3.
Algoritmos Genéticos e Co-evolução Cooperativa.

3.1.
Introdução.

A Computação Evolucionária é uma abordagem para resolução de problemas inspirada na Teoria da Evolução de Darwin, principalmente no princípio da sobrevivência dos mais aptos.

> "Como muito mais indivíduos de cada espécie são gerados do que os que teriam possibilidade de sobreviver; e como, conseqüentemente, há um esforço freqüentemente recorrente para a existência: qualquer ser vivo, se variar, mesmo que ligeiramente, de qualquer maneira lucrativa a si próprio, sob as complexas e, às vezes, variáveis condições de vida, terá uma possibilidade melhor de sobreviver, e assim de ser *selecionado naturalmente*. Do princípio forte da herança, toda a variedade selecionada tenderá a propagar sua nova e modificada forma". (Darwin, 1859)

Algoritmos Evolucionários (EAs) tipicamente são utilizados para resolver problemas na forma:

$$f : S \rightarrow \Re \qquad \textbf{(3.1)}$$

Onde *S* é um espaço de busca constituído por todas as possíveis soluções para um problema particular. Dependendo das peculiaridades do problema, as soluções podem ser representadas por vetores n-dimensionais de binários, inteiros, reais, ou estruturas mais complexas. Para todas as soluções existentes no domínio de *S*, um número real é associado, medindo o quão adequada é a solução para resolver o problema em questão. A tarefa principal de um algoritmo evolucionário é buscar de forma eficiente, em amostras do espaço de busca *S*, soluções que estejam de acordo com o objetivo de problema. É importante mencionar que estas soluções não precisam ser necessariamente *ótimas*, mas sim *satisfatórias*. Ao se

lidar com espaços de busca grandes e complexos, o ótimo pode ser difícil de se atingir e, neste caso, pode-se apenas esperar achar uma solução satisfatória (Zebulum, 2001).

Segundo Zebulum (2001), o funcionamento de um algoritmo evolucionário é definido de forma genérica. A figura 4 mostra este fluxo.

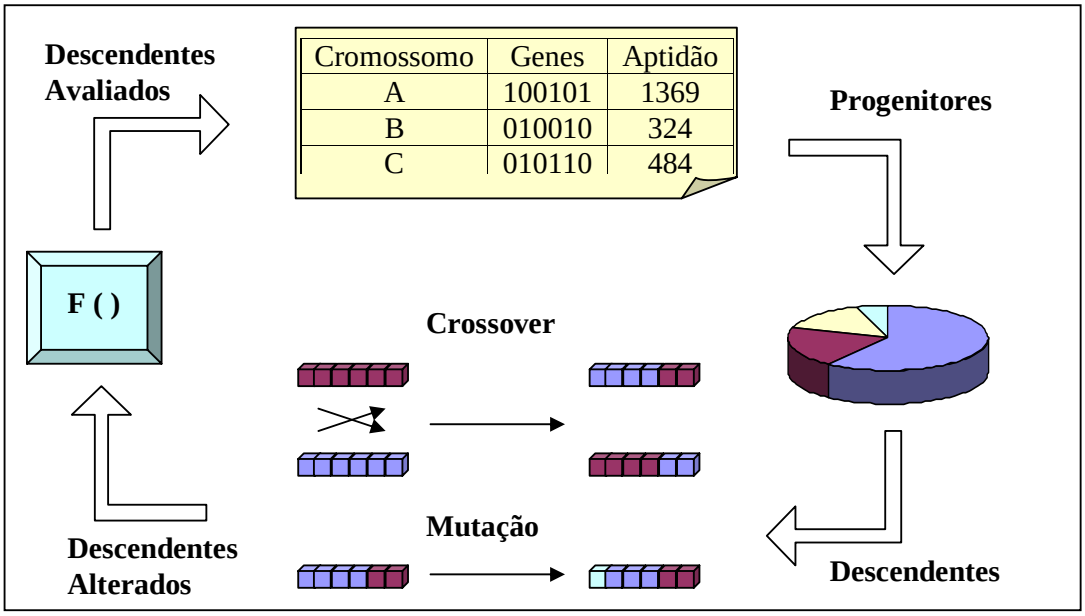

Figura 4-Fluxo básico de um algoritmo evolucionário.

Dado um problema particular de busca, uma representação adequada deve ser selecionada para codificar possíveis soluções em estruturas de dados do tipo definido no conjunto *S*. Após a representação ser escolhida, um número *N* de potenciais soluções, também chamadas de *indivíduos*, é gerado aleatoriamente. Estes indivíduos passam então por duas etapas básicas: avaliação e operações genéticas. Durante a avaliação, um número real, também chamado de *aptidão*, é associado a cada indivíduo. A aptidão do indivíduo mede o quanto ele é adequado para satisfazer à especificação de um problema particular. Após serem avaliados, as seguintes operações, ou *operadores genéticos*, são utilizados nos indivíduos: *seleção*, *crossover* e *mutação*. Depois de passarem por tais operadores, uma nova população é criada, formando assim a próxima *geração*. A avaliação e os operadores genéticos são aplicados na próxima geração e o processo continua, até que um critério de parada é satisfeito. Este critério de parada pode ser um número máximo de gerações ou ter sido alcançada uma solução adequada para o problema (Zebulum, 2001).

