CAPÍTULO 2

ALGORITMOS GENÉTICOS

De maneira geral, pode-se apresentar uma taxionomia dos sistemas computacionais inspirados na natureza englobando a inteligência computacional, vida artificial, sistemas complexos e geometria fractal. A computação evolutiva (em conjunto com sistemas *fuzzy*, redes neurais artificiais e agentes autônomos) está posicionada como um subitem da inteligência computacional.

A computação evolutiva pode ser classificada como sendo uma estratégia de solução concebida para resolver problemas genéricos e operar em espaços não-lineares e não-estacionários. Sabe-se, também, que não garante eficiência total na obtenção da solução. No entanto, geralmente alcança uma boa aproximação para a solução ótima e em um tempo que aumenta a uma taxa menor que exponencial com o crescimento do tamanho do problema.

Independentemente da aplicação, estratégias baseadas em computação evolutiva devem ser consideradas somente quando abordagens clássicas ou dedicadas não existem, não são aplicáveis ou falham quando empregadas. Isso leva à conclusão de que uma estratégia baseada em computação evolutiva não pode ser escolhida como a primeira abordagem na busca pela solução de um problema.

Neste capítulo, são apresentados os principais aspectos relacionados à computação evolutiva, mais especificamente algoritmos genéticos. IYODA (2000) apresenta uma revisão e formalização de conceitos de computação evolutiva, sendo que o material por ele elaborado será parcialmente reproduzido e complementado a seguir, para efeito de completitude do presente trabalho. Este capítulo está estruturado da seguinte forma: na Seção 2.1, são resumidamente apresentadas as idéias evolucionistas de Charles Darwin; na Seção 2.2, apresentam-se algumas idéias referentes à teoria da evolução natural; na Seção 2.3, enfoca-se a computação evolutiva e descrevem-se sucintamente as principais abordagens presentes na literatura; na Seção 2.4, os algoritmos genéticos são mais

detalhadamente descritos e são apresentadas, na Seção 2.5, conclusões e algumas perspectivas na área de computação evolutiva.

2.1 As Idéias Evolucionistas

De acordo com AMABIS & MARTHO (1997), as idéias evolucionistas de Charles Darwin podem ser resumidamente enunciadas em três conclusões, sendo apoiadas em quatro observações:

> Primeira Observação: as populações naturais de todas as espécies tendem a crescer rapidamente, pois o potencial reprodutivo dos seres vivos é muito grande[1]. Isso pode ser verificado, por exemplo, quando se criam determinadas espécies em cativeiro: ao garantir condições ambientais favoráveis ao desenvolvimento, sempre se observa a elevada capacidade reprodutiva inerente às populações biológicas.
>
> Segunda Observação: o tamanho das populações naturais, a despeito de seu enorme potencial de crescimento, se mantém relativamente constante ao longo do tempo, devido à limitação de recursos do ambiente.
>
> *Primeira Conclusão: em cada geração, morre um grande número de indivíduos, muitos deles sem deixar nenhum descendente.*
>
> Terceira Observação: os indivíduos de uma população diferem quanto a diversas características, inclusive aquelas que influem na capacidade de explorar, com sucesso, os recursos do ambiente e de deixar descendentes.

[1] O potencial reprodutivo dos seres vivos deve ser maior que a taxa de mortalidade.

> *Segunda Conclusão: os indivíduos que sobrevivem e se reproduzem a cada geração, são, provavelmente, os que apresentam mais características relacionadas com a adaptação às condições ambientais. Esta conclusão resume o conceito darwinista de seleção natural ou sobrevivência dos mais aptos.*
>
> Quarta Observação: grande parte das características apresentadas por indivíduos de uma dada geração é herdada dos respectivos pais.
>
> *Terceira Conclusão: uma vez que, a cada geração, sobrevivem os mais aptos, eles tendem a transmitir aos descendentes características relacionadas a essa maior aptidão para sobreviver, isto é, para se adaptar. Em outras palavras, a seleção natural favorece, ao longo de gerações sucessivas, a permanência e o aprimoramento de características relacionadas à adaptação.*

2.2 Evolução Natural

A evolução natural é o processo que guia o surgimento de estruturas orgânicas complexas e adaptadas ao ambiente (BÄCK *et al.*, 1997a). De forma sucinta, e com grau elevado de simplificação, BÄCK *et al.* (1997a) descreve a evolução como o resultado da influência mútua entre a criação de informação genética nova, sua avaliação e seleção. Um indivíduo de uma população é afetado por outros indivíduos da população (por exemplo, pela competição por alimento, predadores, acasalamento, etc.) e também pelo ambiente em que vive (por exemplo, pela oferta de comida, clima, etc.). Quanto maior a adaptação de um indivíduo a tais condições, maior a chance do indivíduo sobreviver por mais tempo e gerar uma prole, que por sua vez herda a informação genética dos pais (possivelmente com algum erro de cópia e sujeita à recombinação das informações fornecidas pelos pais). Ao longo do processo de evolução, vai ocorrendo na população uma penetração majoritária de informação genética oriunda de indivíduos com adaptação acima da média. Além disso, o

caráter não-determinístico da reprodução leva a uma produção permanente de informação genética nova e, portanto, à criação de descendentes diferenciados (para maiores detalhes veja ATMAR (1994) e FOGEL (1999)).

Indivíduos e espécies podem ser vistos como uma dualidade entre seu código genético (genótipo) e suas características comportamentais, fisiológicas e morfológicas (fenótipo) (FOGEL, 1994). Em sistemas evoluídos naturalmente, não existe uma relação biunívoca entre um gene (elemento do genótipo) e uma característica (elemento do fenótipo): um único gene pode afetar diversos traços fenotípicos simultaneamente (pleiotropia) e uma única característica fenotípica pode ser determinada pela interação de vários genes (poligenia). Os efeitos de pleiotropia e poligenia geralmente tornam os resultados de variações genéticas imprevisíveis. Sistemas naturais em evolução são fortemente pleiotrópicos e altamente poligênicos (HARTL & CLARK, 1989). O mesmo não ocorre em sistemas artificiais, onde uma das principais preocupações é com o custo computacional do sistema, de modo que geralmente existe uma relação de um-para-um entre genótipo e fenótipo.

