3
Algoritmos Genéticos e Co-Evolução

Algoritmos Genéticos (AGs) constituem uma técnica de busca e otimização, altamente paralela, inspirada no princípio Darwiniano de seleção natural e reprodução genética (Michalewicz, 1996) que privilegia os indivíduos mais aptos com maior longevidade e, portanto, com maior probabilidade de reprodução.

Em AGs um cromossoma é uma estrutura de dados que representa uma das possíveis soluções do espaço de busca do problema. Cromossomas são então submetidos a um processo evolucionário que envolve avaliação, seleção, recombinação sexual (crossover) e mutação. Após vários ciclos de evolução a população deverá conter indivíduos mais aptos (Davis, 1996).

A representação das possíveis soluções do espaço de busca de um problema define a estrutura do cromossoma a ser manipulado pelo algoritmo e depende do tipo de problema e do que, essencialmente, se deseja manipular geneticamente. Os principais tipos de representação são: binária, números reais, inteiros, grupamento de inteiros e baseadas em ordem.

AGs têm sido aplicados a diversos problemas de otimização tais como: Otimização de Funções Matemáticas, Otimização de Planejamento, Problema do Caixeiro Viajante, Problema de Otimização de Rota de Veículos, etc.

Na seção seguinte, revisa-se a representação baseada em ordem e a sua aplicação no problema do caixeiro viajante.

3.1
Representação de Problemas Baseados em Ordem

3.1.1
Problema do Caixeiro Viajante

O problema do caixeiro viajante é um problema combinatorial do tipo NP-hard (Garey & Johnson, 1979) que oferece uma definição conceitualmente simples: um caixeiro viajante deve visitar todas as cidades da sua área uma única vez e retornar ao ponto de partida. Dado que existe um custo de transporte entre todas as cidades, deseja-se encontrar o itinerário com o menor custo possível. O

espaço de busca deste problema é um conjunto de permutações de *n* cidades e este espaço tem um tamanho de *n!* permutações possíveis (Michalewicz, 1996).

Três representações vetoriais para o cromossoma têm sido constantemente usadas para solução de problemas de TSP com algoritmos genéticos: de adjacência, ordinal e de caminho (Michalewicz, 1996).

Na representação de adjacência o caminho é representado por uma lista de *n* cidades. A cidade *j* aparece na posição *i* se e somente se o caminho levar da cidade *i* para a cidade *j*. Por exemplo, o vetor (2 4 8 3 9 7 1 5 6) representa o caminho 1-2-4-3-8-5-9-6-7.

Esta representação entretanto pode gerar caminhos ilegais. Por exemplo, o vetor (2 4 8 1 9 3 5 7 6) representa o caminho 1-2-4-1. Este caminho é ilegal porque o ciclo se fecha prematuramente, não passando por algumas das cidades. Portanto, esta representação não suporta os operadores genéticos convencionais necessitando de um algoritmo de reparação.

Na representação ordinal usa-se uma lista ordenada de referência com as cidades: (1 2 3 4 5 6 7 8 9), por exemplo. Para representar um caminho como, por exemplo, 1-2-4-3-8-5-9-6-7, deve-se usar um cromossoma definido como (1 1 2 1 4 1 3 1 1). Interpreta-se este cromossoma da seguinte forma: pega-se o primeiro número do cromossoma. Neste caso, este número é 1. Retira-se então a cidade número 1 da lista de referência (esta cidade é a cidade 1). O próximo número do cromossoma também é 1 e portanto tira-se novamente a primeira cidade da lista (que agora é 2 pois já foi retirada a cidade 1 no primeiro passo). Repete-se este procedimento até a lista ordenada de referência estar vazia.

Finalmente, na representação de caminho, que é provavelmente a mais natural das três representações, o vetor de cidades representa a ordem na qual as mesmas devem ser visitadas, ou seja, o vetor (5 1 7 8 9 4 6 2 3) representa o caminho 5-1-7-8-9-4-6-2-3.