Os conceitos básicos de um algoritmo evolucionário que foram apresentados acima são mostrados a seguir em maiores detalhes, como apresentados por Zebulum (2001).

Representação. A representação se refere à forma como as soluções de um determinado problema são codificadas em uma estrutura de dados que possa ser processada num computador digital. Uma solução para um dado problema pode ser:

- Um número, representando o valor ótimo de uma função matemática;
- Um vetor de reais, representando o valor ótimo de uma função com múltiplas variáveis;
- Uma lista de eventos, representando uma ordem ótima de eventos para se realizar uma determinada tarefa;
- Uma estrutura, simbolizando algum modelo de engenharia, um circuito elétrico, uma reação química ou qualquer outro sistema.

Para representar soluções nestas classes de aplicações, os algoritmos evolucionários normalmente utilizam vários tipos de estruturas de dados:

- Vetores binários;
- Vetores de valores inteiros ou reais;
- Representações de estados finitos;
- Entre outras.

Avaliação. A avaliação é o processo de associar um valor de aptidão para cada cromossomo, ou indivíduo, selecionado pelo algoritmo evolucionário. Na natureza, a aptidão de um indivíduo mede o quão bem adaptado ele está a um determinado ambiente. Da mesma forma, no caso dos algoritmos evolucionários, a aptidão mede o desempenho de um indivíduo de acordo com a especificação de um problema. Esta medida é, normalmente, um valor escalar, inteiro ou real. No caso da definição padrão de problemas de busca (3.1), pode-se observar que um número real, o valor de aptidão, é associado com cada ponto do espaço de busca *S*. Quando há somente um objetivo a ser satisfeito no problema em questão, a função de avaliação dos indivíduos é normalmente encontrada de forma bem direta. Entretanto, quando um ou mais objetivos devem ser perseguidos pelo algoritmo, encontrar esta função pode ser uma tarefa bem complexa.

Operadores. Existem três mecanismos naturais nos quais os algoritmos evolucionários são baseados: seleção natural, recombinação e mutação. Estes são os principais operadores genéticos utilizados nos algoritmos evolucionários.

- Seleção: O operador de seleção é um componente essencial de um EA. Este operador, baseado no valor da aptidão dos indivíduos, seleciona aqueles que farão parte da próxima geração;
- *Crossover*: Este operador é inspirado na idéia da recombinação de material genético entre indivíduos. Como mostrado na figura 4, o *crossover* ocorre após a seleção. Este operador é aplicado de forma probabilística nos indivíduos. Dois indivíduos são selecionados aleatoriamente e, de acordo com uma probabilidade pré-definida, seu material genético é recombinado ou não. Se isto ocorrer, dois novos indivíduos com material de ambos os progenitores são gerados; caso contrário, os dois indivíduos permanecem inalterados e são passados para a próxima geração;
- Mutação: O operador de mutação fornece ao algoritmo um comportamento exploratório, já que o induz a buscar novos pontos no espaço de busca. Se um EA fosse desenvolvido baseando-se apenas em seleção e *crossover*, o sistema iria convergir prematuramente, já que o operador de *crossover* gera novos indivíduos de forma muito limitada após algumas gerações. Por isso, a mutação é essencial para manter a diversidade e renovar o material genético. A mutação é tipicamente o último passo do processo reprodutivo. Depois do *crossover*, existe um conjunto de N indivíduos, alguns deles modificados pela recombinação. Então, a mutação é aplicada, modificando de forma diversa os indivíduos. Por ser uma mudança extremamente agressiva e, até certo ponto, imprevisível quanto aos resultados, a taxa de aplicação deste operador é normalmente muito baixa.

Do ponto de vista de otimização, uma das principais vantagens das técnicas de computação evolucionária é que estas não impõem muitos requisitos matemáticos sobre o problema a ser otimizado. Só necessitam da avaliação da função objetivo, e com isso podem tratar uma gama enorme de problemas, sejam

eles definidos num espaço discreto, contínuo ou misto, com ou sem restrições (Michalewicz et al, 1996).

Na próxima seção, serão apresentadas técnicas evolucionárias já utilizadas para a solução de problemas de *scheduling*.

3.2. Representação de Problemas de *Scheduling*.

Problemas de *scheduling* são problemas de otimização tipicamente combinatoriais. O *scheduling* consiste em alocar tarefas ao longo do tempo, as quais devem ser realizadas utilizando um número limitado de recursos, respeitando diversas restrições e otimizando um ou mais objetivos. Uma tarefa é dada como completa após a execução de uma seqüência pré-definida de operações, e o resultado final do *scheduling* consiste em um plano que exibe todas as tarefas, associadas a seus instantes iniciais e finais e aos recursos usados para cada operação. Estas associações de tempo e recursos afetam a qualidade de um planejamento com respeito a critérios como custo, tempo de execução, entre outros (Cruz, 2003).