Segundo ATMAR (1994) e FOGEL (1999), o processo de evolução pode ser formalizado do seguinte modo: considere dois espaços de estados distintos – um espaço de estados genotípico (codificação) G e um espaço fenotípico (comportamental) F. Considere também um alfabeto de entrada composto de símbolos provenientes do ambiente I.

O processo de evolução de uma população entre duas gerações consecutivas encontra-se esquematizado na Figura 2.1.

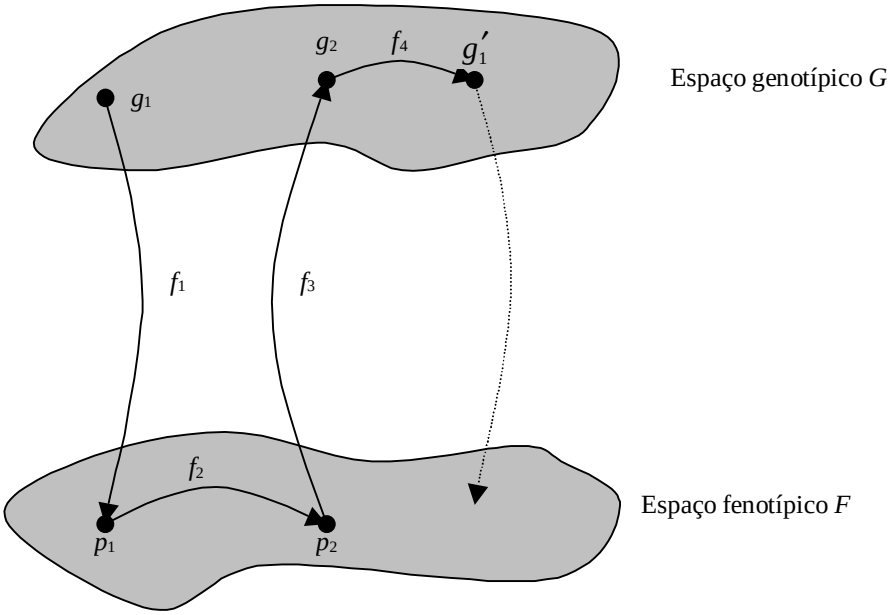


Figura 2.1: Evolução de uma população durante uma geração

Existem quatro mapeamentos atuando neste processo:

$$f_1 : I \times G \rightarrow F,$$
$$f_2 : F \rightarrow F,$$
$$f_3 : F \rightarrow G,$$
$$f_4 : G \rightarrow G.$$

O mapeamento f_1, denominado *epigênese*, mapeia elementos $g_1 \in G$ em uma coleção particular de fenótipos p_1 do espaço fenotípico F, cujo desenvolvimento é modificado por seu ambiente, um conjunto de símbolos $\{i_1, \ldots, i_k\} \in I$. Este mapeamento é inerentemente de muitos-para-um, pois existe uma infinidade de genótipos que podem resultar em um mesmo fenótipo: elementos de um conjunto infinito de códigos não expressos (não participantes na produção do fenótipo) podem existir em g_1 (ATMAR, 1994).

O mapeamento f_2, *seleção*, mapeia fenótipos p_1 em p_2. Este mapeamento descreve os processos de seleção, imigração e emigração de indivíduos dentro da população local. Como a seleção natural opera apenas nas expressões fenotípicas do genótipo, o código g_1 não está diretamente envolvido no mapeamento f_2. ATMAR (1994) enfatiza que a seleção atua apenas no sentido de eliminar as variantes comportamentais menos apropriadas do

inevitável excesso da população, já que assume-se aqui que os recursos provenientes do ambiente são limitados, exigindo a competição pela sobrevivência. Neste processo de competição, a seleção nunca opera sobre uma característica simples isoladamente do conjunto comportamental.

O mapeamento f_3, *representação* (ATMAR, 1994) ou *sobrevivência genotípica* (FOGEL, 1999), descreve os efeitos dos processos de seleção e migração em G.

O mapeamento f_4, *mutação e recombinação*, mapeia códigos $g_2 \in G$ em $g_1' \in G$. Este mapeamento descreve as "regras" de mutação e recombinação, abrangendo todas as alterações genéticas. A mutação é um erro de cópia no processo de transmissão do código genético dos pais para a sua prole. Em um universo com diferencial de entropia positivo, erros de replicação são inevitáveis e a otimização evolutiva torna-se fatal em qualquer população que se reproduz em uma arena limitada (ATMAR, 1994).

Com a criação da nova população de genótipos g_1', uma geração está completa, de maneira que a adaptação evolutiva ocorre em sucessivas iterações destes mapeamentos.

O biólogo Sewell Wright propôs, em 1931, o conceito de superfície de adaptação para descrever o nível de adaptação de indivíduos e espécies ao ambiente (FOGEL, 1999). Uma população de genótipos é mapeada em seus respectivos fenótipos que, por sua vez, são mapeados nos seus valores de adaptação. Cada pico (máximo local) da superfície de adaptação corresponde a uma coleção de fenótipos otimizada e, portanto, a um ou mais conjuntos de genótipos otimizados. A evolução é um processo que conduz, de forma probabilística, populações em direção a picos da superfície, enquanto que a seleção elimina variantes fenotípicas menos apropriadas. Outros pesquisadores propõem uma visão invertida da superfície de adaptação: populações avançam descendo picos da superfície de adaptação até que um ponto de mínimo seja encontrado.