De acordo com Michalewicz (1996), até recentemente, três tipos de crossover diferentes foram definidos para a representação de caminho. O crossover de mapeamento parcial (*Partially Mapped Crossover* – PMX), o crossover de ordem (*Order Crossover* – OX) e o crossover de ciclo (*Cycle Crossover* – CX).

O operador PMX constrói descendentes através da seleção de uma subseqüência de um caminho de um pai e mantendo a ordem e a posição de

quantas cidades for possível do outro pai. Esta subseqüência é escolhida selecionando-se dois pontos de corte aleatoriamente. Por exemplo, sejam os dois cromossomas abaixo (os pontos de corte estão indicados pelo sinal |):

$$p_1=(1\ 2\ 3\ |\ 4\ 5\ 6\ 7\ |\ 8\ 9)$$
$$p_2=(4\ 5\ 2\ |\ 1\ 8\ 7\ 6\ |\ 9\ 3) \quad (1)$$

Estes dois cromossomas produzem descendentes da seguinte maneira. Primeiro, os segmentos que estão entre os pontos de corte são trocados:

$$d_1=(x\ x\ x\ |\ 1\ 8\ 7\ 6\ |\ x\ x)$$
$$d_2=(x\ x\ x\ |\ 4\ 5\ 6\ 7\ |\ x\ x\ ) \quad (2)$$

Esta troca define também uma série de mapeamentos entre os valores:

$$1 \leftrightarrow 4;8 \leftrightarrow 5;7 \leftrightarrow 6;6 \leftrightarrow 7 \quad (3)$$

Preenche-se os elementos que faltam utilizando-se os elementos originais de cada pai, dado que não haja nenhum conflito ou seja, que não se esteja colocando uma cidade que já exista no cromossoma descendente:

$$d_1=(x\ 2\ 3\ |\ 1\ 8\ 7\ 6\ |\ x\ 9)$$
$$d_2=(x\ x\ 2\ |\ 4\ 5\ 6\ 7\ |\ 9\ 3\ ) \quad (4)$$

Finalmente, utilizou-se o mapeamento gerado pela troca feita anteriormente em (3):

$$d_1=(4\ 2\ 3\ |\ 1\ 8\ 7\ 6\ |\ 5\ 9)$$
$$d_2=(1\ 8\ 2\ |\ 4\ 5\ 6\ 7\ |\ 9\ 3\ ) \quad (5)$$

Este operador genético explora importantes similaridades nos valores e na ordem quando usado nas situações corretas.

O operador OX constrói os descendentes selecionando uma subseqüência de um caminho de um pai e preservando a ordem relativa das cidades do outro pai. Por exemplo, sejam os dois cromossomas definidos em (1). Os descendentes são gerados da seguinte forma: primeiro seleciona-se os pontos de corte. Supondo que os cortes sejam os mesmos de (2), copia-se as cidades de um pai para o outro a partir do segundo ponto de corte, mantendo-se a mesma ordem e omitindo-se os símbolos já presentes. Esta seqüência de cidades do segundo pai por exemplo, a partir do segundo ponto de corte seria 9-3-4-5-2-1-8-7-6. Removendo-se as cidades 4, 5, 6 e 7 que já existem no primeiro descendente, teremos a seqüência 9-3-2-1-8. Coloca-se esta seqüência de cidades no primeiro descendente (também a partir do segundo ponto de corte) e repete-se a operação para o segundo (selecionando a seqüência de cidades do primeiro pai). Como resultado obtém-se:

$$\begin{aligned} d_1&=(2\ 1\ 8\ |\ 4\ 5\ 6\ 7\ |\ 9\ 3) \\ d_2&=(3\ 4\ 5\ |\ 1\ 8\ 7\ 6\ |\ 9\ 2) \end{aligned} \tag{6}$$

Este operador explora o fato de que a ordem das cidades é que é importante e não a sua posição. Isto pode ser observado nos caminhos 9-3-4-5-2-1-8-7 e 4-5-2-1-8-7-6-9-3 que, apesar de serem diferentes com relação a posição, seguem a mesma ordem e portanto representam o mesmo caminho.

