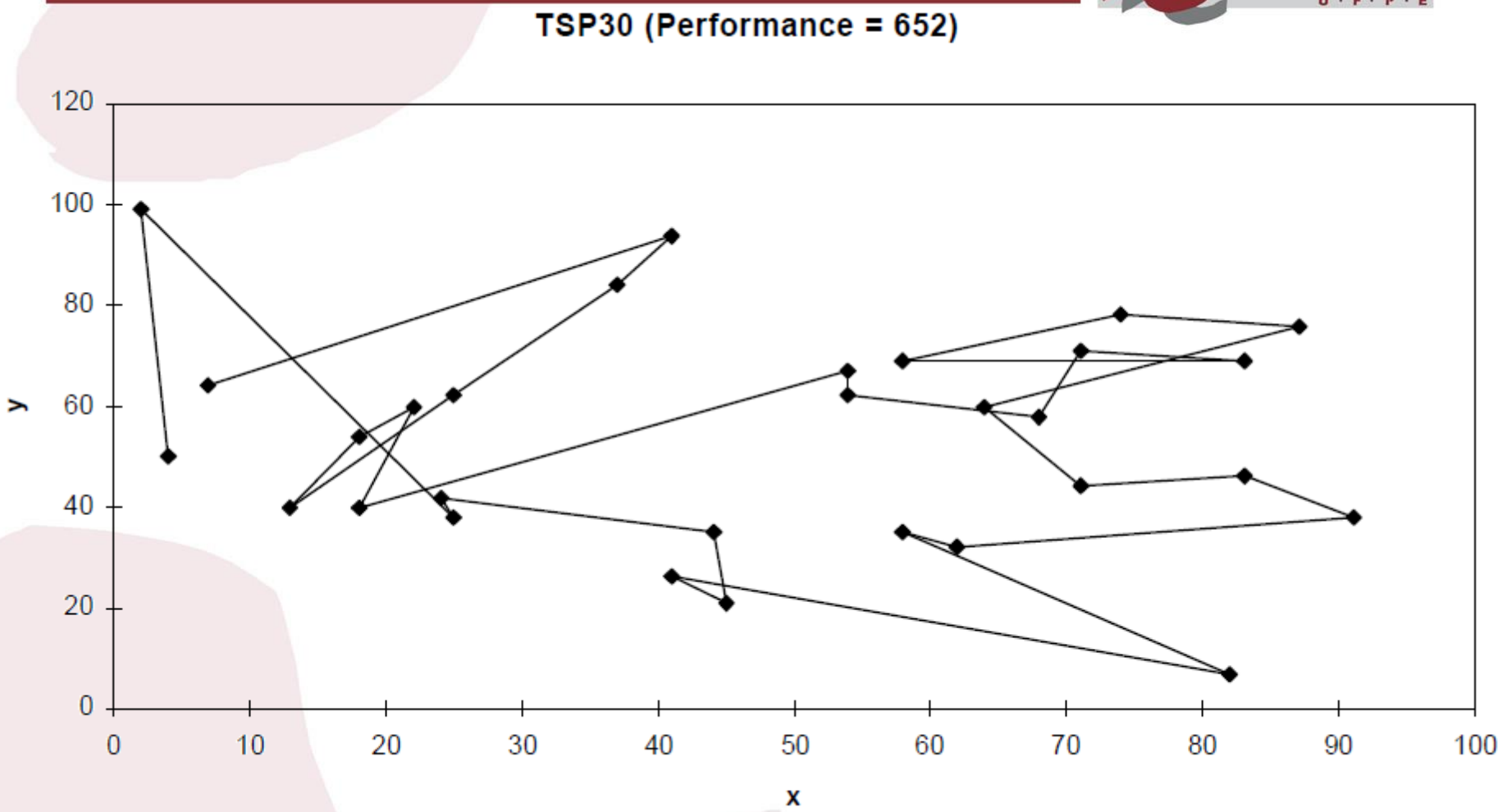
Problema 3 – Primeira solução



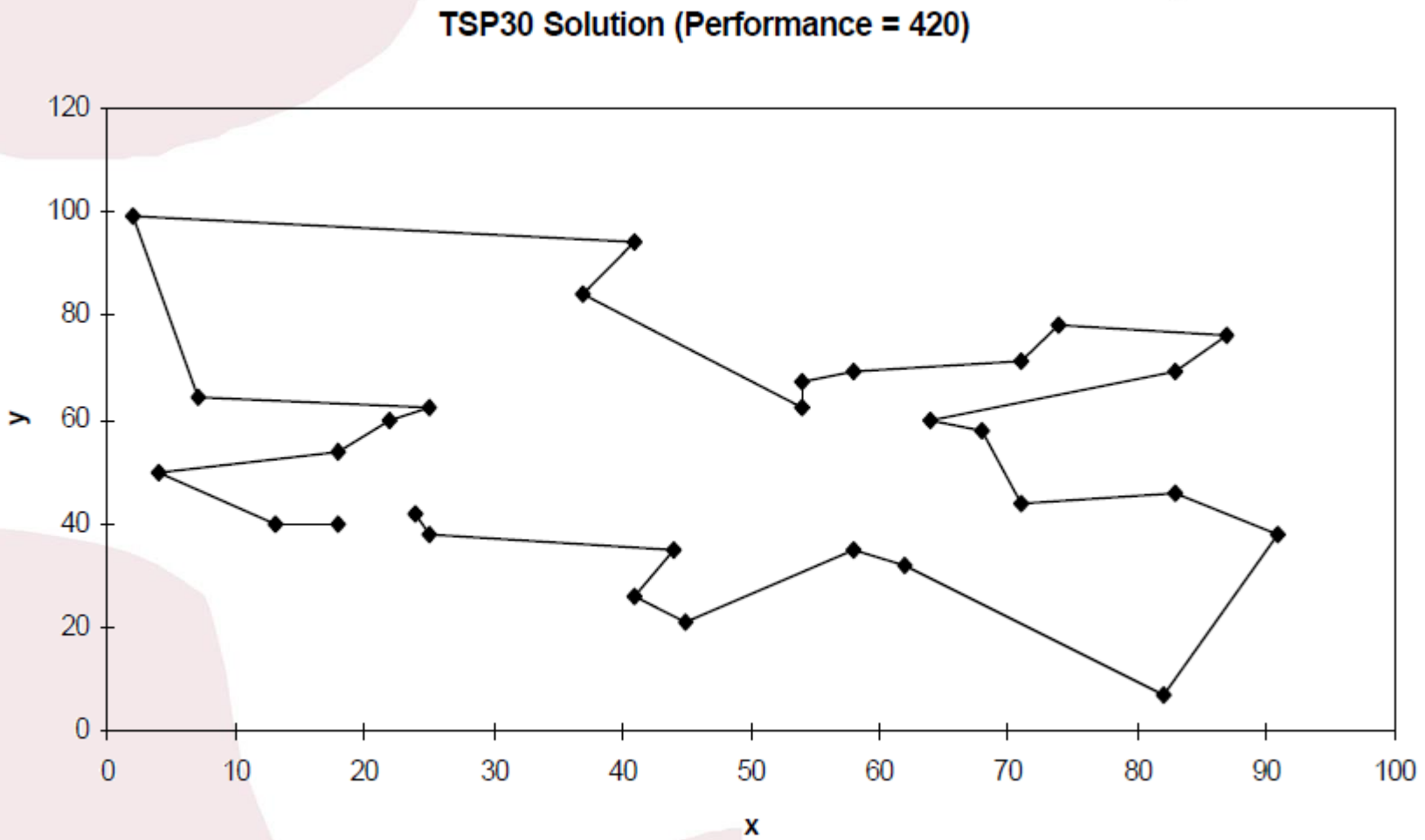
Problema 3 – Segunda solução



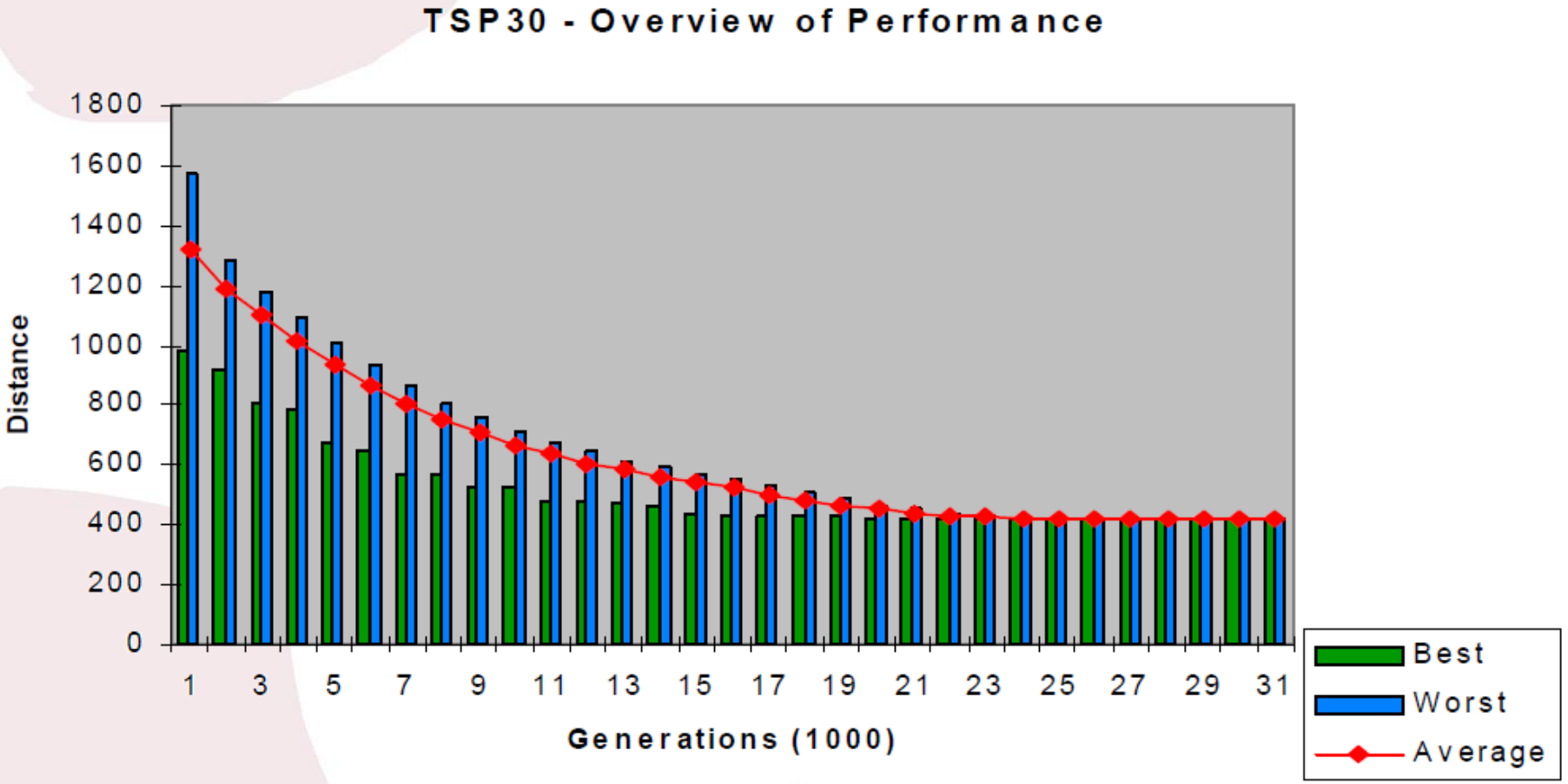
Problema 3 – Terceira solução



Problema 3 – Melhor solução



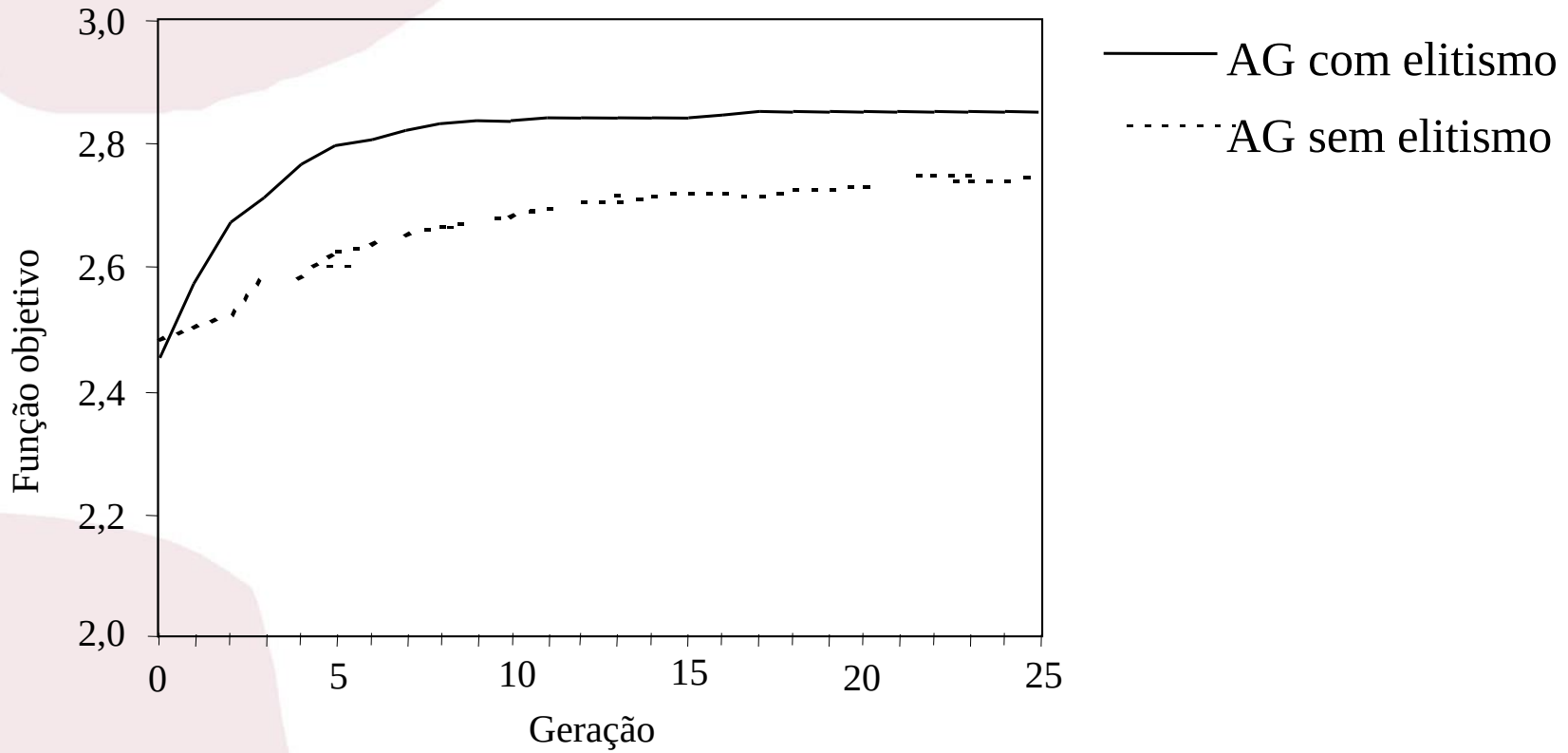
Problema 3 – Progresso da evolução



Elitismo

- O ***crossover*** ou **mutação** podem destruir o melhor indivíduo.
- Por que **perder** a melhor solução encontrada?
- Elitismo **transfere a cópia** do melhor individuo para a **geração seguinte**.

Elitismo no Problema 2



AG com elitismo é melhor ?

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**

Referências

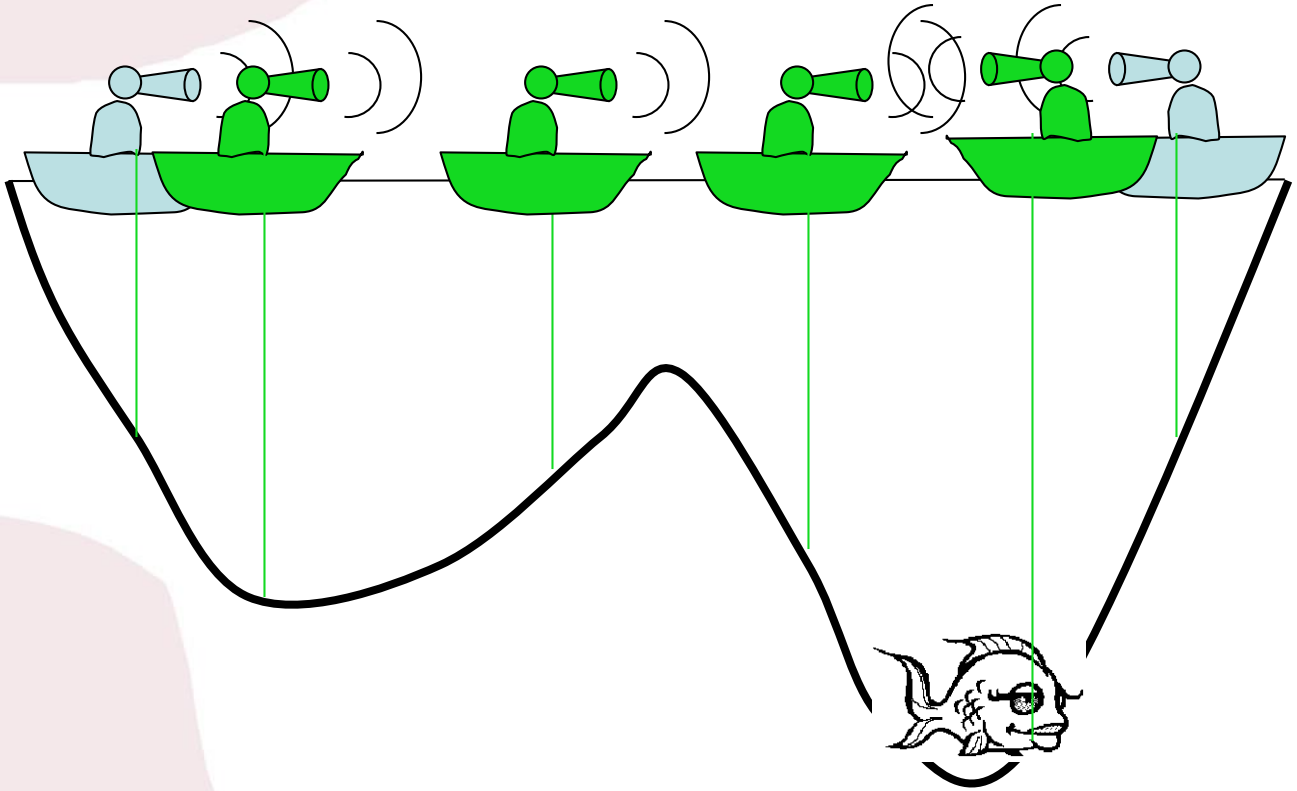
- Henneth A. de Jong, Evolutionary Computation: A Unified Approach, MIT, 2006

