**O que é um algoritmo evolucionário?**

O objetivo mais importante deste capítulo é descrever o que é um algoritmo evolutivo (EA). A fim de fornecer uma visão unificadora, apresentamos um esquema geral que constitui a base comum para todas as diferentes variantes de algoritmos evolutivos. Os principais componentes dos EAs são discutidos, explicando sua função e questões relacionadas à terminologia. Isso é imediatamente seguido por dois aplicativos de exemplo para tornar as coisas mais concretas. Em seguida, passamos a discutir questões gerais relativas à operação de EAs, para colocá-los em um contexto mais amplo e explicar sua relação com outras técnicas de otimização global.

**3.1 O que é um algoritmo evolucionário?**

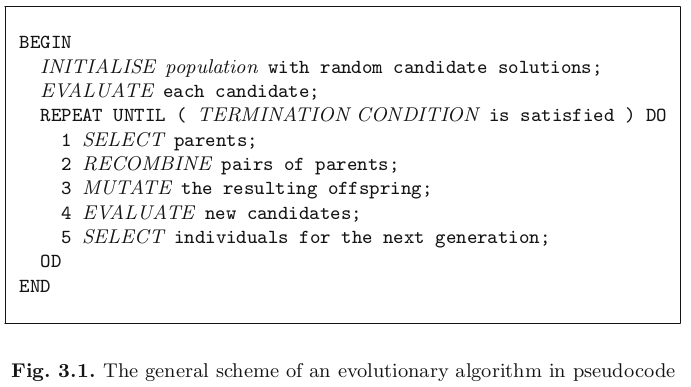
Como sugere a história do campo, existem muitas variantes diferentes de algoritmos evolutivos. A ideia subjacente comum por trás de todas essas técnicas é a mesma: dada uma população de indivíduos em algum ambiente com recursos limitados, a competição por esses recursos causa a seleção natural (sobrevivência do mais apto). Isso, por sua vez, causa um aumento na aptidão da população. Dada uma função de qualidade a ser maximizada, podemos criar aleatoriamente um conjunto de soluções candidatas, ou seja, elementos do domínio da função. Em seguida, aplicamos a função de qualidade a eles como uma medida de aptidão abstrata - quanto maior, melhor. Com base nesses valores de aptidão, alguns dos melhores candidatos são escolhidos para semear a próxima geração. Isso é feito aplicando recombinação e / ou mutação a eles. A recombinação é um operador aplicado a dois ou mais candidatos selecionados (os chamados pais), produzindo um ou mais novos candidatos (os filhos). A mutação é aplicada a um candidato e resulta em um novo candidato. Portanto, a execução das operações de recombinação e mutação nos pais leva à criação de um conjunto de novos candidatos (a prole). Eles têm sua condição física avaliada e então competem - com base em sua condição física (e possivelmente na idade) - com os mais velhos por uma vaga na próxima geração. Este processo pode ser iterado até que um candidato com qualidade suficiente (uma solução) seja encontrado ou um limite computacional previamente definido seja alcançado.

Existem duas forças principais que formam a base dos sistemas evolucionários:

* Operadores de variação (recombinação e mutação) criam a diversidade necessária dentro da população e, assim, facilitam a novidade.
* A seleção atua como uma força que aumenta a qualidade média das soluções na população.