Finalmente, o operador CX constrói descendentes de modo que cada cidade e a sua posição venham de um dos pais. Isto funciona da seguinte maneira: sejam os dois cromossomas abaixo:

$$\begin{aligned} p_1&=(1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9) \\ p_2&=(4\ 5\ 2\ 1\ 8\ 7\ 6\ 9\ 3) \end{aligned} \tag{7}$$

O primeiro descendente é produzido pegando-se o primeiro elemento do primeiro pai (neste caso, 1). A cidade correspondente a esta no segundo pai é a cidade 4 e portanto temos que colocá-la no primeiro descendente, na mesma posição em que ela se encontra no primeiro pai. Desta forma, teremos um cromossoma com a seguinte forma:

$$d_1=(1\ x\ x\ 4\ x\ x\ x\ x\ x) \tag{8}$$

Por sua vez, a escolha da cidade 4 implica na escolha da cidade 8 (que é a que está na posição correspondente no segundo pai). Seguindo esta regra, as próximas cidades a serem escolhidas são a 3 e a 2.

$$d_1=(1\ 2\ 3\ 4\ x\ x\ x\ 8\ x) \tag{9}$$

Deve-se notar no entanto que, após escolher a cidade dois, a próxima cidade que deve ser incluída é a 1, que já está no cromossoma descendente. Portanto, tem-se um ciclo completo e agora deve-se pegar as próximas cidades dos segundo pai. Repetindo-se esta operação para o segundo descendente tem-se finalmente:

$$\begin{aligned} d_1&=(1\ 2\ 3\ 4\ 7\ 6\ 9\ 8\ 5)\\ d_2&=(4\ 1\ 2\ 8\ 5\ 6\ 7\ 3\ 9) \end{aligned} \tag{10}$$

Este operador preserva a posição absoluta dos elementos dos pais nos seus descendentes.

Além disso, cabe citar o operador de recombinação de adjacências. (edge recombination crossover – ERX). Este operador se baseia na idéia geral de que os descendentes devem ser construídos exclusivamente a partir das adjacências presentes nos dois pais. Isto é feito com o auxílio de uma lista de adjacências criada a partir dos caminhos dos dois pais. Esta lista fornece, para cada cidade *c*, todas as outras cidades conectadas a ela em pelo menos um dos pais. Obviamente, para cada cidade *c* existem pelo menos duas e no máximo quatro cidades nesta lista. Por exemplo, dados os indivíduos:

$$\begin{aligned} d_1&=(1\ 2\ 3\ 4\ 7\ 6\ 9\ 8\ 5)\\ d_2&=(4\ 1\ 2\ 8\ 7\ 6\ 9\ 3\ 5) \end{aligned} \tag{11}$$

A lista de adjacências para estes dois cromossomas é:

Cidade	Tem adjacência com
1	9,2,4
2	1,3,8
3	2,4,9,5
4	3,5,1
5	4,6,3
6	5,7,9
7	6,8
8	7,9,2
9	8,1,6,3

Tabela 2 – Lista de adjacências

A construção dos descendentes começa com a seleção de uma cidade inicial de um dos pais (neste caso, 1 ou 4). A cidade com o menor número de adjacências na lista de adjacências é a selecionada. Caso os números de adjacências sejam iguais, uma escolha aleatória pode ser feita. Desta forma, teremos uma chance maior de completar o caminho com todas as adjacências sendo selecionadas dos pais. Se a escolha fosse meramente aleatória, a chance de ocorrer uma falha de adjacência, ou seja, sobrar uma cidade sem continuação da adjacência, seria muito maior.