Applet:

- http://www.glauserweb.ch/gentore.htm
- https://www.ads.tuwien.ac.at/raidl/tspga/TSPGA.html
- http://math.hws.edu/eck/jsdemo/jsGeneticAlgorithm.html
- http://www.ewh.ieee.org/soc/es/May2001/14/Begin.htm
- http://www.obitko.com/tutorials/genetic-algorithms/portuguese/example-function-minimum.php

Particle Swarm Optimization (Pso)

Exemplo de cooperação



Idéia principal

- Cada partícula busca **um ponto** de ótimo
- Cada partícula está em ***movimento*** e possui uma ***velocidade***
- Cada partícula armazena sua melhor posição encontrada (*best-result-so-far* ou *personal best*)

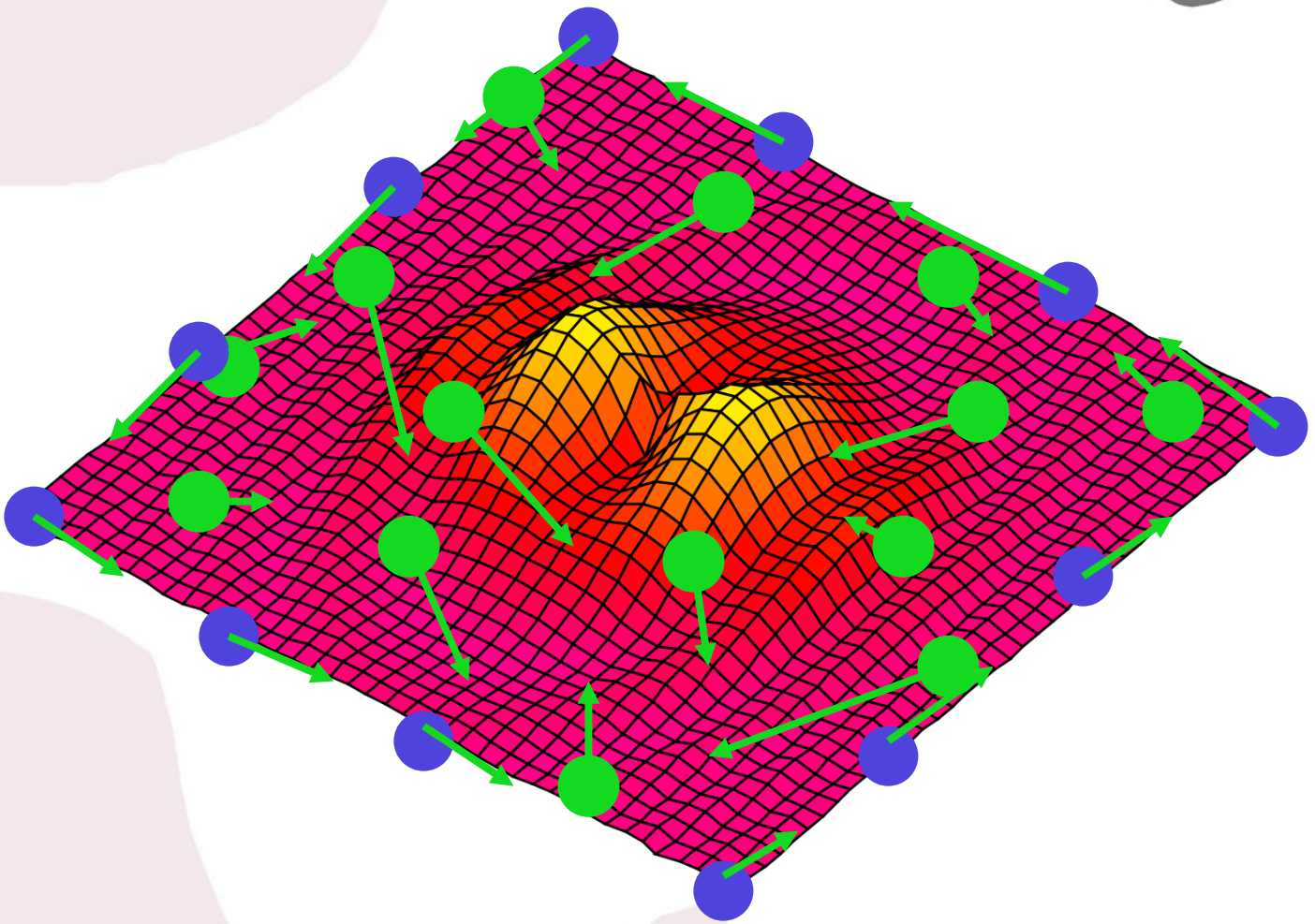
- *Porém isso não é suficiente, as partículas precisam de ajuda para identificar os melhores locais de busca*