Problemas do tipo *flow shop* são aqueles em que todas as diferentes tarefas são feitas pela mesma seqüência de recursos. Um planejamento é, portanto, determinado pela ordem na qual as tarefas são programadas. Em problemas do tipo *job shop*, cada tarefa necessita utilizar cada recurso somente uma vez (Bäck et al, 1997).

Segundo Wall (1996), em sua forma mais genérica, o problema de *scheduling* com restrições pode ser formulado como:

Dados:

- Um conjunto de atividades ou tarefas que devem ser executadas;
- Um conjunto de recursos para realizar estas tarefas;
- Um conjunto de restrições que devem ser satisfeitas;
- Um conjunto de objetivos utilizados para se avaliar o desempenho do planejamento;

Qual a melhor forma de atribuir os recursos às atividades em determinado momento, de modo que todas as restrições sejam respeitadas e que a melhor avaliação do planejamento seja produzida?

Nesta forma, este problema inclui as seguintes características:

- Cada tarefa pode ser executada de uma ou mais maneiras diferentes, dependendo do recurso a ela atribuído;
- Relacionamentos de precedência podem existir entre tarefas e podem incluir sobreposições, de modo que uma dada tarefa pode começar quando seu predecessor estiver parcialmente realizado;
- Cada tarefa pode ser interrompida de acordo com um conjunto pré-definido de modos de interrupção, ou nenhuma interrupção pode ser permitida;
- Cada tarefa pode requerer a alocação de mais de um recurso de tipos diversos;
- Os requisitos de um recurso alocado a uma tarefa podem variar ao longo da duração da mesma;
- Os recursos podem ser renováveis (ex. máquinas, pessoas) ou não-renováveis (ex. matérias-primas);
- A disponibilidade de recursos pode variar ao longo da duração de um planejamento, ou mesmo de uma tarefa;
- Os recursos podem sofrer restrições temporais;

De modo a modelar problemas reais, a formulação do problema de *scheduling* deve admitir as seguintes características:

- A disponibilidade de recursos pode mudar;
- Os requisitos dos recursos podem mudar;
- Os objetivos podem mudar.

A qualidade de um *scheduling* é medida por uma função objetivo que associa um valor numérico a uma programação. Vários objetivos podem ser identificados neste tipo de problema como, por exemplo, duração, atraso, custo de inventário, custo de transporte, etc. Normalmente, somente um objetivo não é suficiente para se avaliar uma programação, mas sim uma combinação destes diversos e, às vezes, conflitantes objetivos. Um exemplo de objetivos conflitantes é o de se querer maximizar o tempo de trabalho dos recursos e minimizar a utilização dos recursos. Então, pode-se perceber que os problemas de *scheduling* são tipicamente problemas de otimização com múltiplos objetivos. Em resumo, o

objetivo final da otimização de um problema de *scheduling* é a construção de um horizonte de programação completo e viável, que minimize ou maximize os vários critérios desejados.

Fora alguns casos teóricos e de pouca importância prática, determinar uma solução ótima para um problema de *scheduling* é um problema do tipo *NP-hard* (Bäck et al, 1997), o que significa que não existe ainda algoritmo determinístico conhecido que resolva este problema em tempo polinomial. A prática prova que a programação é, também, uma tarefa extremamente árdua para especialistas humanos. Quando são abordados problemas do mundo real, onde, além da complexidade da parte combinatória, tem-se que lidar com tipos diferentes de restrições específicas, impostas por vários detalhes particulares do problema em questão como, por exemplo, restrições físicas nas capacidades operacionais dos recursos, preferências na operação, restrições técnicas nos procedimentos de manufatura e produção, a resolução deste problema se torna ainda mais difícil. A programação da produção pode, então, ser descrita como um problema complexo de satisfação de restrições (Bäck et al, 1997).

Segundo Bäck (1997), ao mencionar as abordagens anteriores existentes para problemas de *scheduling*, as maiores dificuldades em se aplicar técnicas evolucionárias a problemas desta natureza ainda são: encontrar uma representação adequada e tratar as várias restrições encontradas neste tipo de problema.

Esta é uma limitação apresentada por grande parte das abordagens evolucionárias para problemas do mundo real, principalmente na engenharia, identificada e comentada por Alle (2003) como uma das deficiências das abordagens evolucionárias ao problema. Mas, segundo Michalewicz (1996), pode-se ver que as técnicas evolucionárias não contam apenas com funções de penalidade, descarte e correção de soluções inviáveis para tratar as restrições do problema. Michalewicz (1996) apresenta várias opções de se lidar com as restrições de um problema, ao se utilizarem técnicas evolucionárias. Nesta dissertação, a representação e a forma de decodificar cada um dos indivíduos não permitem a geração de soluções inválidas e, por isso, não é necessário descartar, penalizar ou corrigir soluções inválidas.

A seguir, apresenta-se uma relação com algumas das representações utilizadas para problemas de *scheduling*.