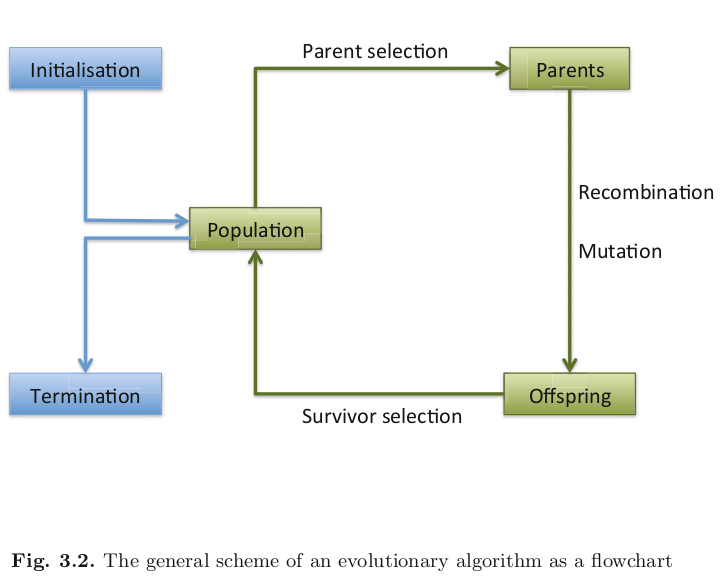
A aplicação combinada de variação e seleção geralmente leva à melhoria dos valores de aptidão em populações consecutivas. É fácil ver este processo como se a evolução estivesse otimizando (ou pelo menos "aproximando") a função de aptidão, aproximando os valores ideais cada vez mais ao longo do tempo. Uma visão alternativa é que a evolução pode ser vista como um processo de adaptação. Nessa perspectiva, o fitness não é visto como uma função objetivo a ser otimizada, mas como uma expressão de requisitos ambientais. A correspondência mais próxima com esses requisitos implica em uma maior viabilidade, que se reflete em um maior número de descendentes. O processo evolutivo resulta em uma população cada vez mais adaptada ao meio ambiente.

Deve-se notar que muitos componentes de tal processo evolutivo são estocásticos. Por exemplo, durante a seleção, os melhores indivíduos não são escolhidos de forma determinística e, normalmente, mesmo os indivíduos fracos têm alguma chance de se tornar um pai ou de sobreviver. Durante o processo de recombinação, a escolha de quais pedaços dos pais serão recombinados é feita de forma aleatória. Da mesma forma, para a mutação, a escolha de quais peças serão alteradas dentro de uma solução candidata e das novas peças para substituí-las é feita aleatoriamente. O esquema geral de um algoritmo evolutivo é dado em pseudocódigo na Fig. 3.1 e é mostrado como um fluxograma na Fig. 3.2.



É fácil ver que esse esquema se enquadra na categoria de algoritmos de geração e teste. A função de avaliação (adequação) fornece uma estimativa heurística da qualidade da solução, e o processo de pesquisa é conduzido pelos operadores de variação e seleção. Algoritmos evolutivos possuem uma série de recursos que podem ajudar a posicioná-los dentro da família de métodos de geração e teste:

* EAs são baseados em população, ou seja, eles processam uma coleção inteira de soluções candidatas simultaneamente.
* A maioria dos EAs usa recombinação, combinando informações de duas ou mais soluções candidatas para criar uma nova.
* EAs são estocásticos.



Os vários dialetos da computação evolucionária que mencionamos anteriormente seguem todos esses contornos gerais, diferindo apenas nos detalhes técnicos. Em particular, diferentes fluxos são frequentemente caracterizados pela representação de uma solução candidata - ou seja, as estruturas de dados usadas para codificar candidatos. Normalmente, isso tem a forma de strings sobre um alfabeto finito em algoritmos genéticos (GAs), vetores de valor real em estratégias de evolução (ESs), máquinas de estado finito em programação evolutiva clássica (EP) e árvores em programação genética (GP). A origem dessas diferenças é principalmente histórica. Tecnicamente, uma representação pode ser preferível a outras se corresponder melhor ao problema fornecido; ou seja, torna a codificação de soluções candidatas mais fácil ou mais natural. Por exemplo, ao resolver um problema de satisfatibilidade com n variáveis ​​lógicas, a escolha direta é usar cadeias de bits de comprimento n de modo que o conteúdo do iº bit denote que a variável i assumiu o valor verdadeiro (1) ou falso (0 ) Portanto, o EA apropriado seria um GA. Para desenvolver um programa de computador que pode jogar damas, as árvores de análise das expressões sintáticas que formam os programas são uma escolha natural para representar soluções candidatas, portanto, uma abordagem GP é provável. É importante observar dois pontos. Primeiro, os operadores de recombinação e mutação trabalhando em candidatos devem corresponder à representação fornecida. Assim, por exemplo, no GP o operador de recombinação trabalha em árvores, enquanto que em AGs ele opera em cordas. Em segundo lugar, em contraste com os operadores de variação, o processo de seleção leva em consideração apenas as informações de adequação e, portanto, funciona independentemente da escolha da representação. Portanto, as diferenças entre os mecanismos de seleção comumente aplicados em cada fluxo são uma questão de tradição e não de necessidade técnica.