Qualquer que seja o ponto de vista, a evolução é sempre um processo de otimização. Dadas as condições iniciais, restrições ambientais e parâmetros evolutivos, a seleção produzirá fenótipos tão próximos do ótimo quanto possível. Observa-se, no entanto, que em sistemas biológicos reais, não existem superfícies de adaptação estáticas, já que o ambiente está em constante mudança, fazendo com que populações estejam em permanente evolução em direção a novos pontos de ótimo. Neste caso, assumindo que as mudanças ambientais

são significativas, embora graduais, a taxa evolutiva deve ser suficientemente elevada para acompanhar as mudanças ambientais.

2.3 Computação Evolutiva

A computação evolutiva é formada por algoritmos inspirados na teoria de evolução natural de Darwin, podendo desempenhar os seguintes papéis básicos:

- ferramenta adaptativa para solução de problemas;
- modelo computacional de processos evolutivos naturais.

Os sistemas inspirados em computação evolutiva mantêm uma população de soluções potenciais, aplicando processos de seleção baseados na adaptação de um indivíduo e, também, empregando outros operadores genéticos. Diversas abordagens para sistemas baseados em evolução foram propostas, sendo que as principais diferenças entre elas dizem respeito aos operadores genéticos utilizados (uma discussão sobre operadores genéticos será apresentada mais adiante neste capítulo). As principais abordagens propostas na literatura são:

- algoritmos genéticos;
- estratégias evolutivas;
- programação evolutiva.

Os a*lgoritmos genéticos* foram introduzidos por J. Holland em 1975 (HOLLAND, 1992) com o objetivo de formalizar matematicamente e explicar rigorosamente processos de adaptação em sistemas naturais e desenvolver sistemas artificiais (simulados em computador) que retenham os mecanismos originais encontrados em sistemas naturais. Os algoritmos genéticos empregam os operadores de *crossover* e mutação (serão apresentados mais adiante neste capítulo). Os algoritmos genéticos têm sido intensamente aplicados em problemas de otimização, apesar de não ter sido este o propósito original que levou ao seu desenvolvimento.

Uma extensão dos algoritmos genéticos, denominada *programação genética*, foi introduzida por KOZA (1992)[2], em que o código genético corresponde a uma árvore de atributos, em lugar da codificação em lista de atributos comumente empregada no caso dos algoritmos genéticos. A programação genética teve por objetivo inicial evoluir programas de computador usando os princípios da evolução natural. Atualmente, a programação genética tem sido aplicada a uma grande variedade de problemas como, por exemplo, na síntese de circuitos elétricos analógicos (KOZA *et al.*, 1997) e na definição de arquiteturas de redes neurais artificiais (GRUAU, 1994).

Estratégias evolutivas (RECHENBERG, 1973; SCHWEFEL, 1995) foram inicialmente propostas com o objetivo de solucionar problemas de otimização de parâmetros, tanto discretos como contínuos, empregando apenas o operador de mutação.

A *programação evolutiva*, introduzida por FOGEL *et al.* (1966), foi originalmente proposta como uma técnica para criar inteligência artificial pela evolução de máquinas de estado finito (empregando, também, apenas mutação). Recentemente, tem sido aplicada a problemas de otimização, sendo, neste caso, virtualmente equivalente às estratégias evolutivas. Atualmente, existem apenas pequenas diferenças no que diz respeito aos procedimentos de seleção e codificação de indivíduos presentes nestas duas abordagens (FOGEL, 1994).

Apesar das abordagens acima citadas terem sido desenvolvidas de forma independente, seus algoritmos possuem uma estrutura comum. O termo *algoritmo evolutivo* será utilizado para denominar todos os sistemas baseados em evolução e a sua estrutura é apresentada na Figura 2.2 (MICHALEWICZ, 1996).

[2] J. R. Koza detém uma patente sobre programação genética.

Procedimento programa evolutivo
início
$t \leftarrow 0$
inicializar *P(t)*
avaliar *P(t)*
enquanto não (condição de parada) **faça**
início
$t \leftarrow t+1$
selecionar *P(t)* a partir de *P(t-1)*
alterar *P(t)*
avaliar *P(t)*
fim
fim

Figura 2.2: Estrutura proposta para um algoritmo evolutivo.

Um algoritmo evolutivo mantém uma população de indivíduos $P(t) = \{x_1^t, \ldots, x_n^t\}$ na iteração (geração) *t*. Cada indivíduo representa um candidato à solução do problema em questão e, em qualquer implementação computacional, assume a forma de alguma estrutura de dados *S*. Cada solução x_i^t é avaliada e produz alguma medida de adaptação ou *fitness*. Assim, uma nova população (iteração $t + 1$) é formada pela seleção dos indivíduos mais adaptados ao meio (passo de seleção). Alguns indivíduos da população são submetidos a transformações (passo de alteração) por meio de operadores genéticos para formar novas soluções. Existem transformações unárias m_i (mutação) que criam novos indivíduos por meio de pequenas mudanças em um indivíduo ($m_i : S \rightarrow S$) e transformações de ordem superior c_j (*crossover*), criando novos elementos pela combinação de dois ou mais indivíduos ($c_j : S \times \ldots \times S \rightarrow S$). Uma condição de parada deve ser definida (por exemplo, um determinado número de gerações) e o indivíduo da população que apresentar o melhor desempenho será tomado como a solução do problema.

As quatro abordagens evolutivas citadas nesta seção diferem em diversos aspectos: nas estruturas de dados utilizadas para codificar um indivíduo, nos operadores genéticos empregados, nos métodos para criar a população inicial, nos métodos para selecionar indivíduos para a geração seguinte, etc. Entretanto, compartilham do mesmo princípio, ou

seja, uma população sofre algumas transformações e durante a evolução os seus indivíduos competem pela sobrevivência.

2.4 Algoritmos Genéticos

Os algoritmos genéticos empregam uma terminologia originada da teoria da evolução natural e da genética. Um indivíduo da população é representado por um único *cromossomo*, contendo a *codificação* (genótipo) de um candidato à solução do problema (fenótipo). Um cromossomo é usualmente implementado na forma de um vetor (lista de atributos), em que cada componente é conhecido como *gene*. Os possíveis valores que um determinado gene pode assumir são denominados *alelos*.