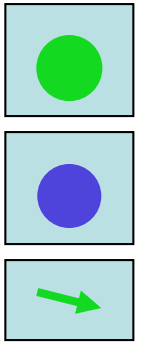
Idéia principal II

- As partículas no enxame **cooperam** entre si. Elas **trocam informações** sobre o que descobrirar nos locais que já visitaram

- A cooperação é muito simples:
  - Uma partícula **possui** uma **vizinhança associada**
  - Uma partícula **conhece** o desempenho (*fitnesses*) das partículas na **sua vizinhança** e **usa** a posição da partícula com **melhor desempenho**
  - Esta posição é utilizada para ajustar a **velocidade da partícula**

Inicialização. Posição e velocidade

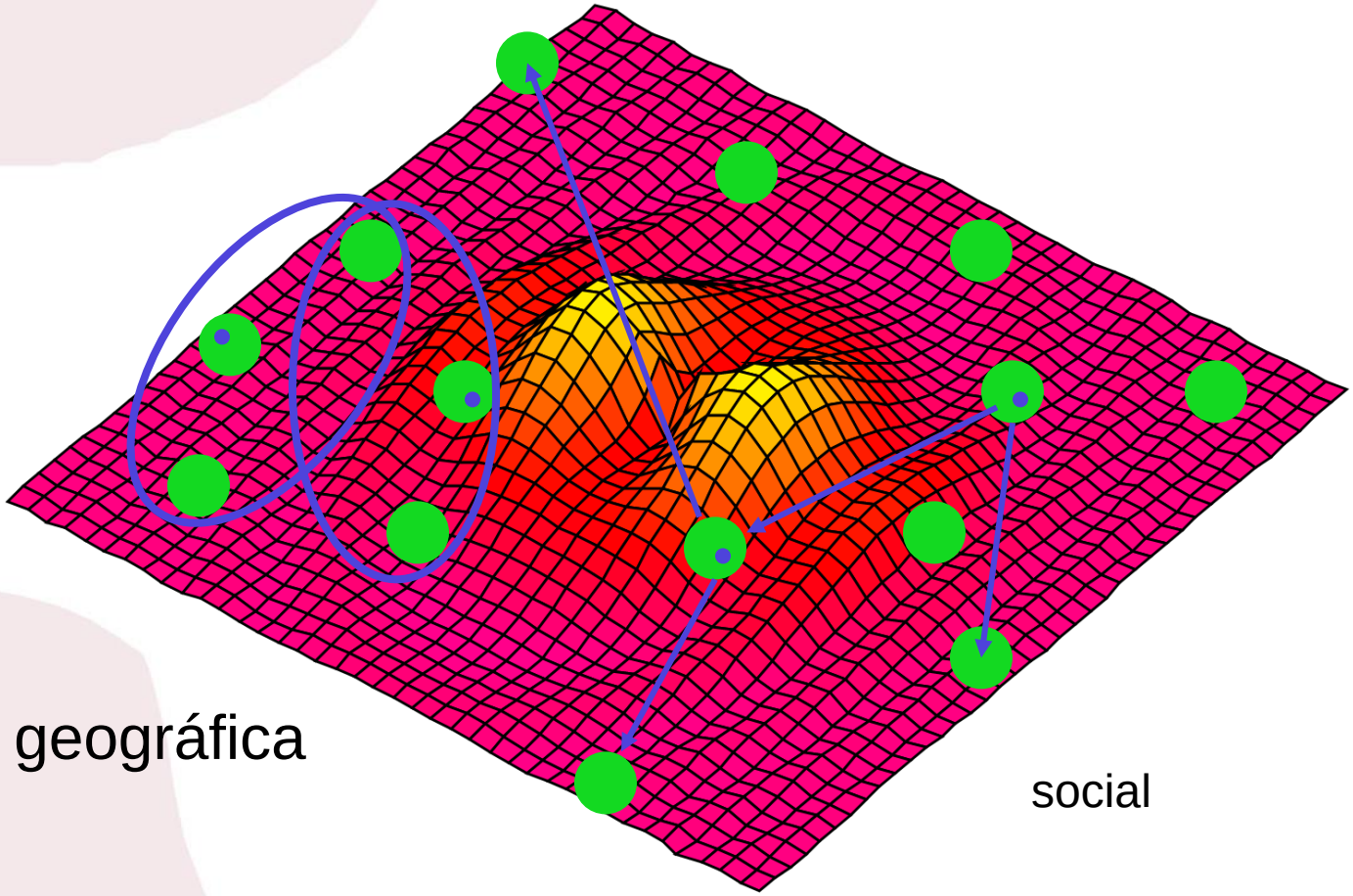




O que uma partícula faz

- Em cada passo (*timestep*), a partícula se move para uma **nova posição**. Ela faz isso ajustando sua ***velocidade*** dado por:
  - *Velocidade atual* +
  - *Uma porção ponderada na direção de sua melhor posição* +
  - *Uma porção ponderada na direção da melhor posição da vizinhança*
- *Após definida a nova velocidade, a nova posição é definida pela posição atual + a nova velocidade*