3.2.1.
Abordagens Anteriores do Problema utilizando Algoritmos Genéticos.

Problemas de *scheduling* têm recebido muita atenção e considerável esforço de pesquisa da comunidade de computação evolucionária (Bäck et al, 1997). Vários algoritmos genéticos e algumas estratégias evolucionárias foram desenvolvidos tendo em vista áreas de aplicação variadas, tais como: programação da produção na indústria de processos químicos e de manufatura, exercícios de treinamento em um laboratório naval, escoamento de produção de cerveja (Bäck et al, 1997), descarga e embarque de minério em um porto (Cruz, 2003), etc. Estes problemas são diferentes essencialmente na especificação de uma representação adequada e nos operadores utilizados.

Os problemas investigados até o momento são, em grande parte, versões simplificadas de problemas de *scheduling* do mundo real como, por exemplo, problemas de *flow shop*, problemas de *job shop*, problemas de uma máquina. E esses tipos de problema têm sido abordados com sucesso por estruturas de representação indiretas e independentes de domínio (Bäck et al, 1997). Apenas em alguns casos problemas do mundo real são abordados, como em Cruz (2003) e Almeida (2001), e normalmente são muito mais complexos pela necessidade de lidar com tipos adicionais de restrições. Motivados pelo problema de lidar com tipos diferentes de restrições específicas do domínio, alguns algoritmos evolucionários foram especificamente construídos tendo em vista o problema de *scheduling*, integrando o conhecimento específico do problema na representação e nos operadores.

As abordagens a seguir serão organizadas pelo tipo de representação utilizado. O esquema de representação pode ser direto ou indireto, como apresentado por Bagchi et al (1991). E, ainda, existem dois modos distintos de representação indireta: a dependente e a independente de domínio (Bäck et al, 1997)

3.2.1.1. Abordagens baseadas em representações indiretas independentes de domínio.

A maioria dos algoritmos evolucionários para *scheduling* usa uma representação indireta de soluções, ou seja, o algoritmo atua numa população de soluções codificadas. E, como a representação não representa diretamente uma programação, uma transição, da representação para a programação propriamente dita, deve ser feita por um decodificador antes da avaliação. O decodificador garante a viabilidade das programações. Como ele tem que procurar por informação não contida no indivíduo, sua atividade depende da quantidade de informação contida na representação – quanto mais informação contida na representação, menor a busca a ser feita pelo decodificador, e vice-versa.

Uma representação indireta independente de domínio não contém nenhuma informação auxiliar a respeito do problema em questão. O algoritmo evolucionário faz uma reprodução cega da solução codificada ao aplicar operadores convencionais. O conhecimento do domínio permanece separado, dentro do procedimento de avaliação, para o cálculo da aptidão (Bäck et al, 1997).

Representação binária. Neste modelo, a solução é representada por uma cadeia de bits e o algoritmo genético utiliza-se de operadores genéticos convencionais como, por exemplo, o *crossover* de um ponto. As diferenças nesta abordagem são encontradas em relação ao significado de cada bit. Como a maioria dos indivíduos produzidos por operações convencionais de *crossover* é composta por soluções inválidas, um algoritmo de reparo deve ser utilizado para gerar uma solução válida o mais próxima possível da solução inválida gerada. Fox & McMahon (1991), por exemplo, projetaram uma matriz lógica representando a seqüência de operações. Novos operadores foram desenvolvidos para preservar as propriedades desta seqüência.

Representação por seqüência de tarefas. Neste caso, uma lista de todas as tarefas que devem ser planejadas é representada em cada indivíduo. A ordenação da lista representa a prioridade das tarefas. Deste modo, o problema passa a ser semelhante a um problema de seqüenciamento como, por exemplo, o problema do caixeiro-viajante (*Traveling Salesman Problem* – TSP (Michalewicz, 1996)). O algoritmo genético, portanto, se encarrega de gerar novas permutações desta lista

de tarefas. Várias abordagens se utilizaram desta representação, e vários operadores e decodificadores foram também desenvolvidos para ela.

Representação por seqüência de operações. Segundo Fang et al (1993) e Morikawa (1992), a representação contém para cada operação o número da tarefa correspondente. Esta abordagem é, em todos os outros aspectos, similar à representação por seqüência descrita acima. O decodificador planeja as operações de uma mesma tarefa respeitando a relação de precedência entre elas. Por exemplo, se (3 1 3 2) é um indivíduo, então a primeira operação da tarefa 3 será planejada primeiro, então a primeira da tarefa 1, então a segunda da tarefa 3, e assim por diante.

Representação por chave aleatória (Bean, 1994). Uma programação é codificada por números aleatórios. Estes valores são usados como chaves de ordenação para decodificar a solução. Cada tarefa está associada a uma posição na representação. Uma programação é construída ao se ordenar as chaves, e as tarefas são seqüenciadas seguindo esta ordenação. Por exemplo, o indivíduo (0.45, 0.36, 0.79, 0.81) representaria a seqüência de tarefas (2-1-3-4), que está na ordem crescente dos números gerados.

3.2.1.2. Abordagens baseadas em representação indireta dependente de domínio.

Em uma representação indireta dependente de domínio, o conhecimento do problema de *scheduling* em questão é explicitamente representado nos indivíduos. Para ser possível trabalhar nesta representação, novos operadores dependentes do domínio devem ser criados. O conhecimento do domínio está contido na representação, nos operadores e na função de avaliação.