O processo de evolução executado por um algoritmo genético corresponde a um processo de busca em um espaço de soluções potenciais para alcançar o objetivo proposto. Como enfatiza MICHALEWICZ (1996), essa busca requer um equilíbrio entre dois objetivos aparentemente conflitantes: o aproveitamento das melhores soluções e a exploração do espaço de busca (explotação $\times$ exploração). Métodos de otimização do tipo *hill-climbing* são exemplos de algoritmos que aproveitam a melhor solução na busca de possíveis aprimoramentos. Por outro lado, esses métodos ignoram a exploração do espaço de busca.

Exemplos típicos de métodos que exploram o espaço de busca ignorando o aproveitamento de regiões promissoras do espaço são os que fazem busca aleatória ou não-direcionada. Algoritmos genéticos constituem uma classe de métodos de busca de propósito geral que apresentam um balanço notável entre aproveitamento de melhores soluções e exploração do espaço de busca.

Os algoritmos genéticos pertencem à classe dos algoritmos probabilísticos, mas eles não são métodos de busca puramente aleatórios, pois combinam elementos de procura direcionada e estocástica. Outra propriedade importante dos algoritmos genéticos (assim como de todos os algoritmos evolutivos) é a manutenção de uma população de soluções candidatas enquanto que métodos alternativos, como *simulated annealing* (AARTS & KORST, 1989) e busca *tabu* (GLOVER & LAGUNA, 1997), processam um único ponto no espaço de busca a cada instante.

O processo de busca realizado pelos algoritmos genéticos é multi-direcional, pela preservação de múltiplas soluções candidatas, encorajando a troca de informação entre elas.

A cada geração, soluções relativamente “boas” se reproduzem, enquanto que soluções relativamente “ruins” são eliminadas. Para fazer a distinção entre diferentes soluções é empregada uma função-objetivo (de avaliação ou de adaptabilidade) que simula o papel da pressão exercida pelo ambiente sobre o indivíduo.

A estrutura de um algoritmo genético é a mesma do algoritmo evolutivo apresentado na Figura 2.2. MICHALEWICZ (1996) descreveu um algoritmo genético da seguinte maneira: durante a iteração *t*, um algoritmo genético mantém uma população de soluções potenciais (cromossomos, vetores), $P(t) = \{\mathbf{x}_1^t, \ldots, \mathbf{x}_n^t\}$. Cada solução $\mathbf{x}_i^t$ é avaliada e produz uma medida de sua adaptação ou *fitness*. Uma nova população (iteração $t + 1$) é então formada privilegiando a participação dos indivíduos mais adaptados. Alguns membros da nova população passam por alterações por meio de *crossover* e mutação para formar novas soluções potenciais. Esse processo se repete até que um número pré-determinado de iterações seja atingido ou até que o nível de adaptação esperado seja alcançado.

Um algoritmo genético para um problema particular deve ter os seguintes componentes:

- uma representação genética para soluções candidatas ou potenciais (processo de codificação);
- uma maneira de criar uma população inicial de soluções candidatas ou potenciais;
- uma função de avaliação que faz o papel da pressão ambiental, classificando as soluções em termos de sua adaptação ao ambiente (representa a sua capacidade de resolver o problema);
- operadores genéticos;
- valores para os diversos parâmetros usados pelo algoritmo genético (tamanho da população, probabilidades de aplicação dos operadores genéticos, etc.).

2.4.1 Codificação de Indivíduos

Cada indivíduo de uma população representa um candidato em potencial à solução do problema em questão. No algoritmo genético clássico, proposto por HOLLAND (1992), as soluções candidatas são codificadas em arranjos binários de tamanho fixo. A motivação para o uso de codificação binária vem da teoria dos esquemas (*schemata theory*). HOLLAND (1992) argumenta que seria benéfico para o desempenho do algoritmo maximizar o paralelismo implícito inerente ao algoritmo genético e prova que um alfabeto binário maximiza o paralelismo implícito.

Entretanto, em diversas aplicações práticas, a utilização de codificação binária leva a um desempenho insatisfatório. Em problemas de otimização numérica com parâmetros reais, algoritmos genéticos com representação em ponto flutuante freqüentemente apresentam desempenho superior à codificação binária. MICHALEWICZ (1996) argumenta que a representação binária apresenta desempenho pobre quando aplicada a problemas numéricos com alta dimensionalidade e onde alta precisão é requerida. Suponha, por exemplo, que exista um problema com 100 variáveis com domínio no intervalo [–500, 500] e que serão necessários 6 dígitos de precisão após a casa decimal, empregando representação binária em ponto fixo. Neste caso, seria necessário um cromossomo de comprimento 3000 e um espaço de busca com aproximadamente 10^{1000} candidatos à solução, fazendo com que para este tipo de problema o algoritmo genético clássico apresente desempenho insatisfatório. MICHALEWICZ (1996) mostra simulações computacionais comparando o desempenho de algoritmos genéticos com codificação binária e com ponto flutuante, aplicados a um problema de controle com as características descritas acima. Os resultados apresentados indicam uma clara superioridade da codificação em ponto flutuante.

A argumentação de MICHALEWICZ (1996), de que o desempenho de um algoritmo genético com codificação binária é pobre quando o espaço de busca é de dimensão elevada, não é universalmente aceita na literatura que trata de algoritmos genéticos. FOGEL (1994) argumenta que o espaço de busca por si só (sem levar em conta a escolha da representação) não determina a eficiência do algoritmo genético. Espaços de busca de dimensão elevada podem às vezes ser explorados eficientemente, enquanto que espaços de busca de dimensão reduzida podem apresentar dificuldades significativas. FOGEL (1994), entretanto, concorda

que a maximização do paralelismo implícito, associada à codificação binária, nem sempre produz um desempenho ótimo. Fica claro, portanto, que a codificação é uma das etapas mais críticas na definição de um algoritmo genético e sua definição inadequada pode levar a problemas de convergência prematura. A estrutura de um cromossomo deve representar uma solução como um todo e, ainda, ser a mais simples possível. Em problemas de otimização com restrições, a codificação adotada pode fazer com que indivíduos modificados por *crossover*/mutação sejam inválidos. Nestes casos, cuidados especiais devem ser tomados na definição da codificação e/ou dos operadores.