Nesta seção, discutimos algoritmos evolutivos em detalhes. Existem vários componentes, procedimentos ou operadores que devem ser especificados para definir um determinado EA. Os componentes mais importantes, indicados por itálico na Fig. 3.1, são:

* representação (definição de indivíduos)
* função de avaliação (ou função de aptidão)
* população
* mecanismo de seleção de pais
* operadores de variação, recombinação e mutação
* mecanismo de seleção de sobrevivente (substituição)

Para criar um algoritmo completo e executável, é necessário especificar cada componente e definir o procedimento de inicialização. Se quisermos que o algoritmo pare em algum estágio 1, devemos também fornecer uma condição de finalização.

**3.2.1 Representação (definição de indivíduos)**

A primeira etapa na definição de uma EA é vincular o ‘mundo real’ ao ‘mundo EA’, ou seja, estabelecer uma ponte entre o contexto do problema original e o espaço de solução de problemas onde a evolução ocorre. Isso geralmente envolve a simplificação ou abstração de alguns aspectos do mundo real para criar um contexto de problema bem definido e tangível dentro do qual soluções possíveis podem existir e ser avaliadas, e este trabalho é frequentemente realizado por especialistas no domínio. A primeira etapa do ponto de vista da solução automatizada de problemas é decidir como as soluções possíveis devem ser especificadas e armazenadas de forma que possam ser manipuladas por um computador. Dizemos que os objetos que formam soluções possíveis dentro do contexto do problema original são chamados de fenótipos, enquanto sua codificação, ou seja, os indivíduos dentro da EA, são chamados de genótipos. Essa primeira etapa do projeto é comumente chamada de representação, pois equivale a especificar um mapeamento dos fenótipos em um conjunto de genótipos que supostamente os representam. Por exemplo, dado um problema de otimização onde as soluções possíveis são inteiros, o dado conjunto de inteiros formaria o conjunto de fenótipos. Nesse caso, pode-se decidir representá-los por seu código binário, então, por exemplo, o valor 18 seria visto como um fenótipo e 10010 como um genótipo que o representa. É importante entender que o espaço do fenótipo pode ser muito diferente do espaço do genótipo, e que toda a busca evolutiva ocorre no espaço do genótipo. Uma solução - um bom fenótipo - é obtida pela decodificação do melhor genótipo após a terminação. Portanto, é desejável que a solução (ótima) para o problema em questão - um fenótipo - seja representada no espaço de genótipo dado. Na verdade, uma vez que em geral não saberemos com antecedência como é essa solução, geralmente é desejável que todas as soluções viáveis ​​possíveis possam ser representadas 2.

A literatura de computação evolucionária contém muitos sinônimos:

* Do lado do contexto do problema original, os termos solução candidata
* ção, fenótipo e indivíduo são usados para denotar soluções possíveis. O espaço de todas as soluções candidatas possíveis é comumente chamado de espaço do fenótipo.
* Do lado da EA, os termos genótipo, cromossomo e, novamente, indivíduo são usados para denotar pontos no espaço onde a busca evolutiva realmente ocorre. Este espaço é frequentemente denominado espaço do genótipo.
* Existem também muitos termos sinônimos para os elementos de indivíduos. Um espaço reservado é comumente chamado de variável, um locus (plural: loci), uma posição ou - em uma terminologia orientada para a biologia - um gene. Um objeto em tal lugar pode ser chamado de valor ou alelo.

Deve-se notar que a palavra 'representação' é usada de duas maneiras ligeiramente diferentes. Às vezes, significa o mapeamento do fenótipo para o espaço do genótipo. Neste sentido, é sinônimo de codificação, por exemplo, pode-se mencionar a representação binária ou codificação binária de soluções candidatas. O mapeamento inverso de genótipos para fenótipos costuma ser chamado de decodificação, sendo necessário que a representação seja invertível de forma que para cada genótipo haja no máximo um fenótipo correspondente. A palavra representação também pode ser usada em um sentido ligeiramente diferente, onde a ênfase não está no mapeamento em si, mas na estrutura de dados do espaço do genótipo. Essa interpretação é a que usamos quando, por exemplo, falamos sobre operadores de mutação para representação binária.