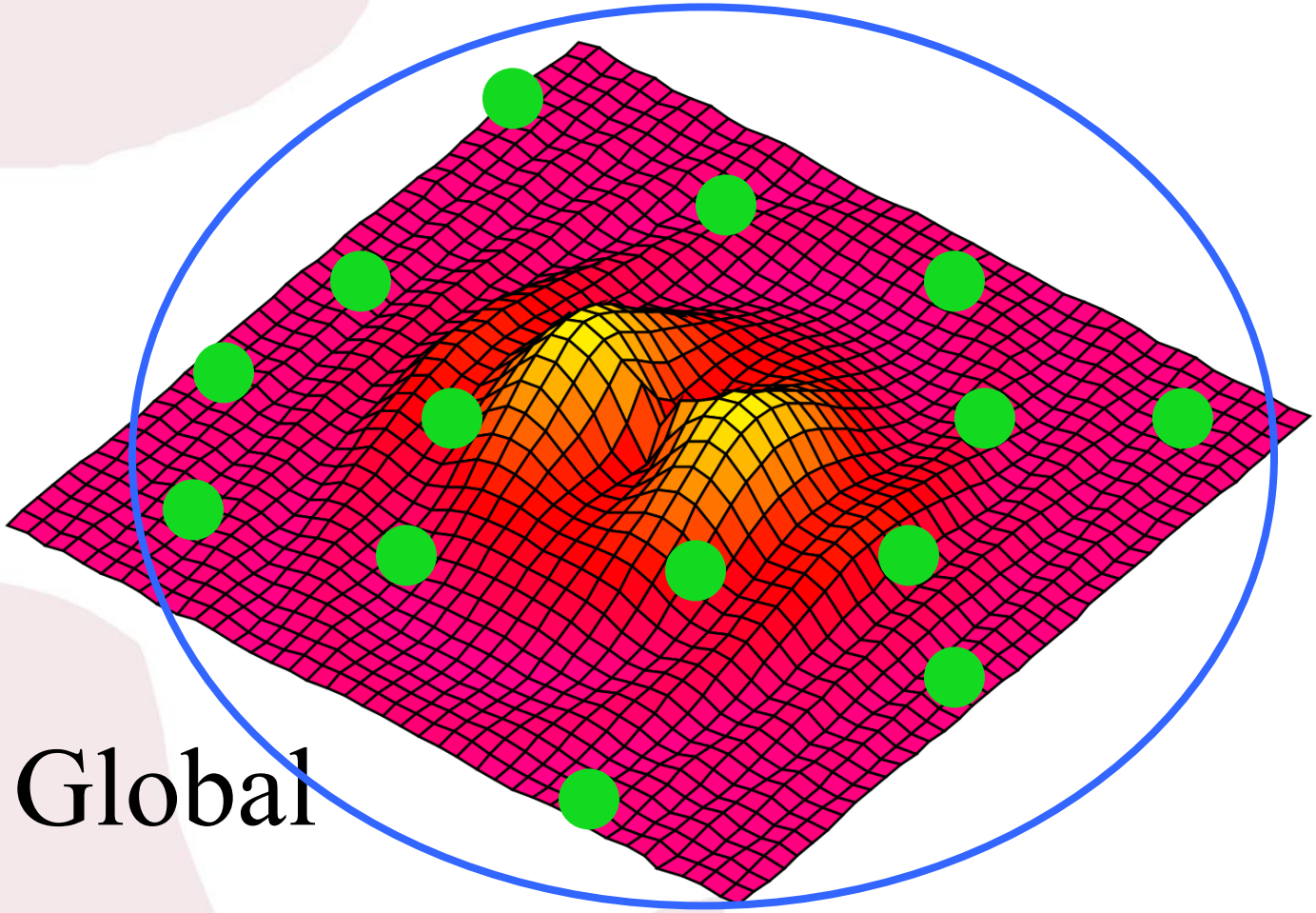
Vizinhança



geográfica

social

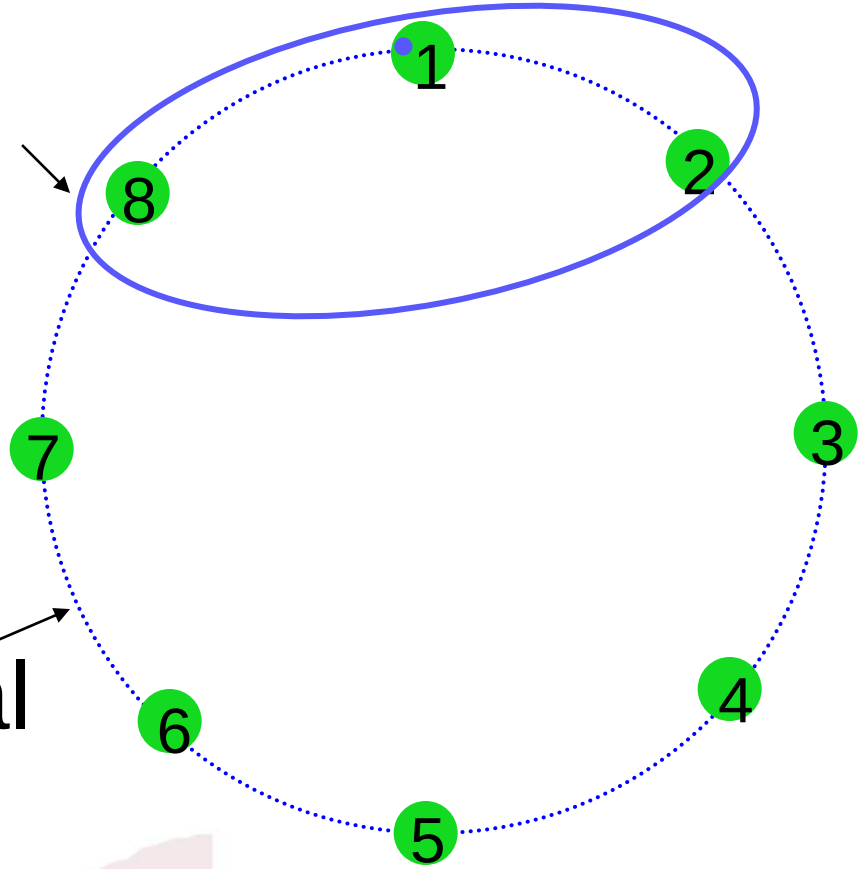
Vizinhança



Global

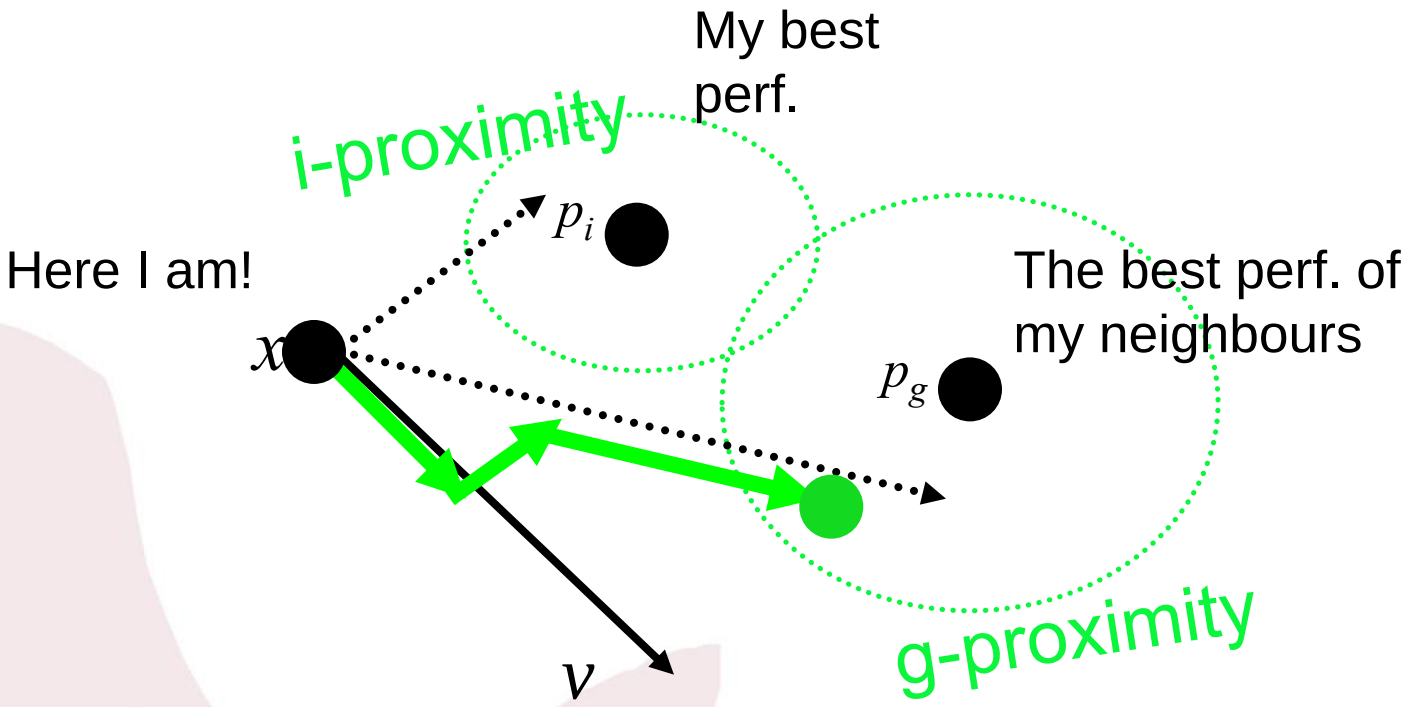
Vizinhança circular

3-vizinhança da partícula 1



Círculo virtual

As partículas ajustam sua posição de acordo com um compromisso psicossocial entre o conforto de um indivíduo e o que a sociedade reconhece



Equation (a)

```
v[] = c0 *v[]
    + c1 * rand() * (pbest[] - present[])
    + c2 * rand() * (gbest[] - present[])
```

(in the original method, c0=1, but many researchers now play with this parameter)

Equation (b)

```
present[] = present[] + v[]
```

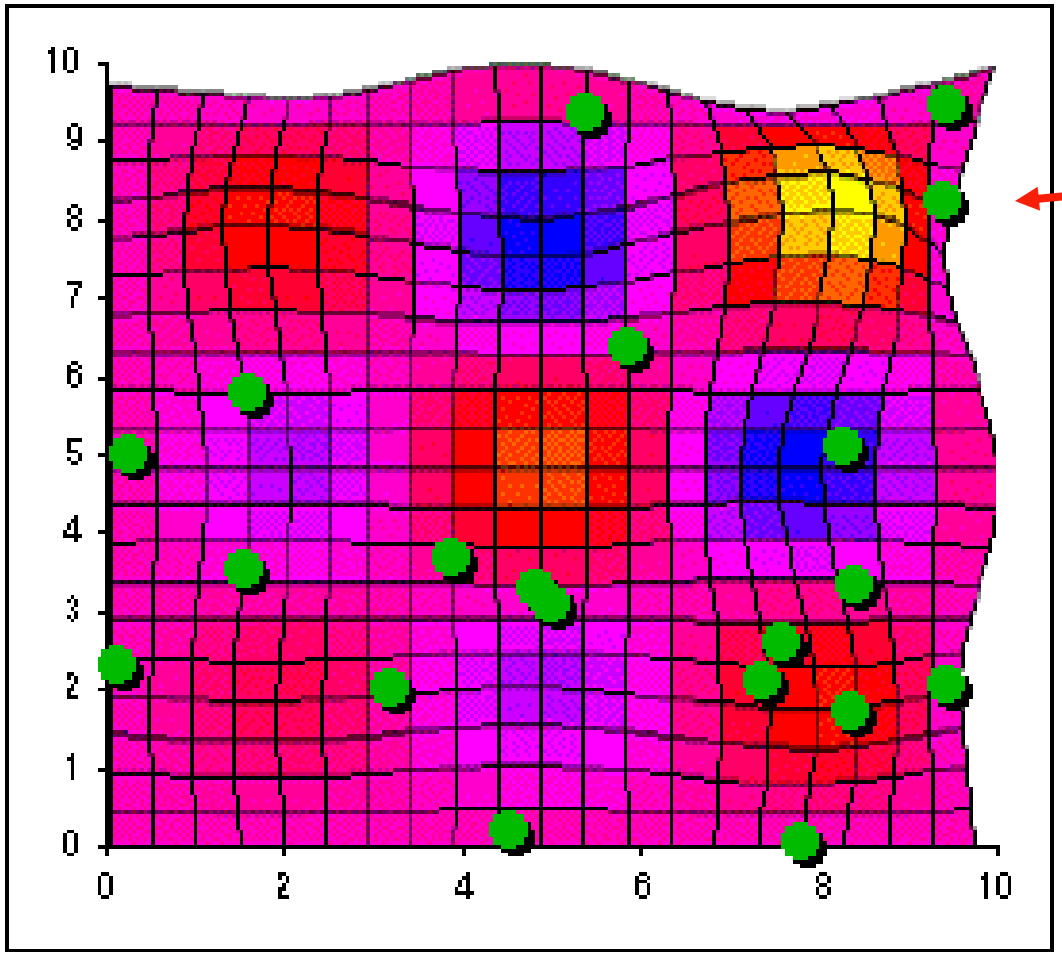
Pseudocode
http://www.swarmintelligence.org/tutorials.php

```
For each particle
    Initialize particle
END

Do
    For each particle
        Calculate fitness value
        If the fitness value is better than its peronal best
        Set current value as the new pBest
    End

    Choose the particle with the best fitness value of all as gBest
    For each particle
        Calculate particle velocity according equation (a)
        Update particle position according equation (b)
    End
While maximum iterations or minimum error criteria is not attained
```

As velocidades das partículas em cada dimensão estão vinculadas a uma velocidade máxima **Vmax**.

Se a soma das acelerações ultrapassar o valor de Vmax (definido pelo projetista), esta velocidade é limitada a Vmax.

Ilustração



Ótimo global

Parametros

- Número de partículas
- C0 (importância da *posição atual*)
- C1 (importância do *personal best*)
- C2 (importância do *neighbourhood best*)

Parâmetros

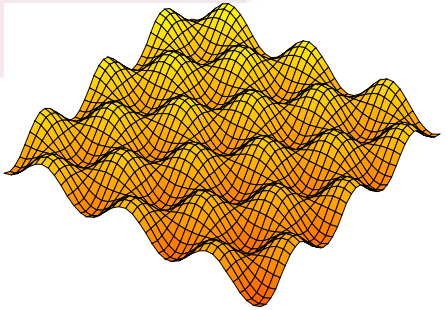
- Número de partículas (10—50) normalmente são suficientes

- Normalmente C1 + C2 = 4. Sem outra razão senão testes empíricos ☺

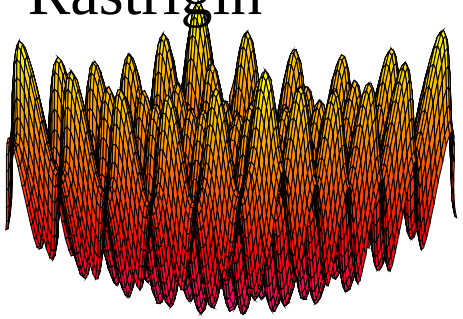
- Vmax
  - muito baixo, muito lento
  - muito alto, muito instável.