Para evitar essa situação no desenvolvimento deste trabalho, optou-se pela elaboração de uma configuração genética compacta (envolve apenas um número reduzido de genes) aliada a um algoritmo de expansão de código que garante a factibilidade do indivíduo associado ao código expandido, relacionado ao genótipo completo do candidato (veja capítulos 5 e 6 para maiores detalhes).

Como o código genético compacto pertence a um espaço em que não há infactibilidade, os operadores genéticos sempre irão produzir descendentes factíveis (Figura 2.3).

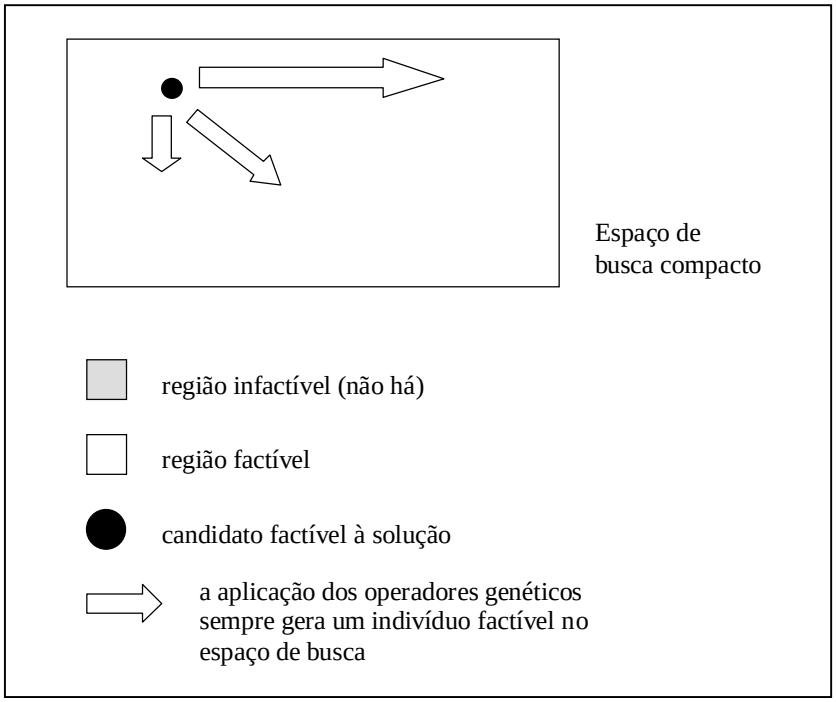


Figura 2.3: Produção de descendentes factíveis após a aplicação dos operadores genéticos em um espaço de busca compacto.

Se os operadores genéticos forem aplicados sobre o código genético pertencente ao espaço de busca expandido, haveria grande chance de que o descendente produzido fosse levado para uma região infactível do espaço de busca. Dessa maneira, seria necessário trazê-lo para uma região de factibilidade, o que implicaria na utilização de mais recursos computacionais e na necessidade de se produzir um algoritmo de factibilização (Figura 2.4).

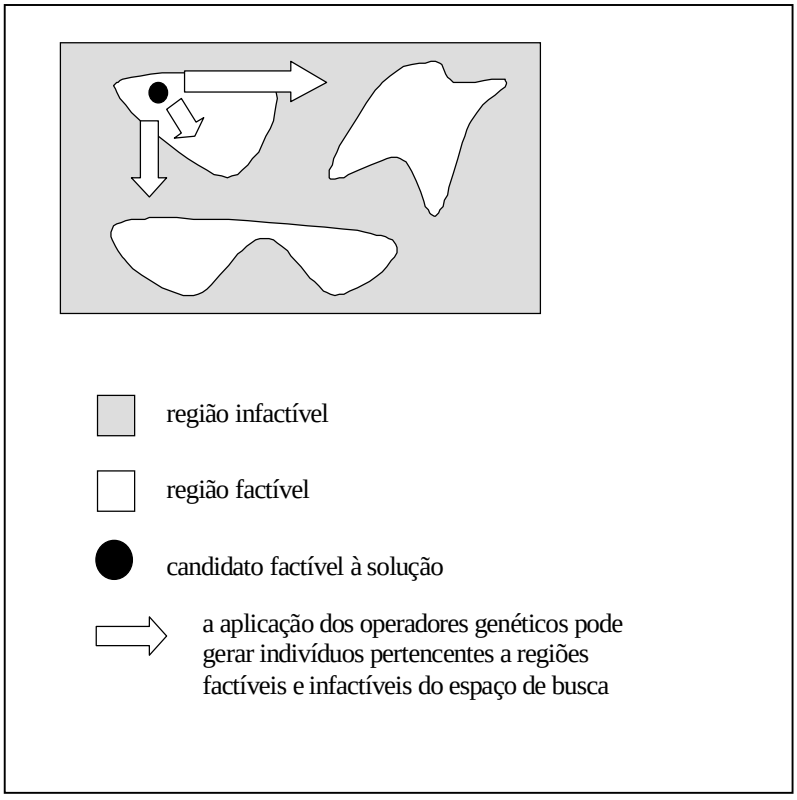


Figura 2.4: Produção de descendentes após a aplicação dos operadores genéticos em um espaço de busca expandido.

Por outro lado, o processo de compactação corresponde a uma transformação topológica no espaço genotípico, garantindo, assim, o seu fechamento em relação aos operadores genéticos.