Supondo então que a cidade 1 fosse escolhida, nota-se pela lista que ela está conectada diretamente a três outras cidade: 9, 2 e 4. A próxima cidade escolhida deve então ser uma dessas. Como as cidades 4 e 2 têm apenas 3 adjacências e a cidade 9 tem 4 adjacências, uma escolha aleatória é feita entre a 4 e a 2. Supondo que a cidade 4 tenha sido a escolhida, as candidatas para a próxima cidade no caminho seriam 3 e 5. Prosseguindo com este algoritmo, poderia se obter um cromossoma com a configuração (1 4 5 6 7 8 2 3 9).

Para operadores genéticos de mutação existem basicamente quatro tipos de operadores: Swap, Inversão de Posição (Position Inversion – PI), Rotação a Esquerda (Rotation Left Mutation – RLM), Rotação a Direita (Rotation Right Mutation – RRM).

O operador Swap seleciona duas cidades aleatoriamente e troca suas posições. Desta forma o cromossoma (1 2 3 4 5 6 7 8 9) após serem selecionadas para mutação a cidade 3 e 8 fica (1 2 8 4 5 6 7 3 9).

O operador PI seleciona dois pontos de corte no cromossoma e inverte os elementos situados entre estes dois ponto. Suponha o cromossoma anterior com os seguintes pontos de corte, designados pelo símbolo |: (1 2 3 | 4 5 6 7 | 8 9). Após a mutação este cromossoma se torna (1 2 3| 7 6 5 4| 8 9).

Os operadores de rotação a esquerda e a direita simplesmente deslocam o cromossoma um número aleatório de posições para a esquerda ou para a direita. Por exemplo, se o número de posições sorteadas for 3 o cromossoma do exemplo anterior ficaria (4 5 6 7 8 9 1 2 3) após a rotação para a esquerda ou (7 8 9 1 2 3 4 5 6) após a rotação para a direita.

Finalmente, existe um algoritmo evolucionário chamado *inver-over* que também é utilizado para problemas baseados em ordem (Michalewicz & Fogel, 2000). Ele tem o propósito de ser um algoritmo evolucionário "puro", ou seja, não usa nenhum tipo de busca local e, ao mesmo tempo, ser um modelo competitivo com relação aos algoritmos que usam busca local.

Este algoritmo tem as seguintes características:

- Cada indivíduo compete apenas com os seus descendentes;
- Existe apenas um operador de variação mas este operador (*inver-over*) é adaptativo;
- O número de vezes que o operador é aplicado a um indivíduo durante uma única geração é variável.

O algoritmo é mostrado na figura 3:

```
procedure inver-over
    input população P
    while (not condição de término)
        for each indivíduo Si em P
            S'←Si
            selecione aleatoriamente uma cidade c de S'
            repeat
                if (rand()<=p)
                    selecione a cidade c' das cidades que sobraram
                        em S'
                else
                    selecione aleatoriamente um indivíduo de P
                    associe à cidade c' a cidade adjacente a cidade
                        c no indivíduo selecionado
                if (a cidade anterior ou posterior a cidade c em S' é c')
                    exit repeat
                inverter a seção entre a cidade posterior a cidade c até
                    a cidade c' em S'
                c←c'
            end repeat
            if (avaliação(S') <= avaliação(Si))
                Si←S'
        end for each
    end while
end procedure
```

Figura 3 – Algoritmo inver-over

Este algoritmo pode ser entendido como um conjunto de procedimentos paralelos de "subida de montanha" (*hill climbing*). Cada "alpinista" (*hill climber*) executa um número variável de trocas de adjacências. Entretanto, o operador inver-over tem componentes adaptativos:

1. o número de inversões aplicadas a um único indivíduo;
2. o segmento a ser invertido é determinado por outro indivíduo selecionado aleatoriamente.

Desta forma, é sempre possível visualizar este método como um algoritmo evolucionário com uma forte pressão seletiva e um operador de variação adaptativo.

3.1.2
Problema do Caixeiro Viajante com Restrições de Precedência

Quando se deseja tratar problemas de planejamento com restrições de precedência, a representação baseada em ordem, muito utilizada para solucionar estes tipos de problemas, torna-se ineficaz pois ela pode gerar soluções inviáveis que devem ser descartadas (Moon et al, 2002).