Representação por lista de preferência. A primeira abordagem de *scheduling* utilizando algoritmos genéticos foi desenvolvida por Davis (1985). É baseada em uma representação onde uma programação é especificada a partir de uma lista de preferências para cada máquina em cada intervalo de tempo. Esta lista de preferências é uma lista de tarefas, mais os elementos “em espera” e “não operando”. Ela é interpretada como uma indicação de qual tarefa a máquina prefere executar num dado momento, ou se ela deve ficar em espera, ou não operacional. O operador de *crossover* criado para essa abordagem troca listas de

preferências entre indivíduos, o operador de mutação reordena o conteúdo de uma lista de preferência, e um outro operador foi criado para inserir intervalos não operacionais para as máquinas. Esta abordagem foi utilizada e ajustada por outros autores, como Cleveland (1989) e Hou et al (1991). E, em Falkenauer & Bouffouix (1991), foi introduzido um operador de *crossover* modificado que, desta vez, operava de forma independente em cada uma das listas de preferência.

3.2.1.3. Abordagens baseadas em representação direta.

Em uma representação direta do problema, uma programação completa e viável é utilizada como indivíduo. Toda informação relevante para se descrever unicamente uma programação é incluída na representação. O algoritmo evolucionário é o único algoritmo que realiza busca, já que a informação representada engloba todo o espaço de busca. Assim, um decodificador deixa de ser necessário neste tipo de representação. Esta representação exige que novos operadores sejam definidos, já que os operadores convencionais, independentes de domínio, na maioria dos casos irão gerar muitos descendentes inválidos.

Representação por Lista de Tarefas-Máquinas-Início. Em Kanet & Sridharan (1991) uma programação é representada por uma lista de tarefas, onde cada tarefa tem uma máquina associada e um instante de início programado. Como cada tarefa tem uma e somente uma operação, cada indivíduo representa uma programação completa. Os descendentes são criados ao se selecionarem instantes de início das programações-pai e ao se ajustá-los para se formar uma programação válida.

3.2.1.4. Outras abordagens evolucionárias.

A seguir, algumas abordagens que não se enquadram em nenhuma das classificações acima.

Técnicas de aprendizado. Em Hilliard et al (1988) foram aplicados sistemas de classificação para se descobrir regras para o problema de *job shop scheduling*. As regras de programação determinam onde e qual tarefa posicionar em uma fila. O objetivo do aprendizado é aprender a ordenar filas de tarefas.

Já em Dorndorf & Pesch (1992) foi conduzida uma abordagem de aprendizado probabilístico, onde cada item em um indivíduo representa uma regra, extraída de um conjunto de regras. O item na i-ésima posição diz que um conflito na i-ésima iteração de uma programação heurística deve ser resolvido usando a i-ésima regra. O algoritmo busca a melhor seqüência de regras de decisão para selecionar operações para guiar a busca de um algoritmo de programação heurístico.

Seqüenciamento e dimensionamento de lotes. Os dois problemas, inter-relacionados, de determinar o tamanho de lotes e a seqüência de tarefas são abordados simultaneamente.

Em Lee et al (1993) os tamanhos originais dos lotes são divididos em pequenas unidades de lote. A população inicial contém permutações destes pequenos lotes. Novas permutações são geradas utilizando o operador de recombinação de adjacências (Michalewicz, 1996), também utilizado neste trabalho. Se alguns tipos de tarefa devem ser agrupados, então elas são transformadas em um único lote. A estrutura de representação evolui gradativamente.

A estratégia de evolução desenvolvida por Zimmermann (1985) usa uma representação onde cada dígito especifica um tipo de tarefa e o número de dígitos iguais consecutivos especifica o tamanho do lote. Tamanhos de lotes são mutados por simultâneas duplicações e exclusões de dígitos. Um operador de inversão altera a seqüência de produção.

Outros métodos. Uma abordagem paralela para planejamento e programação integrados (*planning and scheduling*) foi proposta por Husbands (1993) (veja também Husbands (1990, 1991)), onde populações de planos de processo, separadas, evoluem de forma independente e são combinadas por um decodificador que monta programações e retorna os valores de aptidão.

Paredis (1992, 1993) propôs um método geral para se lidar com restrições em Algoritmos Genéticos, e o aplicou em um problema de *scheduling*. Os membros da população representam estados de busca (programações parciais), a partir dos quais soluções podem ser encontradas com uma busca baseada em restrições.

3.2.2.
Alternativas à computação evolucionária

Problemas de *scheduling* têm sido investigados intensivamente em áreas de pesquisa operacional e inteligência artificial. Tradicionalmente, esta pesquisa tem se concentrado nos métodos para obtenção de soluções ótimas para problemas simplificados, utilizando, por exemplo, programação inteira ou algoritmos de *branch-and-bound*. Para achar uma solução ótima, várias simplificações são colocadas no problema, como redução do número de recursos ou de tarefas, a redução do horizonte de programação, etc., que tornam a aplicação do modelo em problemas mais complexos muito mais difícil e às vezes até impossível. Devido às dificuldades deste tipo de problema, em várias situações do mundo real, o objetivo é uma programação viável em um tempo razoável, que não necessariamente precisa ser uma programação ótima, mas que seja, é claro, tão boa quanto for possível (Bäck et al, 1997).