2.4.2 Formação da População Inicial

O método mais comum utilizado na criação da população é a inicialização aleatória dos indivíduos. Se algum conhecimento prévio a respeito do problema estiver disponível, poderá ser utilizado na inicialização da população. Por exemplo, se é sabido que a solução final (assumindo codificação binária) vai apresentar mais 0's do que 1's, então esta informação pode ser utilizada, mesmo que não se saiba exatamente a proporção. Já em problemas com restrições, deve-se tomar cuidado para não gerar indivíduos inválidos na etapa de inicialização.

2.4.3 Operadores Genéticos

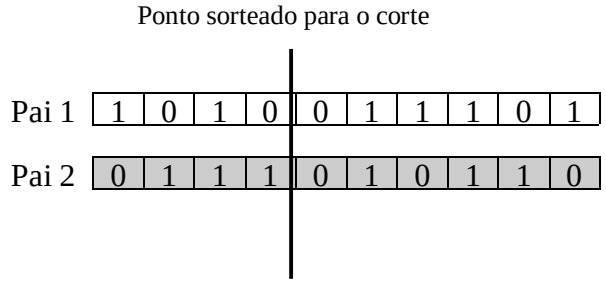
Os operadores genéticos mais freqüentemente utilizados são o *crossover* e a mutação. Nesta seção, são apresentados os principais aspectos relacionados a estes operadores.

O Operador de Crossover

O operador de *crossover* ou recombinação cria novos indivíduos por intermédio da combinação de dois ou mais indivíduos. A idéia intuitiva por trás do operador de *crossover* é a troca de informação entre diferentes soluções candidatas. No algoritmo genético clássico, é atribuída uma probabilidade de *crossover* fixa aos indivíduos da população.

O operador de *crossover* mais comumente empregado é o *crossover* de um ponto. Para a aplicação desse operador, são selecionados dois indivíduos (pais) e a partir de seus cromossomos são gerados dois novos elementos (filhos). Para gerar os filhos, seleciona-se um ponto de corte aleatoriamente nos cromossomos-pais, de modo que os segmentos a partir do ponto de corte sejam trocados. Veja o exemplo a seguir:

Exemplo 1: Considere dois indivíduos selecionados como pais a partir da população inicial de um algoritmo genético e suponha que o ponto de corte escolhido (aleatoriamente) encontra-se entre as posições 4 e 5 dos cromossomos-pais:



Após o *crossover*, são gerados os seguintes cromossomos-filhos:

Filho 1	0	1	1	1	0	1	1	1	0	1
Filho 2	1	0	1	0	0	1	0	1	1	0

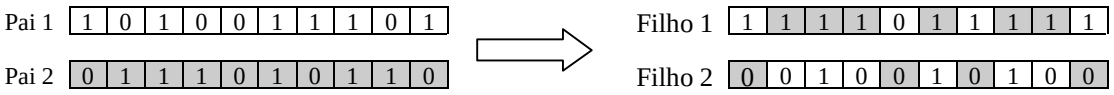
Outros tipos de *crossover* têm sido propostos na literatura. Uma extensão simples do *crossover* de um ponto é o *crossover* de dois pontos. Neste, dois pontos de corte são escolhidos e o material genético será invertido entre eles na posição de ruptura. Observe o exemplo a seguir:

Exemplo 2: Considere os mesmos cromossomos-pais do Exemplo 1. Suponha que os pontos de *crossover* escolhidos estão localizados entre as posições 3 e 4 e entre as posições 7 e 8. Assim, os novos indivíduos produzidos serão:

Filho 1	1	0	1	1	0	1	0	1	0	1
Filho 2	0	1	1	0	0	1	1	1	1	0

Outro tipo de *crossover* muito comum é o *crossover uniforme* (SYSWERDA, 1989): para cada bit no primeiro filho é decidido (com alguma probabilidade fixa p) qual pai vai contribuir com seu respectivo bit para aquela posição. Considere o seguinte exemplo:

Exemplo 3: Tomando como base os mesmos cromossomos-pais dos exemplos anteriores e seja $p = 0{,}5$. Podem ser obtidos pela aplicação do *crossover* uniforme os seguintes indivíduos:



Como o *crossover* uniforme troca bits ao invés de segmentos de bits (que aqui fazem o papel dos genes), ele pode combinar características independentemente da sua posição relativa no cromossomo.

ESHELMAN *et al.* (1989) relata diversos experimentos com vários operadores de *crossover*. Os resultados indicam que o operador com pior desempenho é o *crossover* de um ponto; entretanto, não há nenhum operador de *crossover* que apresente um desempenho superior aos demais em todos os casos. Uma conclusão a que se pode chegar a partir desses resultados é que cada operador de *crossover* é particularmente eficiente para um determinado conjunto de problemas e pode ser extremamente ineficiente para outros.

Embora não seja uma razão determinística para explicar o diferencial de desempenho entre os vários tipos de operadores de *crossover*, é fato que, no caso de a ordenação adotada para posicionar os genes ao longo do cromossomo não interferir no fenótipo, o *crossover* uniforme tende a apresentar um desempenho superior, por ser justamente aquele que não leva em conta a ordenação dos genes, já que opera cada um individualmente. É evidente que um dado gene em um cromossomo, quando selecionado para *crossover* uniforme, será trocado pelo gene de mesma posição no outro cromossomo (aqui a ordem importa), mas esta alteração não interfere na probabilidade de troca dos genes vizinhos (é aqui que a ordem não importa).