Os problemas de planejamento com restrições de precedência se caracterizam pela necessidade de executar ou planejar, obrigatoriamente, algumas tarefas antes de outras. Pode-se adaptar o problema do caixeiro viajante para que ele tenha estas restrições de precedência (Travelling Salesman Problem with Precedence Constraints – TSPPC). Nesta situação, algumas cidades devem ser visitadas obrigatoriamente antes de outras. Quando se usa uma representação baseada em ordem para solucionar este tipo de problema geram-se soluções incorretas (ou seja, que não atendem as restrições de precedência) e surge a necessidade de corrigi-las através de algum tipo de heurística, aumentando consideravelmente a complexidade do modelo. Como conseqüência, a performance do algoritmo evolucionário é degradada o que torna esta maneira de tratar o problema insatisfatória.

Alguns tipos de abordagem têm sido utilizados para se resolver problemas deste tipo. Em Renaud et al (2000) foi usada uma solução heurística para resolver o problema do TSPPC. Em Mingozzi et al (1997) foram usadas estratégias de programação dinâmica para o problema do caixeiro viajante com janelas de tempo e restrições de precedência. Em Fagerholt & Christiansen (2000) um modelo para otimizar uma aplicação voltada para o planejamento de navegação foi solucionado como um problema do tipo *shortest path* em um grafo. Em Moon et al (2002), foi utilizado um algoritmo genético associado a um grafo dirigido para solução do problema de TSPPC. Este último modelo apresentou resultados superiores em relação aos algoritmos tradicionais

Curiosamente, este tipo de problema tem sido pouco tratado na área de computação evolucionária apesar de contar com uma variedade de aplicações práticas entre as quais podemos citar: planejamento de embarque de minério em portos (onde no caso tratado aqui, os navios devem ser atracados na ordem em que chegam ao porto, o embarque do minério não pode ser feito antes que o minério tenha sido virado do trem e não se pode virar o minério antes que uma área de estocagem para ele tenha sido definida), problemas de transporte de carga (veículos de transporte devem recolher cargas em determinados pontos e distribuí-las em outros locais), entre outros.

3.2
Co-Evolução

3.2.1
Co-Evolução Cooperativa

Para se aplicar algoritmos evolucionários com sucesso em problemas com complexidade cada vez maior, torna-se necessário introduzir noções explícitas de modularidade nas soluções para que elas disponham de oportunidades razoáveis de evoluir na forma de subcomponentes co-adaptados (Potter & Jong, 2000).

Existem duas razões principais pelas quais algoritmos evolucionários convencionais não são totalmente adequados para resolver este tipo de problema. Em primeiro lugar, os algoritmos genéticos convencionais impedem, a longo prazo, a preservação de certos componentes da solução pois, por estarem codificados por completo em um indivíduo, eles são avaliados como um todo e apenas os subcomponentes que pertencem a indivíduos com avaliações altas serão preservados. Em segundo lugar, o fato da representação estar relacionada a uma solução completa e por não haver interações entre os membros da população, não existe pressão evolucionária para a ocorrência de co-adaptação, ou seja, não existe pressão para a adaptação de um subcomponente dada a ocorrência de uma mudança em outro subcomponente (Potter, 2000).

A decomposição do problema no entanto tem um aspecto importante que deve ser levado em conta e que está relacionado com a evolução de subcomponentes interdependentes. Se for possível decompor o problema em subcomponentes independentes, cada um deles pode evoluir sem haver

necessidade de se preocupar com os outros. Pode-se visualizar isto imaginando-se que cada subcomponente evolui no seu próprio espaço de busca, desacoplado do espaço de busca dos outros componentes. Entretanto, a maioria dos problemas só pode ser decomposta em subcomponentes que possuem uma interdependência complexa. O efeito de mudar um desses subcomponentes provoca uma "deformação" no espaço de buscas dos outros subcomponentes que estão acoplados por esta interdependência. É possível visualizar isto graficamente da seguinte forma: suponha que deseje-se minimizar uma função de duas variáveis e supondo que essas duas variáveis formem dois subcomponentes interdependentes. Esta função pode ser por exemplo descrita pela equação:

$$f(x,y) = x^2 + (x+y)^2 \tag{12}$$

O gráfico desta equação é mostrado a seguir:

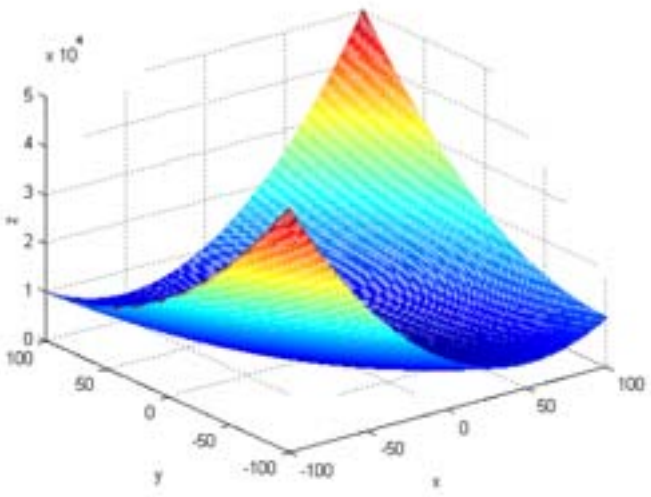

Figura 4 – Gráfico da equação

Supondo que deseje-se encontrar o valor mínimo desta função, que ocorre quando *x* e *y* são iguais a zero (sendo então *f(x,y)* igual a zero), o espaço de busca da variável y, assume a seguinte forma quando *x* é igual a 40:

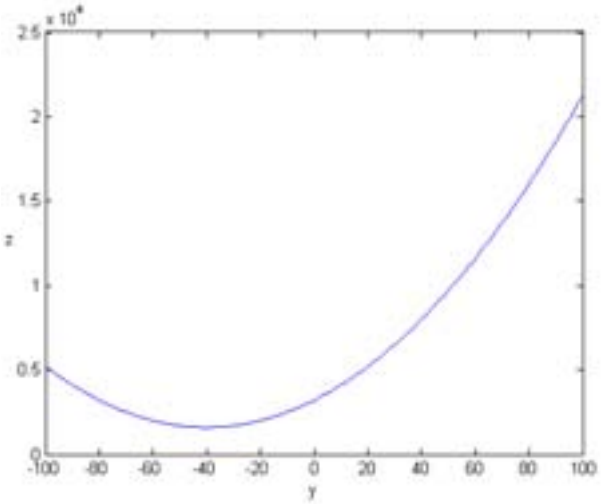

Figura 5 – Gráfico para x igual a 40

Quando a variável *x* por outro lado, assume o valor -40, o espaço de busca da variável *y* se deforma e assume a seguinte forma:

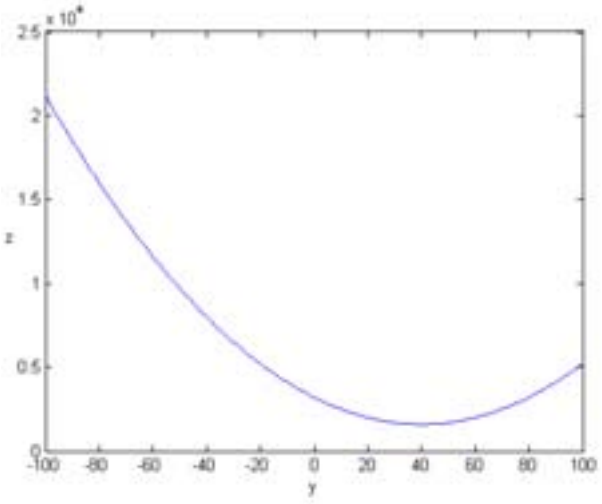

Figura 6 – Gráfico para x igual a -40

Da figura 5 para a figura 6 pode-se notar a deformação do espaço de busca da variável *y* quando muda-se o valor da variável *x*. No primeiro caso, o mínimo no espaço de buscas de *y* encontra-se em -40 enquanto que no segundo caso o mínimo encontra-se em 40.