**3.2.2 Função de avaliação (função de fitness)**

O papel da função de avaliação é representar os requisitos aos quais a população deve se adaptar. Ele forma a base para a seleção e, portanto, facilita melhorias. Mais precisamente, ele define o que significa melhoria. Do ponto de vista da resolução de problemas, representa a tarefa a ser resolvida no contexto evolucionário. Tecnicamente, é uma função ou procedimento que atribui uma medida de qualidade aos genótipos. Normalmente, essa função é composta a partir da representação inversa (para criar o fenótipo correspondente) seguida por uma medida de qualidade no espaço do fenótipo. Para ficar com o exemplo acima, se a tarefa é encontrar um inteiro x que maximize x 2, a adequação do genótipo 10010 pode ser definida pela decodificação de seu fenótipo correspondente (10010 → 18) e, em seguida, tomando seu quadrado: 18 2 = 324 .

A função de avaliação é comumente chamada de função de aptidão na CE. Isso pode causar uma terminologia contra-intuitiva se o problema original exigir minimização, porque o termo adequação geralmente está associado à maximização. Matematicamente, no entanto, é trivial transformar minimização em maximização e vice-versa. Muitas vezes, o problema original a ser resolvido por um EA é um problema de otimização (tratado com mais detalhes técnicos na Seção 1.1). Nesse caso, o nome função objetivo é freqüentemente usado no contexto do problema original, e a função de avaliação (adequação) pode ser idêntica ou uma simples transformação da função objetivo fornecida.

**3.2.3 População**

O papel da população é conter (a representação de) soluções possíveis. Uma população é um multiset 3 de genótipos. A população forma a unidade de evolução. Os indivíduos são objetos estáticos que não mudam nem se adaptam; é a população que o faz. Dada uma representação, definir uma população pode ser tão simples quanto especificar quantos indivíduos estão nela, ou seja, definir o tamanho da população. Em alguns EAs sofisticados, uma população tem uma estrutura espacial adicional, definida por meio de uma medida de distância ou uma relação de vizinhança. Isso corresponde vagamente à maneira como as populações reais evoluem dentro do contexto de uma estrutura espacial dada pelas localizações geográficas dos indivíduos. Nesses casos, a estrutura adicional também deve ser definida para especificar completamente uma população.

Em quase todos os aplicativos de EA, o tamanho da população é constante e não muda durante a busca evolutiva - isso produz os recursos limitados necessários para criar competição. Os operadores de seleção (seleção de pais e seleção de sobreviventes) trabalham no nível da população. Em geral, eles levam em consideração toda a população atual, e as escolhas são sempre feitas em relação ao que está presente no momento. Por exemplo, o melhor indivíduo de uma determinada população é escolhido para semear a próxima geração, ou o pior indivíduo de uma determinada população é escolhido para ser substituído por um novo. Essa atividade no nível da população está em contraste com os operadores de variação, que atuam em um ou mais indivíduos pais.

A diversidade de uma população é uma medida do número de soluções diferentes presentes. Não existe uma medida única para a diversidade. Normalmente as pessoas podem se referir ao número de valores de aptidão diferentes presentes, ao número de fenótipos diferentes presentes ou ao número de genótipos diferentes. Outras medidas estatísticas, como entropia, também são usadas. Observe que a presença de apenas um valor de aptidão em uma população não implica necessariamente que apenas um fenótipo esteja presente, uma vez que muitos fenótipos podem ter a mesma aptidão. Da mesma forma, a presença de apenas um fenótipo não implica necessariamente apenas um genótipo. No entanto, se apenas um genótipo estiver presente, isso implica que apenas um fenótipo e valor de aptidão estão presentes.