Embora existam operadores de recombinação especialmente desenvolvidos para uso com codificação em ponto flutuante, os operadores de *crossover* descritos até aqui também poderiam ser utilizados. Um exemplo específico para o caso de codificação em ponto flutuante é o chamado *crossover aritmético* (MICHALEWICZ, 1996). Este operador é definido como uma fusão de dois vetores (cromossomos): se $\mathbf{x}_1$ e $\mathbf{x}_2$ são dois indivíduos selecionados para *crossover*, os dois filhos resultantes serão $\mathbf{x}'_1 = a\mathbf{x}_1 + (1-a)\mathbf{x}_2$ e $\mathbf{x}'_2 = (1-a)\mathbf{x}_1 + a\mathbf{x}_2$, sendo a um número aleatório pertencente ao intervalo [0, 1]. Esse operador é particularmente apropriado para problemas de otimização numérica com

restrições, onde a região factível é convexa. Se $\mathbf{x}_1$ e $\mathbf{x}_2$ pertencem à região factível, combinações convexas de $\mathbf{x}_1$ e $\mathbf{x}_2$ serão também factíveis, garantindo que o *crossover* não vai gerar indivíduos inválidos para o problema em questão. Outros exemplos de *crossover* especialmente desenvolvidos para utilização em problemas de otimização numérica restritos e codificação em ponto flutuante são o *crossover geométrico* e o *crossover esférico*, descritos em MICHALEWICZ & SCHOENAUER (1996).

Um aspecto importante em um algoritmo genético diz respeito a como escolher os indivíduos que serão submetidos à recombinação. Aqui também existem diversas alternativas possíveis, sendo que entre as mais comuns destacam-se:

- *crossover* entre indivíduos aleatórios: são escolhidos indivíduos da população atual aleatoriamente ou por meio de *Roulette Wheel* (veja Seção 2.4.4);
- *crossover* entre um indivíduo aleatório e o melhor indivíduo: é escolhido um indivíduo da população atual aleatoriamente ou por meio de *Roulette Wheel*, sendo o outro elemento o melhor da população.

O Operador de Mutação

O operador de mutação modifica aleatoriamente um ou mais genes de um cromossomo. A probabilidade de ocorrência de mutação em um gene é denominada *taxa de mutação*. Usualmente, são atribuídos valores pequenos para a taxa de mutação. A idéia intuitiva por trás desse operador é a criação de variabilidade extra na população, mas sem destruir o progresso já obtido no decorrer do processo evolutivo, ou seja, a variabilidade deve se comportar como uma perturbação de efeito localizado.

Considerando codificação binária, o operador de mutação padrão simplesmente troca o valor do gene selecionado (HOLLAND, 1992). Assim, se um gene selecionado para mutação tiver valor 1, passará para 0 após a aplicação do operador e vice-versa.

No caso de problemas com codificação em ponto flutuante, o operador de mutação mais popular é a *mutação gaussiana* (MICHALEWICZ & SCHOENAUER, 1996), modificando todos os componentes de um cromossomo $\mathbf{x} = [x_1 \ldots x_n]$ na forma:

$$\mathbf{x}' = \mathbf{x} + N(0, \sigma),$$

sendo $N(0, \sigma)$ um vetor de variáveis aleatórias independentes, com distribuição normal, média zero e desvio padrão σ.

Um operador importante para problemas em que os indivíduos empregam codificação em ponto flutuante é a *mutação uniforme* (MICHALEWICZ ,1996). Este operador seleciona aleatoriamente um componente $k \in \{1, 2, \ldots, n\}$ do cromossomo $\mathbf{x} = [x_1 \ldots x_k \ldots x_n]$ e gera um indivíduo $\mathbf{x}' = [x_1 \ldots x_k' \ldots x_n]$, onde x_k' é um número aleatório (com distribuição de probabilidade uniforme) amostrado no intervalo $[LB, UB]$. LB e UB são, respectivamente, os limites inferior e superior da variável x_k.

Outro operador de mutação, especialmente desenvolvido para problemas de otimização com restrição e codificação em ponto flutuante é a chamada *mutação não-uniforme* (MICHALEWICZ, 1996; MICHALEWICZ & SCHOENAUER, 1996). A mutação não-uniforme é um operador dinâmico, destinado a melhorar a sintonia fina ao longo do processo evolutivo. Pode ser definido da seguinte forma: seja $\mathbf{x}^t = [x_1 \ldots x_n]$ um cromossomo e suponha que o elemento x_k foi selecionado para mutação. O cromossomo resultante será $\mathbf{x}^{t+1} = [x_1 \ldots x_k' \ldots x_n]$, com

$$x_k' = \begin{cases} x_k + \Delta(t, UB - x_k), & \text{com 50\% de probabilidade} \\ x_k - \Delta(t, x_k - LB), & \text{com 50\% de probabilidade} \end{cases}.$$

A função $\Delta(t, y)$ retorna um valor no intervalo $[0, y]$, tal que a probabilidade de $\Delta(t, y)$ ser próximo de zero aumenta à medida que t aumenta. Esta propriedade faz com que este operador explore mais amplamente o espaço nas gerações iniciais (quando t é pequeno) e mais localmente em gerações avançadas (quando t é grande). MICHALEWICZ (1996) propõe a seguinte função, sendo r um número aleatório no intervalo $[0, 1]$, T um número máximo de gerações e b um parâmetro que determina o grau de dependência do número de iterações (valor proposto: $b = 5$):

$$\Delta(t, y) = y \cdot \left(1 - r^{(1 - t/T)^b}\right).$$

Este operador foi usado com sucesso por DE CASTRO *et al.* (1998) na evolução de condições iniciais para o treinamento de redes neurais artificiais. Outros exemplos de operadores de mutação para problemas de otimização numérica e com codificação em ponto flutuante podem ser encontrados em MICHALEWICZ & SCHOENAUER (1996).

2.4.4 Seleção de Indivíduos para a Próxima Geração

O algoritmo genético clássico utiliza um esquema de seleção de indivíduos para a próxima geração chamado *Roulette Wheel* (GOLDBERG, 1989). O *Roulette Wheel* atribui a cada indivíduo de uma população uma probabilidade de passar para a próxima geração proporcional ao seu *fitness* medido, em relação à somatória do *fitness* de todos os elementos da população. Assim, quanto maior for o *fitness* de um indivíduo, maior a probabilidade dele passar para a próxima geração. Para um exemplo, veja a Figura 2.5.