Algoritmos heurísticos têm sido projetados para gerar programações de forma eficiente. Mas, normalmente, a qualidade das programações não é satisfatória. Métodos de pesquisa operacional são abordados em mais detalhes por Blazewicz et al (1994).

Alguns *softwares* comerciais se propõem a simular horizontes de programação auxiliando o programador a avaliar sua programação e refinar seus modelos. Outros simuladores incorporam regras heurísticas para geração automática de parte das atividades, com base em regras informadas por um especialista, neste caso o programador de produção. A vantagem destes métodos é que eles incorporam diretamente as informações do especialista, e fornecem uma resposta rápida para o programador, mas têm a desvantagem óbvia de ser apenas um auxílio ao trabalho, que permanece manual, de programar as atividades e criar uma programação viável.

Nos últimos anos, pôde-se observar um interesse crescente no uso de inteligência computacional na área de *scheduling.* Vários sistemas baseados em conhecimento surgiram usando paradigmas diferentes, como por exemplo, abordagens baseadas em regras, lógica nebulosa, ou sistemas multiagentes. Além destes, técnicas baseadas em conhecimento foram aplicados em problemas de *scheduling* reativo, onde se deseja adaptar uma programação existente pela

ocorrência de certos eventos que a invalidam. Uma visão geral da pesquisa de técnicas baseadas em conhecimento para problemas de *scheduling* pode ser encontrada em Smith (1992) e Zweben (1994).

A maior vantagem da computação evolucionária é que a busca permanece viável em termos de tempo de processamento e recursos computacionais, mesmo para problemas maiores e mais realistas. Isto em contraste com algoritmos que garantem a obtenção do ótimo, mas são aplicáveis somente a problemas muito restritos. Então, a computação evolucionária tem o potencial para gerar soluções de alta qualidade com esforço computacional aceitável para vários problemas de *scheduling* do mundo real (Bäck et al, 1997).

Outra grande vantagem dos algoritmos evolucionários é a possibilidade de interromper a otimização a qualquer momento, de forma que a melhor programação até o momento esteja sempre disponível para ser usada, se necessário.

Uma desvantagem significativa das abordagens evolucionárias, principalmente em relação aos sistemas baseados em conhecimento, é a impossibilidade de um especialista humano verificar a consistência dos critérios utilizados no processo de solução do problema. Por exemplo, se o sistema gerasse um conjunto de regras que seria utilizado para construir uma programação, o conjunto escolhido poderia ser analisado e validado por um especialista. Esta é uma grande desvantagem, já que a tomada de decisão, em última instância, é sempre de um especialista que é responsável por tais decisões.

A computação evolucionária não pode substituir todos os outros métodos desenvolvidos para este problema, mas tem o potencial de se tornar uma parte complementar em sistemas híbridos, de modo a aproveitar as vantagens de cada abordagem. As técnicas evolucionárias parecem ser especialmente indicadas para problemas de programação onde uma solução de alta qualidade deve ser gerada em tempo limitado sem a necessidade de se garantir a solução ótima global.

3.3. Co-evolução.

Existem duas razões principais pelas quais algoritmos evolucionários convencionais não são totalmente adequados para resolver o tipo de problema

considerado nesta dissertação. Em primeiro lugar, os algoritmos genéticos convencionais impedem, em 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 & DeJong, 2000).

O *Oxford Dictionary of Natural History* (Allaby, 1985) apresenta a seguinte definição de co-evolução:

> **Co-evolução**: Evolução complementar de espécies intimamente associadas. As adaptações inter-relacionadas existentes nas plantas floríferas e seus insetos polinizadores são exemplos claros de co-evolução. Em um senso mais amplo, o relacionamento predador-presa também envolve co-evolução, com um avanço evolucionário no predador estimulando uma resposta evolucionária na presa.

De acordo com o apresentado acima, co-evolução envolve íntima interação entre espécies.

Em sistemas do tipo “predador-presa”, por exemplo, existe uma interação inversa de aptidões: o sucesso de uma das partes é sentido pela outra como uma falha que deve ser respondida para que as suas chances de sobrevivências sejam mantidas. Existe uma forte pressão evolucionária para as presas se defenderem melhor e, em contrapartida, para os predadores desenvolverem melhores estratégias de ataque. Isto tipicamente resulta em uma disputa onde a complexidade de predador e presa aumenta gradativamente. Este tipo é chamado **Co-evolução competitiva**.

A co-evolução competitiva geralmente emprega uma forma de avaliar indivíduos de acordo com medidas relativas de aptidão, ou seja, o quão mais adaptado um determinado indivíduo é, em relação aos seus competidores. Esta medida é chamada de aptidão competitiva, ou *competitive fitness*. Mais detalhes sobre esta abordagem podem ser vistos em (Bäck et al, 1997).

3.3.1.
Co-evolução Cooperativa.

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 têm uma relação apenas de cooperação.