Espera-se, portanto, que com o desacoplamento e com a capacidade de adaptação dos algoritmos evolucionários os mesmos consigam se adaptar facilmente a este tipo de dinâmica do problema.

Várias pesquisas foram feitas visando estender o modelo evolucionário básico de modo que o mesmo suportasse subcomponentes co-adaptados. Uma das primeiras extensões feitas foi o modelo de sistemas classificadores (Holland, 1978)(Holland, 1986). Em Giordana et al. (1994) tem-se um sistema (chamado REGAL) que aprende regras de classificação e que utiliza um operador chamado "voto universal" (*universal suffrage*) para fazer a decomposição do problema. Em Giordana & Neri. (1996) o desempenho do sistema REGAL foi melhorado através do uso de ilhas populacionais semi-isoladas. Modelos com múltiplas espécies, ou seja, que incorporam populações geneticamente isoladas também têm sido usados para evoluir subcomponentes co-adaptados. Trabalhos nesta área incluem modelos hospedeiros-parasitas (*host-parasite*) para redes de ordenação onde uma espécie (os hospedeiros) evolui redes de ordenação para serem aplicadas em casos de teste representados pela outra espécie (os parasitas) como uma lista de números a serem ordenados (Hillis, 1991). Outros modelos com duas espécies competindo foram usados para solucionar problemas de aprendizado de jogos fazendo com que cada uma das espécies representasse um dos jogadores (Rosin & Belew, 1995).

Alguns outros pesquisadores exploraram o uso de modelos com espécies em cooperação como, por exemplo, para a função de três bits de Goldberg (Paredis, 1995) e para a otimização de funções (Potter & Jong, 1994). Finalmente, em Potter & Jong (2000), procurou-se definir um modelo co-evolucionário cooperativo genérico.

Nesta arquitetura duas ou mais espécies diferentes formam um ecossistema. Como na natureza, as espécies são geneticamente isoladas ou seja, os indivíduos só podem se reproduzir com outros indivíduos da mesma espécie. Isto é feito simplesmente isolando-se as espécies em populações separadas. As espécies somente interagem umas com as outras através de um domínio compartilhado e tem uma relação apenas de cooperação.

O modelo co-evolucionário cooperativo genérico (Potter & Jong, 2000) é mostrado na figura 7:

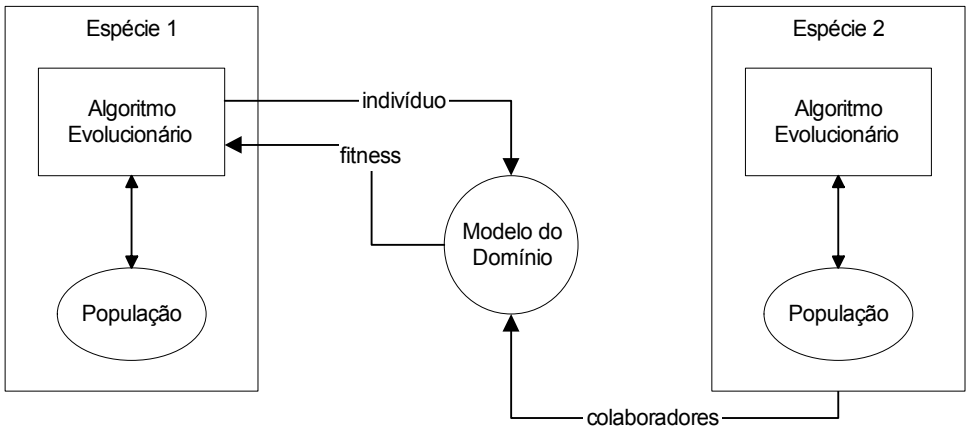

Figura 7 – Modelo co-evolucionário genérico

Apesar de serem mostradas apenas duas espécies neste modelo, o mesmo pode ser usado para *n* espécies diferentes. Cada espécie evolui em sua própria população (pois como já foi mencionado, elas são isoladas geneticamente) e se adaptam ao ambiente através de repetidas aplicações do algoritmo evolucionário. Para se calcular o *fitness* dos indivíduos de uma determinada espécie, deve-se submetê-lo ao modelo do domínio (que contém a função de avaliação) juntamente com um ou mais colaboradores de cada uma das outras espécies (de modo a formar a solução completa).