Exemplos de aplicação

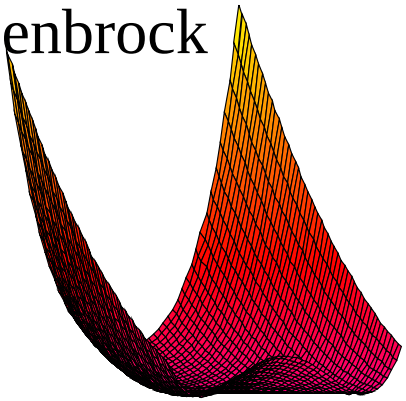
Griewank



Rastrigin



Rosenbrock



... resultados

Ótimo=0, dimensão=30, melhor resultado após 40.000 iterações

30D function	PSO Type 1"	Evolutionary algo. (Angeline 98)
Griewank [±300]	0.003944	0.4033
Rastrigin [±5]	82.95618	46.4689
Rosenbrock [±10]	50.193877	1610.359

- Applet PSO

http://natcomp.liacs.nl/NC/applets/SwarmApplet/SwarmApplet.html

http://www.projectcomputing.com/resources/psovis/

http://web.ist.utl.pt/gdgp/VA/pso.htm

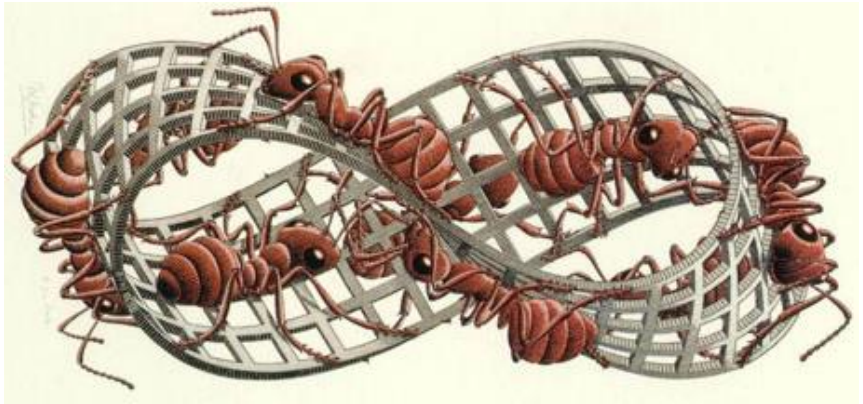
https://web.eecs.utk.edu/~mclennan/Classes/420-594-F07/NetLogo/PSO.html

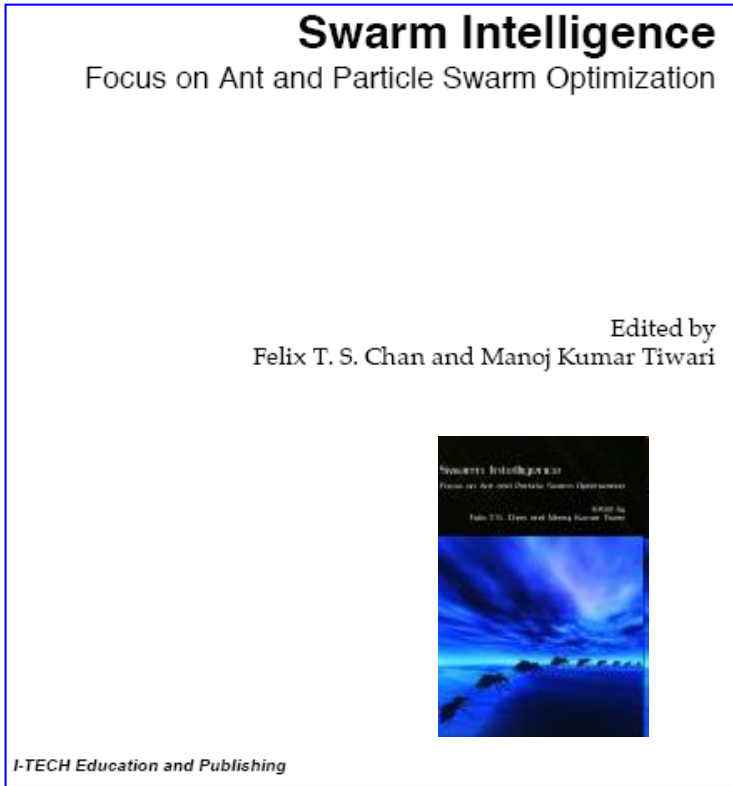
PSO - Referências

- J. Kennedy and R. Eberhart. *Swarm Intelligence*. Morgan Kaufmann Publishers, Inc, San Francisco, CA, 2001
- F. van den Bergh. *"An Analysis of Particle Swarm Optimizers"*. Phd dissertation, Faculty of Natural and Agricultural Sciences, Univ. Pretoria, Pretoria, South Africa, 2002.
- J. Kennedy and R. Eberhart. Particle Swarm Optimization, in: *Proc. IEEE Int'l. Conf. on Neural Networks (Perth, Australia)*, IEEE Service Center,Piscataway, NJ, IV:1942-1948.
- C.A. Coello Coello, G. Toscano, M.S. Lechuga. Handling Multiple Objectives with Particle Swarm Optimization. *IEEE Transactions on Evol. Computation*, Vol. 8, No. 3, June 2004.
- M. Settles, T. Soule. Breeding Swarms: A GA/PSO Hybrid. *Proc. of GECCO'05*, Washington DC, USA, 2005.

Otimização por Colônia de Formigas (*ACO*)

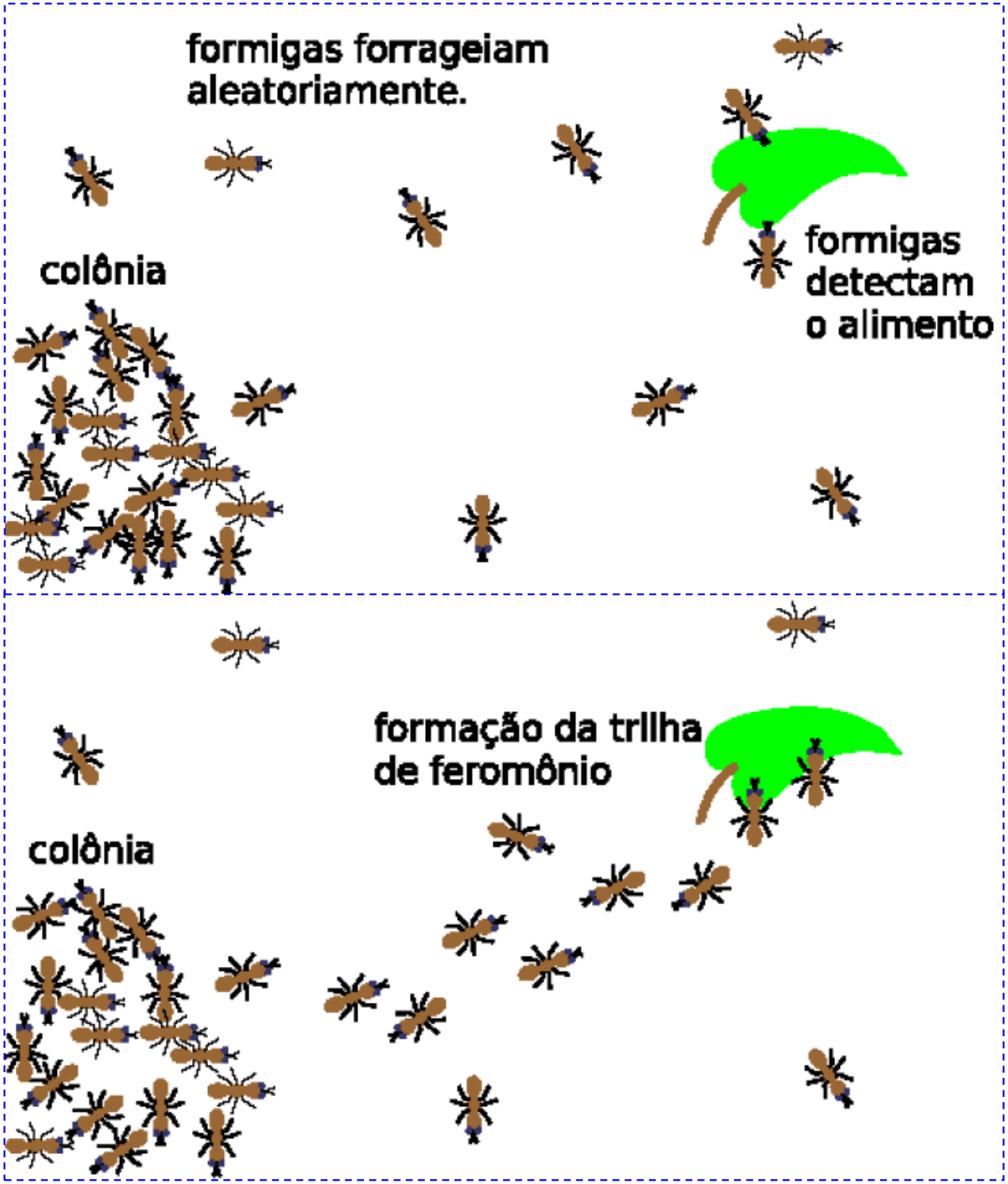
- Inspiração Biológica
- Proposto por Dorigo e Gambardella em 1997
- *ACO* (*Ant Colony Optimization*)
- Principal aplicação no PCV
- Programação do algoritmo