A co-evolução cooperativa foi inspirada no conceito de simbiose. Na co-evolução cooperativa, como na competitiva, duas ou mais espécies interagem e colaboram para a evolução uma da outra. Mas, ao contrário da evolução competitiva, onde o aumento da aptidão de um indivíduo ocasiona uma queda da aptidão relativa do outro, aqui cada espécie evolui em seu nicho ou especialidade, cooperando para que a união delas faça com que o ecossistema que elas representam evolua como um todo.

O modelo co-evolucionário cooperativo básico é mostrado na figura 5. Apesar de serem mostradas apenas três espécies neste modelo, o mesmo pode ser usado para n espécies diferentes. Cada espécie evolui em sua própria população e se adapta ao ambiente através de repetidas aplicações do algoritmo evolucionário.

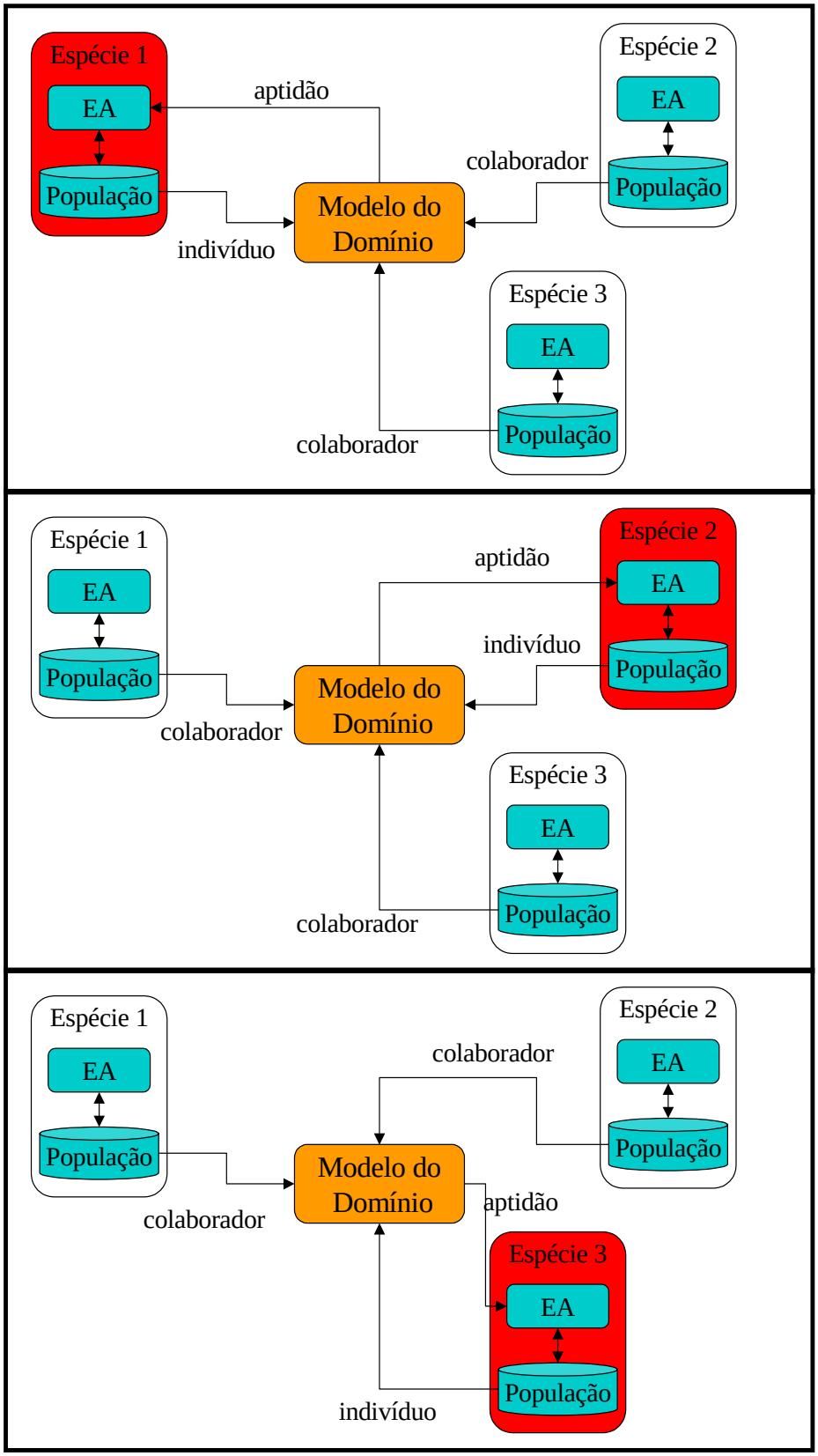

Figura 5-Arquitetura do Modelo Co-evolucionário.

Para se calcular a aptidão dos indivíduos de uma determinada espécie, deve-se submetê-los 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).

Existem vários métodos possíveis para se escolher colaboradores. Em alguns casos é apropriado simplesmente deixar o melhor indivíduo de cada espécie ser o colaborador. Em outros casos, esta pode ser uma estratégia muito gananciosa, e outras alternativas devem ser estudadas. Por exemplo, uma amostra de indivíduos de cada espécie pode ser aleatoriamente escolhida, ou uma forma de seleção mais ecológica que selecione indivíduos baseada em sua aptidão, de forma não determinística, pode ser utilizada (Potter & DeJong, 2000).