3.2.2
Métodos de Interação entre as Populações

O processo de seleção dos colaboradores (*collaborator selection*) é feito em geral usando-se o último conjunto de avaliações feitas em cada espécie. O grau com que estas avaliações vão ter peso na seleção dos colaboradores é chamado de pressão de seleção de colaboradores (*collaborator selection pressure*).

A solução final encontrada ao juntar-se indivíduos de cada uma das espécies de modo a formar um elemento que represente a solução completa é chamada de colaboração. Após ter-se montado a colaboração, a mesma pode ser submetida a função de avaliação no modelo do domínio. Se apenas um indivíduo de cada espécie for selecionado para construir a colaboração, ter-se-á apenas um valor de avaliação que pode ser usado como *fitness*.

Entretanto, pode-se escolher diferentes combinações de colaboradores das outras espécies. Neste caso, a avaliação de um indivíduo de uma espécie vai consistir de múltiplas colaborações. O número de colaboradores de cada uma das outras espécies é chamado de tamanho do grupo de colaboração (*collaboration pool size*). Como cada uma dessas colaborações vai ter uma avaliação diferente, é necessário a partir delas calcular uma avaliação única para o *fitness*. Esta operação chama-se associação de crédito de colaboração (*collaboration credit assignment*).

Como já foi dito, existem três aspectos com relação a interação entre as diversas espécies do modelo co-evolucionário: a associação de crédito de colaboração, a pressão de seleção de colaboradores e o tamanho do grupo de colaboração.

Existem três métodos diferentes para se definir como o crédito de colaboração deve ser associado ao indivíduo (Wiegand et al, 2001):

- Otimista: este é o método mais tradicional. Neste método, associa-se ao *fitness* do indivíduo o valor da avaliação da melhor colaboração feita por ele.
- Média: neste método, associa-se ao *fitness* do indivíduo o valor médio das colaborações feitas por ele.
- Pessimista: neste modelo, associa ao *fitness* do indivíduo o valor da sua pior colaboração.

A justificativa para o método pessimista é de que talvez seja melhor usar um valor que seja mais "seguro", que premia o indivíduo pela sua pior colaboração. Entretanto, em Wiegand et al (2001), todos os testes feitos com este modelo e o modelo de média produziram resultados fracos com relação ao modelo otimista.

Com relação a pressão de seleção de colaboradores existem diversos métodos diferentes que variam o grau de pressão exercido nesta seleção. Pode-se, por exemplo, utilizar um método extremamente ganancioso que seleciona sempre o melhor indivíduo da geração anterior. Ou, pode-se aliviar um pouco esta pressão selecionando-se dois colaboradores sendo o primeiro o melhor da geração anterior e o segundo um indivíduo escolhido aleatoriamente. De qualquer forma, este método ainda é ganancioso dado que continua utilizando o melhor indivíduo. Para

aliviar ainda mais esta pressão, pode-se escolher por exemplo o melhor e o pior indivíduo da geração anterior e um indivíduo aleatório.

Finalmente, a escolha do tamanho do grupo de colaboração, que consiste no fator mais importante para o sucesso dos algoritmos co-evolucionários cooperativos (Wiegand et al, 2001), se resume a decidir quantos elementos das outras espécies, irão participar na colaboração. Em geral, aumentar o número de colaboradores melhora a performance do algoritmo, mas por outro lado aumenta o trabalho computacional consideravelmente, de forma proporcional ao número de espécies do modelo.