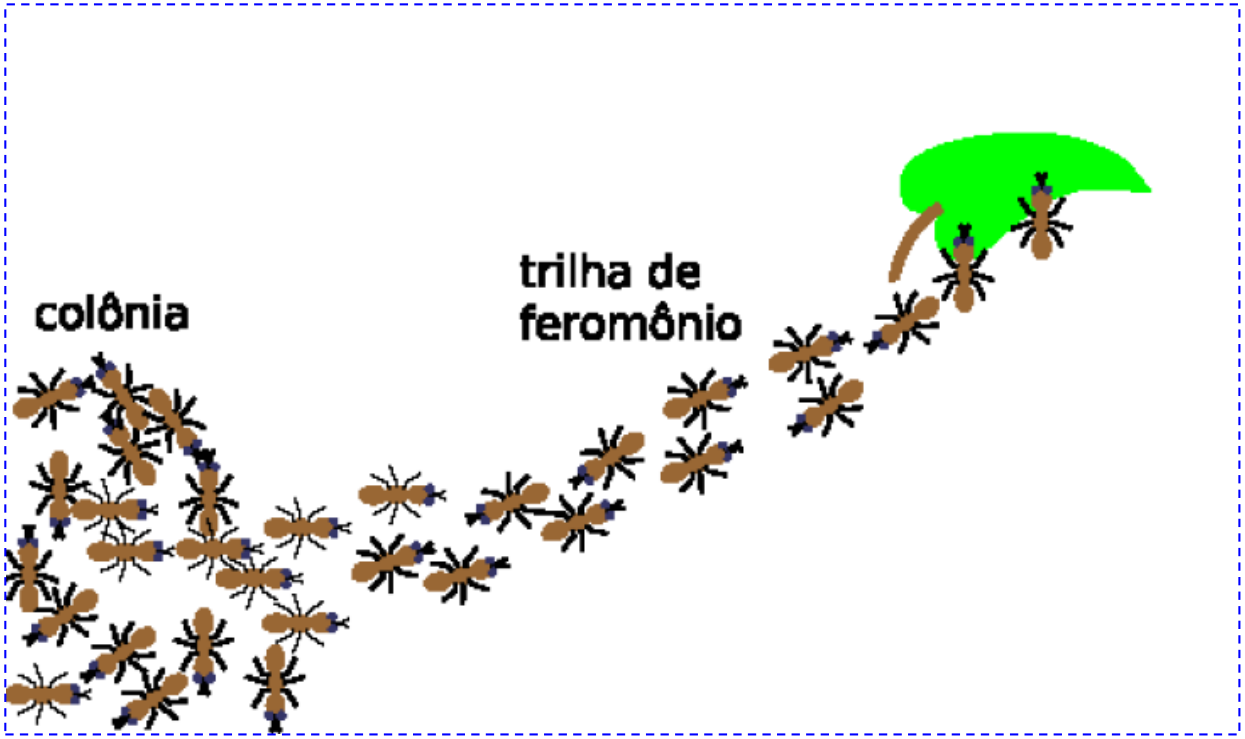




Inspiração Biológica

- Metaheurística baseada em uma população de formigas
- Relação com o comportamento na busca de alimento ou deslocamento.
- Muitas espécies de formigas são quase cegas
- A comunicação é através de feromônios (usado para criar caminhos – trilhas de formigas)

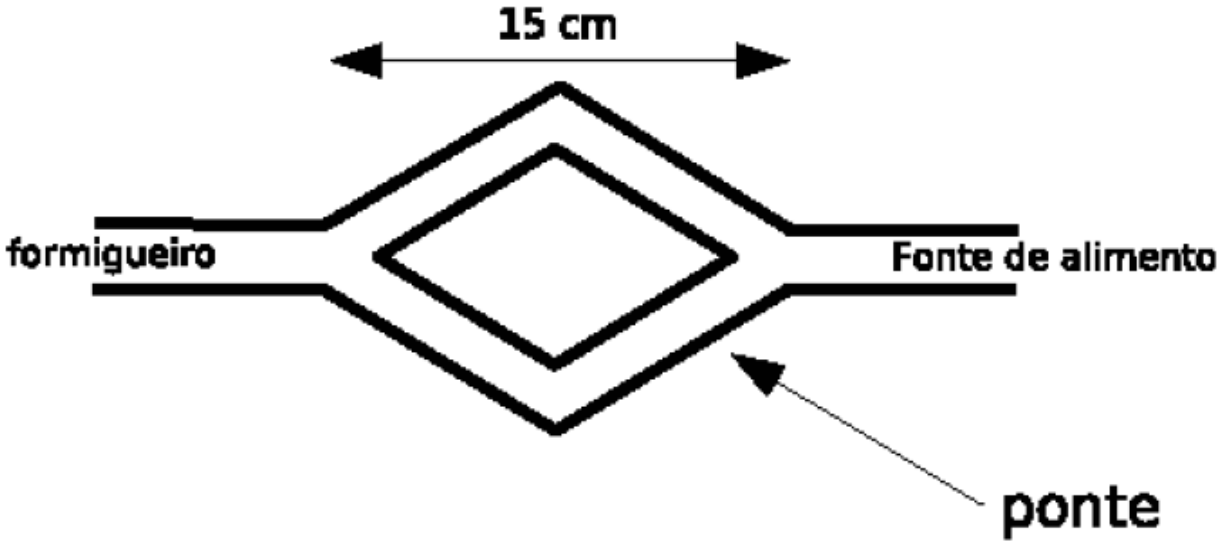

formigas forrageiam aleatoriamente.
colônia
formigas detectam o alimento
formação da trilha de feromônio
colônia



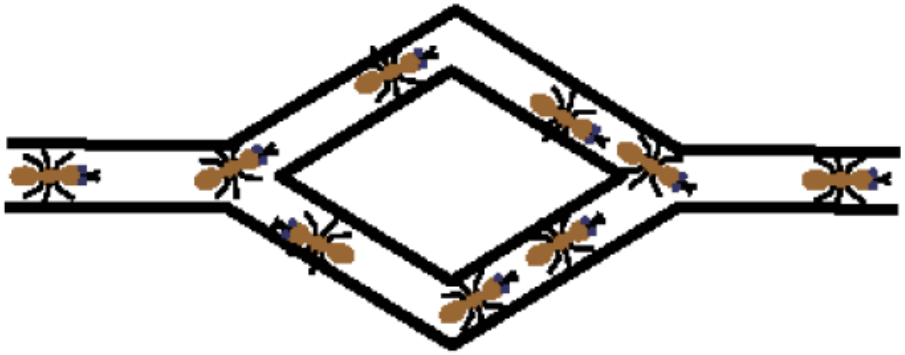
- Ao caminhar, as formigas depositam no chão o feromônio, formando, uma trilha
- Através do olfato, as formigas escolhem, conforme a probabilidade, o caminho com maior feromônio
- Esta trilha auxilia a formiga a encontrar o alimento e a volta ao formigueiro, além de ajudar as outras formigas a encontrar o alimento

O experimento da ponte binária

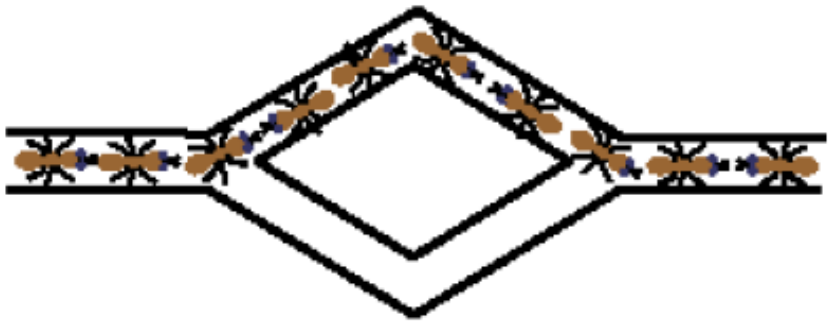
Experimento realizado por Denebourg et al., 1990, para estudar o comportamento forrageiro das formigas



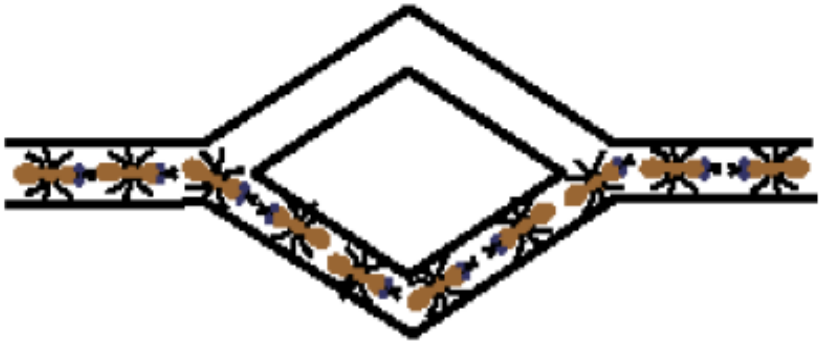
- No início, as formigas são deixadas livres para escolher o caminho. Não há feromônio ainda



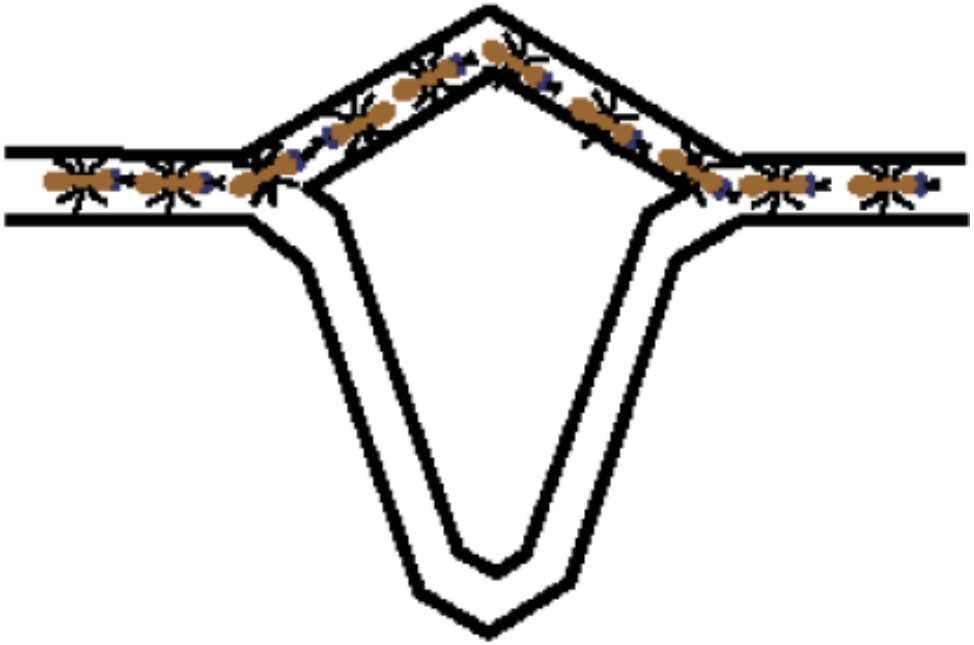
- As formigas convergem para um dos caminhos com igual probabilidade
- Devido a flutuações, uma das pontes terá mais feromônio e atrairá as formigas com maior probabilidade