Nº	Cromossomo	*Fitness*	Graus	% do Total
1	0001110101	6,0	180	50,0
2	0101110011	3,0	90	25,0
3	1101010001	1,5	45	12,5
4	1101010011	1,5	45	12,5
Total		12,0	360	100,0

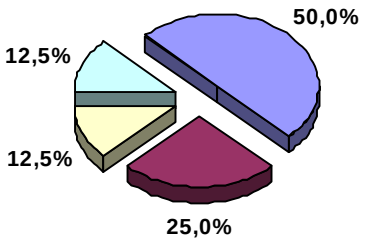


Figura 2.5: Exemplo de aplicação do *Roulette Wheel*: cada indivíduo em uma determinada geração recebe uma probabilidade de passar à próxima geração proporcional ao seu *fitness*, medido em relação ao *fitness* total da população.

Observe que a seleção de indivíduos por *Roulette Wheel* pode fazer com que o melhor indivíduo da população seja perdido, já que a probabilidade dele ser escolhido para compor a próxima geração não é 1. Um recurso paliativo seria escolher como solução o melhor indivíduo encontrado ao longo de todas as gerações do algoritmo. Outra opção mais consistente é simplesmente manter sempre o melhor indivíduo da geração atual na geração seguinte, estratégia esta conhecida como *seleção salvacionista* (FOGEL, 1994; MICHALEWICZ, 1996). Com isso, além do melhor indivíduo ser sempre preservado, ele vai continuar a contribuir com seu código genético na produção de descendentes que irão compor as próximas gerações.

Outro exemplo de mecanismo de seleção é a *seleção baseada em rank* (BÄCK *et al.*, 1997a). Esta estratégia utiliza as posições dos indivíduos quando ordenados de acordo com o *fitness* para determinar a probabilidade de seleção, podendo ser usados mapeamentos lineares ou não-lineares para determinar esta probabilidade.

Para um exemplo de mapeamento não-linear, veja MICHALEWICZ (1996). Uma variação deste mecanismo é simplesmente passar os *N* melhores indivíduos para a próxima geração.

A seguir, são apresentados alguns outros possíveis mecanismos de seleção:

- seleção por diversidade: são selecionados os indivíduos mais diversos da população;
- seleção bi-classista: são selecionados os *P*% melhores indivíduos e os (100 – *P*)% piores indivíduos;
- seleção aleatória: são selecionados aleatoriamente *N* indivíduos da população. Este mecanismo de seleção pode ser subdividido em:
  - salvacionista: seleciona-se o melhor indivíduo e os outros aleatoriamente;
  - não-salvacionista: seleciona-se aleatoriamente todos os indivíduos.

2.5 Considerações Finais

Neste capítulo, foram apresentados os principais conceitos relacionados à evolução natural e à computação evolutiva (mais especificamente aos algoritmos genéticos).

Apesar da área de computação evolutiva ter experimentado um crescimento vertiginoso nos últimos anos, ainda há muitas questões em aberto. MICHALEWICZ (1996) aponta direções promissoras nas quais podem ser esperadas grande atividade e resultados significativos. A seguir, são citadas algumas:

- **Fundamentação Teórica**: apesar da existência de alguns resultados teóricos importantes, a teoria atualmente disponível para análise e síntese de algoritmos evolutivos não é capaz de prover uma base sólida para usuários da computação evolutiva (EIBEN *et al.*, 1999). Além da complexidade inerente aos processos evolutivos, muitas modificações incorporadas ao algoritmo genético clássico (como por exemplo o uso de codificação em ponto flutuante ao invés de codificação binária) impedem a aplicação dos conceitos teóricos já desenvolvidos para o caso clássico.
- **Sistemas Auto-Adaptativos**: os algoritmos evolutivos clássicos exigem definição por parte do usuário de diversos parâmetros, como tamanho da população e probabilidades de *crossover* e mutação. Estes parâmetros permanecem fixos durante toda a execução do algoritmo ou são alterados arbitrariamente em pontos específicos do processo

evolutivo. Em geral, os valores assumidos por estes parâmetros são determinantes para que o algoritmo seja capaz de encontrar uma solução de qualidade, e também para a eficiência na busca desta solução. Entretanto, encontrar valores apropriados para estes parâmetros é uma tarefa custosa em termos computacionais, pois não há uma metodologia eficiente que ajude nesta definição. Assim, muitas abordagens têm sido propostas no sentido do ajuste destes valores durante a execução do algoritmo. Em EIBEN *et al.* (1999), encontram-se diversas técnicas para o controle dos parâmetros durante a execução do algoritmo, mostrando-se uma das áreas de pesquisa mais promissoras em computação evolutiva.

- **Sistemas Co-Evolutivos**: em sistemas co-evolutivos, mais de um processo evolutivo ocorre simultaneamente. Usualmente, considera-se mais de uma população (por exemplo, uma população de "presas" e outra de "predadores") como parte de um processo interativo. Em sistemas desse tipo, a função de *fitness* de uma população pode depender do estado da outra população. Sistemas co-evolutivos podem ser abordagens interessantes para problemas de larga escala, de modo que um problema complexo é decomposto em subproblemas menores. Dessa maneira, existiria um processo evolutivo para cada subproblema e os processos estariam todos inter-relacionados.
- **Algoritmos de Implementação Paralela**: algoritmos evolutivos são adequados para implementação paralela. Como observado por GOLDBERG (1989), não deixa de ser irônico que, em um mundo onde algoritmos seriais são paralelizados por meio de inúmeros truques e contorcionismos, algoritmos genéticos (algoritmos inerentemente paralelos) sejam implementados serialmente através de truques igualmente antinaturais. Entretanto, não há atualmente uma metodologia padrão para paralelização de algoritmos evolutivos. Para uma análise das principais abordagens já propostas para a paralelização de algoritmos genéticos, veja CANTÚ-PAZ (1998).

Apesar de todas as questões em aberto, a computação evolutiva caminha rapidamente para consolidar-se como uma área de atuação científica e, portanto, tudo leva a crer que os algoritmos evolutivos tornar-se-ão em breve parte permanente do conjunto de ferramentas de engenharia (GOLDBERG, 1998).