A seguir, apresenta-se uma pequena análise sobre os métodos de seleção de colaboradores encontrados na literatura.

3.3.1.1. Métodos de seleção de colaboradores.

Quando aplicamos a co-evolução cooperativa a um problema, a abordagem mais comum é decompor o problema em subcomponentes e associar a cada um deles uma população. A evolução de cada uma das populações ocorre de forma independente, exceto na avaliação. Como um indivíduo de uma dada espécie representa apenas uma parte da solução, indivíduos de outras espécies devem ser selecionados, segundo algum critério, e associados a este para se poder avaliar a solução como um todo. Isto é chamado de colaboração.

A cada geração, todos os indivíduos de uma das populações são avaliados selecionando-se um conjunto de colaboradores das outras populações para formar soluções completas. Em seguida, o algoritmo passa para a próxima população, que por sua vez selecionará indivíduos das outras populações, e assim por diante.

A avaliação de indivíduos num sistema co-evolucionário pode ser feita de várias formas. Alguns algoritmos competitivos efetuam avaliações combinando cada indivíduo de uma população com cada indivíduo de outra (Hillis, 1991). Ou, no caso de modelos de avaliação competitiva com apenas uma população, é comum a avaliação comparativa dois a dois (Axelrod, 1989). Estas abordagens tendem a ser muito custosas em modelos com várias populações, já que o número de avaliações cresce exponencialmente com o número de espécies. Algumas

abordagens menos dispendiosas, como torneios de eliminação única, apresentado por Angeline & Pollack (1993), também foram utilizadas.

Em Wiegand et al (2001) há uma identificação de três atributos determinantes para a escolha de um modelo de colaboração:

1. Pressão de seleção de colaboradores (*collaborator selection pressure*): O peso que a avaliação de um determinado indivíduo tem na seleção deste como colaborador;
2. Tamanho do grupo de colaboração (*collaboration pool size*): O número de colaboradores por espécie a ser usado em uma avaliação;
3. Associação de crédito de colaboração (*collaboration credit assignment*): Método para associar um valor de aptidão a um colaborador, dadas várias avaliações da função objetivo utilizando este indivíduo;

Como opções para a associação de crédito de colaboração temos três métodos (Wiegand et al, 2001): Otimista, Pessimista e Médio. No otimista, é associado ao valor de aptidão do indivíduo o valor da aptidão da melhor colaboração feita por ele. No pessimista a avaliação na pior colaboração é utilizada, enquanto que no médio é utilizada a média das avaliações em todas as suas colaborações. Apesar de o otimista ser apresentado como muito ganancioso, e os outros dois parecerem mais seguros que ele, ao premiarem um indivíduo não apenas por sua melhor colaboração, todos os testes indicam um melhor desempenho do algoritmo quando se utiliza o método otimista.

Com relação à 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.

Quanto à escolha do tamanho do grupo de colaboração que, conforme os testes realizados por Wiegand et al (2001), consiste no fator mais importante para

o desempenho dos algoritmos co-evolucionários cooperativos, se resume a decidir quantos elementos das outras espécies irão participar na colaboração. Em geral, aumentar o número de colaboradores melhora o desempenho do algoritmo, mas por outro lado aumenta consideravelmente o custo computacional.

3.3.1.2. Exemplos de Uso.

Vários problemas foram solucionados utilizando esta arquitetura. Em Potter & DeJong (1994), a arquitetura foi utilizada para maximizar uma função $f(\vec{x})$ de *n* variáveis independentes, onde o problema foi decomposto manualmente em *n* espécies e cada uma delas foi associada a uma variável independente. Para avaliar um indivíduo de uma espécie, os melhores indivíduos das outras espécies eram selecionados e combinados com ele para se formar um vetor completo de variáveis, que então era aplicado na função a ser otimizada.

Em Potter et al (1995), um controle de um robô simulado baseado em regras foi desenvolvido usando esta arquitetura. Existiam duas espécies, cada uma delas com um conjunto de regras de uma classe de comportamentos. As espécies eram inicializadas com algum conhecimento na sua área de atuação para estimular a evolução por uma trajetória desejada. Para se avaliar um indivíduo de uma espécie, o melhor conjunto de regras da outra espécie era selecionado, combinado com o conjunto em avaliação, e o conjunto resultante era usado para controlar o robô. Um indivíduo era premiado baseado na qualidade de sua colaboração.

Cruz (2003) propôs uma arquitetura para problemas de planejamento aplicado a um problema de descarga de minério em portos. O modelo contava com duas espécies, uma encarregada da alocação das tarefas no tempo, enquanto a outra decidia em quais recursos alocar as tarefas. A avaliação era feita com base no custo por atraso na entrega dos carregamentos, ou custo de *Demurrage*. Maiores detalhes deste modelo serão apresentados no próximo capítulo, já que a representação das espécies nesta dissertação é similar à apresentada nesta arquitetura.