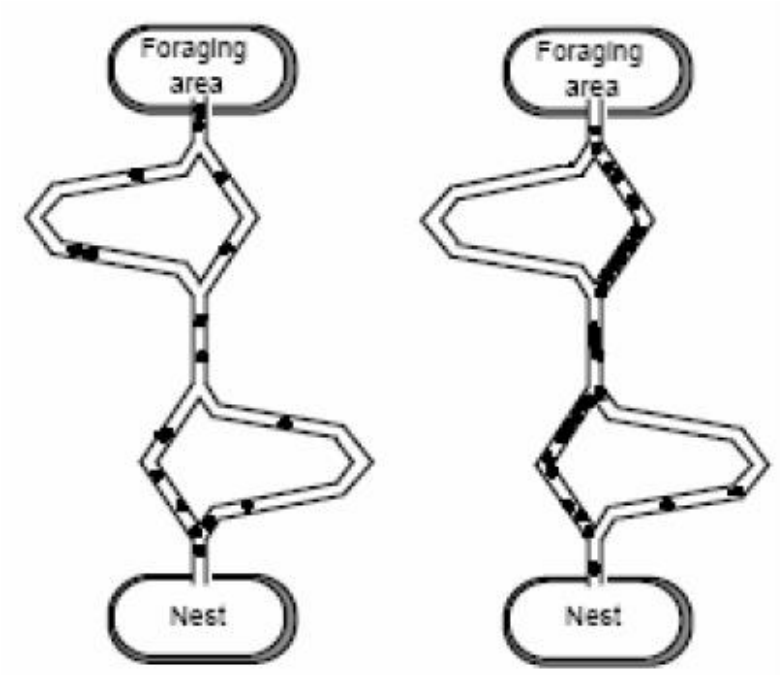


ou



- Usando pontes de tamanhos diferentes, as formigas convergem para a ponte mais curta
- A ponte curta é percorrida em menos tempo, fazendo com que mais formigas a atravessem. Logo, mais feromônio é depositado
- As formigas escolhem, com maior probabilidade a ponte mais curta (com mais feromônio)





Método da Colônia de Formigas

- Formigas artificiais são heurísticas construtivas
- Soluções contruídas de forma probabilística utilizando duas informações
- A trilha de feromônios (artificial) – muda dinâmicamente durante a execução do programa
- A informação heurística específica do problema a ser resolvido

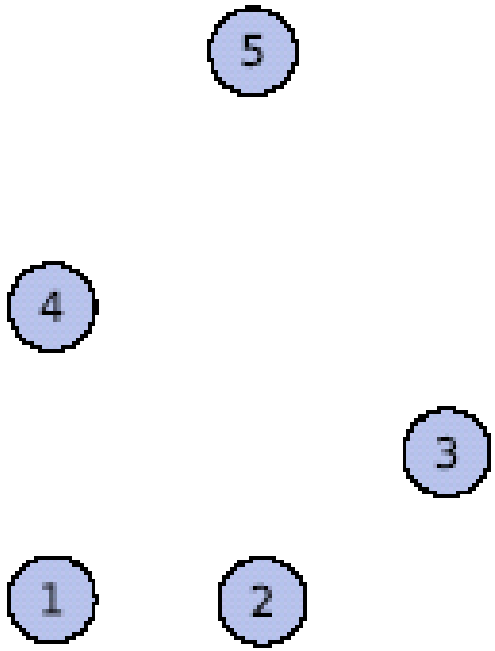
Ant System

- Proposto por Marco Dorigo e colaboradores (DORIGO et al., 1991)
- O Ant System é o primeiro algoritmo que surgiu inspirado em colônia de formigas.
- Peculiaridades do ambiente das formigas utilizadas:
  - Ao **tomar um caminho** a formiga **deixa** no mesmo uma certa **quantidade de feromônio**;
  - Uma formiga **escolhe** determinado caminho de acordo com uma **função probabilística** envolvendo a **distância** deste caminho e a **quantidade de feromônio** presente neste;
  - As formigas **lembram** os pontos por **onde já passaram** e **não retornam** a estes pontos até que tenham chegado à fonte de alimento.

ACO aplicado ao PCV

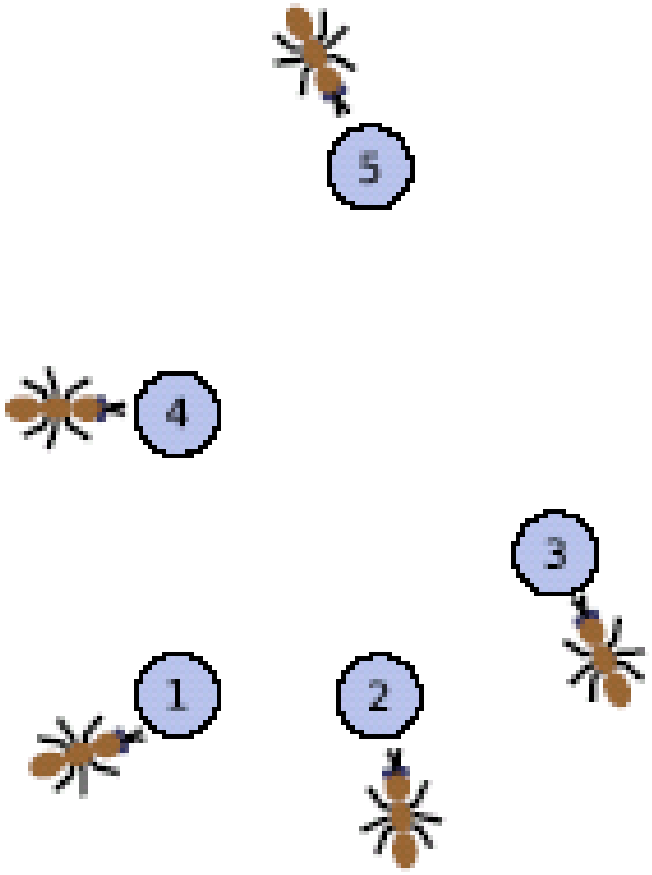
Matriz de distâncias

dij	1	2	3	4	5
1	0,0	1,0	2,2	2,0	4,1
2	1,0	0,0	1,4	2,2	4,0
3	2,2	1,4	0,0	2,2	3,2
4	2,0	2,2	2,2	0,0	2,2
5	4,1	4,0	3,2	2,2	0,0

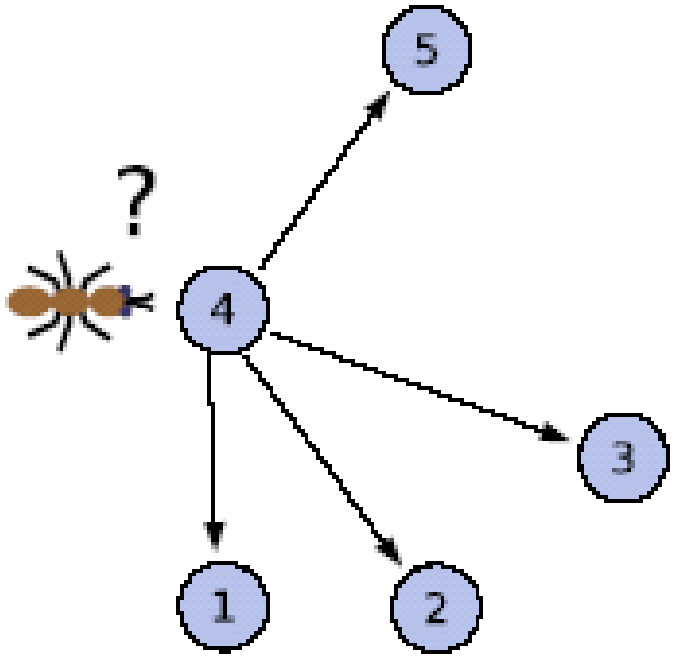


Cidades do PCV

- Cada formiga irá construir uma solução movendo-se de uma cidade para outra. No início, cada formiga é colocada em uma cidade diferente (ou colocada aleatoriamente)



- Começando de uma cidade *i*, a formiga move-se escolhendo probabilisticamente a cidade vizinha *j* (entre os vizinhos factíveis)



Probabilidade de Transição

- A probabilidade da formiga k que está na cidade i de escolher a cidade j é dada pela regra

$$p_{ij}^{k} = \frac{\tau_{ij}^{\alpha} \eta_{ij}^{\beta}}{\sum_{l \in N_j^k} \tau_{il}^{\alpha} \eta_{jl}^{\beta}}, \ \textit{quando} \ \ j \in N_i^k$$

em que:

- τ_{ij} é o **feromônio** associado à aresta (i, j)
- α e β são parâmetros para determinar a influência do **feromônio** e da **informação heurística**
- N_j^k é a vizinhança **factível** da formiga k (isto é, o conjunto de cidades ainda não visitadas pela formiga k).

Informação heurística do PCV

- Associada a aresta (i, j) existe um valor heurístico η_{ij} dado por

$$\eta_{ij} = 1/d_{ij}$$

  que representa a **atratividade** da formiga visitar a cidade i depois de visitar a cidade j

- O valor η_{ij} é **inversamente proporcional** a distância d_{ij} entre as cidades i e j
- A partir de uma cidade i, a escolha da cidade candidata j é feita de acordo com a probabilidade de transição, com idéia similar à escolha por roleta de algoritmos genéticos

Exemplo

Considere o PCV dado abaixo pela matriz de distâncias, $\alpha = \beta = \rho = 0{,}5$, e a matriz de feromônios iniciais:

dij	1	2	3	4	5
1	0,0	1,0	2,2	2,0	4,1
2	1,0	0,0	1,4	2,2	4,0
3	2,2	1,4	0,0	2,2	3,2
4	2,0	2,2	2,2	0,0	2,2
5	4,1	4,0	3,2	2,2	0,0

τij	1	2	3	4	5
1	∞	0,30	0,25	0,20	0,30
2	0,30	∞	0,20	0,20	0,30
3	0,25	0,20	∞	0,10	0,15
4	0,20	0,20	0,10	∞	0,45
5	0,30	0,30	0,15	0,45	∞

Encontre soluções para o PCV considerando a matriz de probabilidades.

$$p_{ij}^{k} = \frac{\tau_{ij}^{\alpha} \eta_{ij}^{\beta}}{\sum_{l \in N_j^k} \tau_{il}^{\alpha} \eta_{jl}^{\beta}}, \quad quando \quad j \in N_i^k$$

aco

Passo 1

formiga	Candidatos / prob. de transição	solução parcial
1	2(45%), 3(21%), 4(23%), 5(11%)	1-2
2	1(41%), 3(30%), 4(19%), 5(10%)	2-1
3	1(23%), 2(37%), 4(23%), 5(16%)	3-4
4	1(27%), 2(24%), 3(24%), 5(24%)	4-5
5	1(19%), 2(20%), 3(25%), 4(36%)	5-2

A escolha do candidato é de acordo com a probabilidade de transição. É feita de forma similar ao algoritmo da roleta dos algoritmos genéticos.

Passo 2

formiga	Candidatos / prob. de transição	solução parcial
1	3(50%), 4(32%), 5(18%)	1-2-3
2	3(38%), 4(42%), 5(20%)	2-1-4
3	1(35%), 2(32%), 5(32%)	3-4-5
4	1(30%), 2(31%), 3(39%)	4-5-2
5	1(46%), 3(33%), 4(21%)	5-2-1

Passo 3

formiga	Candidatos / prob. de transição	solução parcial
1	4(59%), 5(41%)	1-2-3-5
2	3(50%), 5(50%)	2-1-4-5
3	1(49%), 2(51%)	3-4-5-1
4	1(58%), 3(42%)	4-5-2-1
5	3(48%), 4(52%)	5-2-1-4

Passo 4

formiga	Candidatos / prob. de transição	solução parcial
1	4(100%)	1-2-3-5-4
2	3(100%)	2-1-4-5-3
3	2(100%)	3-4-5-1-2
4	3(100%)	4-5-2-1-3
5	3(100%)	5-2-1-4-3

Término da Primeira Iteração

formiga (k)	solução completa	comprimento da viajem (L_k)
1	1-2-3-5-4-1	9,8
2	2-1-4-5-3-2	9,8
3	3-4-5-1-2-3	10,9
4	4-5-2-1-3-4	11,6
5	5-2-1-4-3-5	12,4

Algoritmo

Coloque cada formiga em uma cidade aleatória
Para t = 1 até o número de iterações
    Para k = 1 até m (nº de formigas)
        Enquanto a formiga k não construir a viagem S_k
            Selecione a próxima cidade pela regra da probabilidade
        Fim
        Calcule a distância L_k da viagem S_k
        Se $L_k < L^*$ então
            $S^* = S_k$, $L^* = L_k$
        Fim
    Fim
    Atualize os feromônios
Fim
Retornar S^*

Atualização do Feromônio

No feromônio τ_{ij} associado a aresta (i, j) ocorrem dois eventos:

1. Evaporação
   - ❑ evita que o feromônio acumulado cresça indefinidamente
   - ❑ permite esquecer pobres decisões do passado de busca
   - ❑ permite soluções diferentes
2. Depósito de feromônio de todas as formigas que passaram sobre (i, j)

Depois que todas as formigas construíram suas viagens, o feromônio é atualizado

- $\Delta\tau_{ij}^{k}$ é a quantidade de feromônio que a formiga k deposita sobre a aresta (i, j):

$$\begin{cases} \Delta\tau_{ij}^{k} = Q/L_k \text{ quando a aresta } (i, j) \text{ pertence } S_k \\ \Delta\tau_{ij}^{k} = 0 \text{ em caso contrário} \end{cases}$$

em que Q é uma constante

evaporação

$$\tau_{ij} = (1-\rho)\tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}$$

depósito

em que $0 < \rho \leq 1$ é a taxa de evaporação de feromônio

Atualização do feromônio na aresta $(3, 5)$

Apenas as formigas 1, 2 e 5 depositam feromônio nesta aresta. Suponha $Q = 1,0$. A contribuição de cada formiga:

k	viagem	L_k
1	1-2-3-5-4-1	9,8
2	2-1-4-5-3-2	9,8
3	3-4-5-1-2-3	10,9
4	4-5-2-1-3-4	11,6
5	5-2-1-4-3-5	12,4

$$\Delta\tau_{3,5}^{(1)} = 1/L_1 = 0,102$$
$$\Delta\tau_{3,5}^{(2)} = 1/L_2 = 0,102$$
$$\Delta\tau_{3,5}^{(5)} = 1/L_5 = 0,081$$

Suponha $\rho = 0,5$.

$$\begin{aligned}\tau_{3,5} &= (1-\rho)\tau_{3,5} + \Delta\tau_{3,5}^{(1)} + \Delta\tau_{3,5}^{(2)} + \Delta\tau_{3,5}^{(5)} \\ &= (1-0,5)1,0 + 0,102 + 0,102 + 0,081 \\ &= 0,785\end{aligned}$$

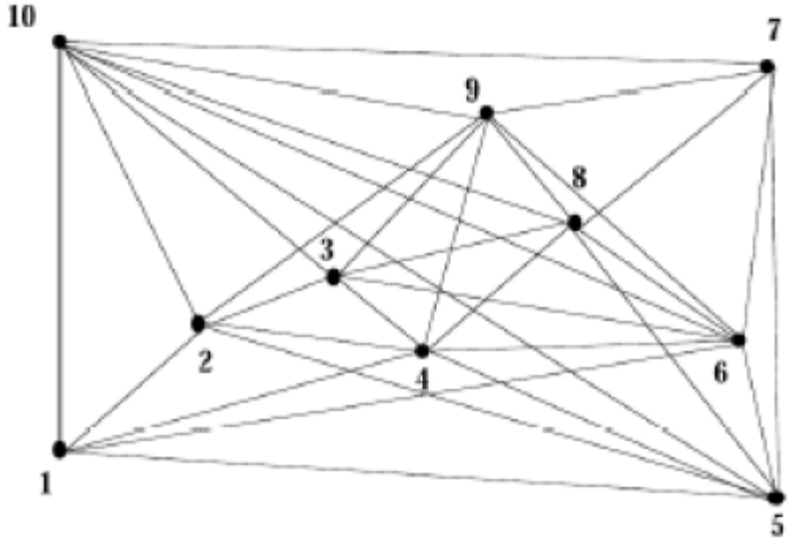
Critérios de parada

- Número máximo de iterações
- Estagnação ou convergência
- Situação na qual todas as formigas seguem sempre o mesmo percurso
- A estagnação é causado pelo excessivo crescimento de feromônio nas arestas de uma viagem sub-ótima

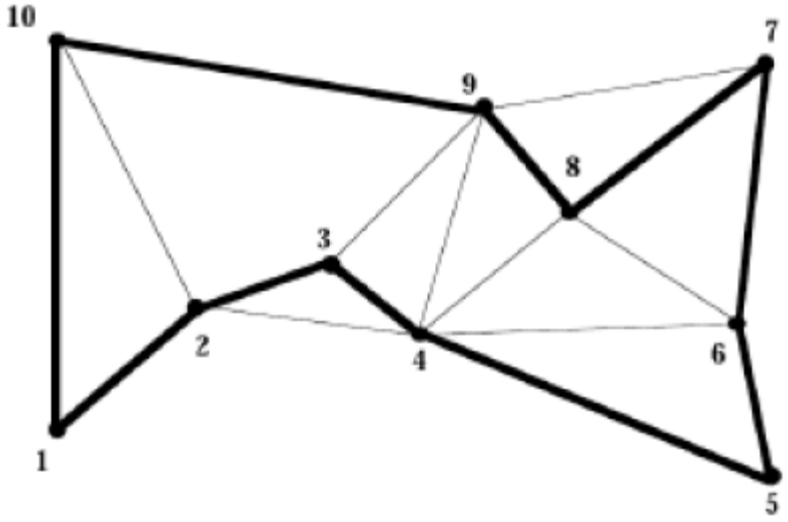
Estagnação

- Apesar da natureza estocástica do algoritmo, uma forte concentração de feromônio nas arestas força a formiga a fazer sempre o mesmo percurso

Distribuição de feromônio no inicio da busca



Distribuição de feromônio após 100 iterações



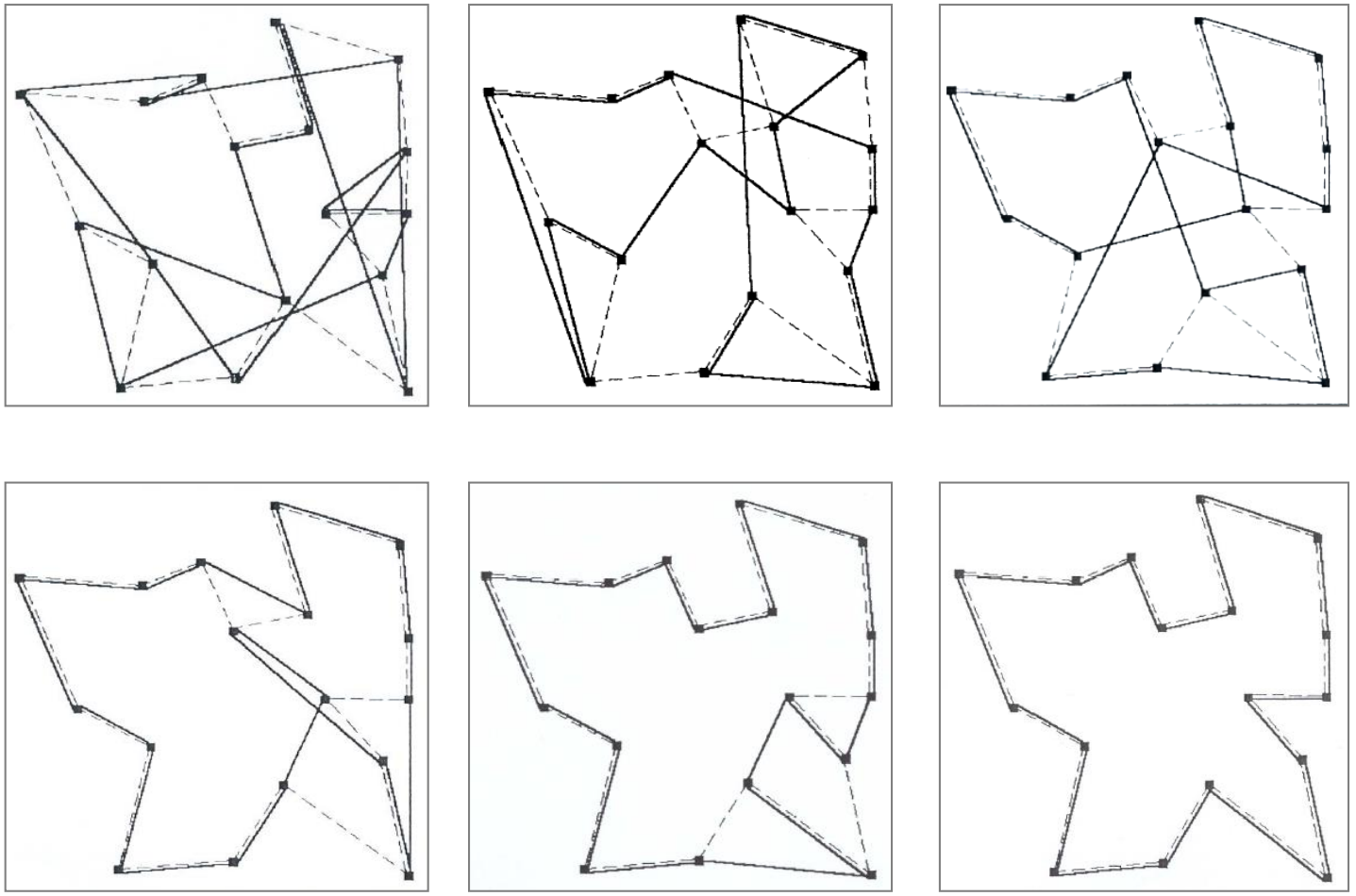
- As formigas artificiais possuem movimentação discreta, sendo que seus movimentos consistem em origens e destinos discretos

- Existe, nas formigas artificiais, um estado interno ou memória, para que não haja sobreposição de movimentos

- O depósito de feromônio no mundo artificial ocorre com base na qualidade da solução encontrada, diferentemente do mundo real, no qual formigas depositam feromônio sob demanda.

- Aproximação para o modelo computacional: formigas deixam o feromônio em cada arco visitado após chegar ao destino (na vida real as formigas deixam o feromônio

Resultados parciais em um PCV



——— Solução encontrada ---------- Solução ótima

- Applet:

  http://natcomp.liacs.nl/NC/applets/AntsApplet/AntsApplet.html

  http://www.openprocessing.org/sketch/15109

  http://www.djoh.net/inde/ANTColony/applet.html