**3.2.4 Mecanismo de Seleção de Pais**

O papel da seleção dos pais ou do companheiro é distinguir os indivíduos com base em sua qualidade e, em particular, permitir que os melhores indivíduos se tornem pais da próxima geração. Um indivíduo é pai se tiver sido selecionado para sofrer variação a fim de gerar descendentes. Juntamente com o mecanismo de seleção de sobreviventes, a seleção dos pais é responsável por impulsionar as melhorias de qualidade. Na CE, a seleção dos pais é tipicamente probabilística. Assim, indivíduos de alta qualidade têm mais chance de se tornarem pais do que aqueles de baixa qualidade. No entanto, indivíduos de baixa qualidade geralmente têm uma chance pequena, mas positiva; caso contrário, toda a busca poderia se tornar muito gananciosa e a população poderia ficar presa em um ótimo local.

**3.2.5 Operadores de variação (mutação e recombinação)**

O papel dos operadores de variação é criar novos indivíduos a partir dos antigos. No espaço do fenótipo correspondente, isso equivale à geração de novas soluções candidatas. Da perspectiva de pesquisa de geração e teste, os operadores de variação executam a etapa de geração. Operadores de variação em EC são divididos em dois tipos com base em sua aridade, distinguindo versões unárias (mutação) e n-árias (recombinação).

**Mutação**

Um operador de variação unário é comumente chamado de mutação. É aplicado a um genótipo e fornece um mutante (ligeiramente) modificado, a criança ou descendência.

Um operador de mutação é sempre estocástico: sua saída - a criança - depende dos resultados de uma série de escolhas aleatórias. Deve-se notar que nem todos os operadores unários são vistos como mutação. Por exemplo, pode ser tentador usar o termo mutação para descrever um operador heurístico específico do problema que atua sistematicamente em um indivíduo tentando encontrar seu ponto fraco e melhorá-lo realizando uma pequena mudança. No entanto, em geral, a mutação deve causar uma mudança aleatória e imparcial. Por esta razão, pode ser mais apropriado não chamar de mutação de operadores unários heurísticos. Historicamente, a mutação desempenhou um papel diferente em vários dialetos da CE. Assim, por exemplo, na programação genética muitas vezes não é usado de todo, ao passo que nos algoritmos genéticos tem sido tradicionalmente visto como um operador de fundo, fornecendo ao pool genético 'sangue fresco' e na programação evolutiva é o único operador de variação , único responsável pela geração de novos indivíduos.

Operadores de variação formam a implementação evolutiva de etapas elementares (pesquisa), dando ao espaço de pesquisa sua estrutura topológica. Gerar um filho equivale a avançar para um novo ponto neste espaço. Nessa perspectiva, a mutação também tem um papel teórico: pode garantir que o espaço esteja conectado. Existem teoremas que afirmam que um EA irá (com tempo suficiente) descobrir o ótimo global de um determinado problema. Freqüentemente, eles contam com essa propriedade de conexão de que cada genótipo que representa uma solução possível pode ser alcançado pelos operadores de variação [129]. A maneira mais simples de satisfazer essa condição é permitir que o operador de mutação salte para qualquer lugar: por exemplo, permitindo que qualquer alelo seja transformado em qualquer outro com probabilidade diferente de zero. No entanto, muitos pesquisadores sentem que essas provas têm importância prática limitada e as implementações de EA muitas vezes não possuem essa propriedade.

**Recombinação**

Um operador de variação binária é chamado de recombinação ou cruzamento. Como os nomes indicam, esse operador mescla informações de dois genótipos progenitores em um ou dois genótipos descendentes. Assim como a mutação, a recombinação é um operador estocástico: as escolhas de quais partes de cada pai são combinadas e como isso é feito dependem de desenhos aleatórios. Mais uma vez, o papel da recombinação difere entre os dialetos EC: na programação genética, é frequentemente o único operador de variação, e nos algoritmos genéticos é visto como o principal operador de busca, enquanto na programação evolutiva nunca é usado. Operadores de recombinação com maior aridade (usando mais de dois pais) são matematicamente possíveis e fáceis de implementar, mas não têm equivalente biológico. Talvez seja por isso que eles não são comumente usados, embora vários estudos indiquem que eles têm efeitos positivos na evolução [126, 128].

O princípio por trás da recombinação é simples - ao acasalar dois indivíduos com características diferentes, mas desejáveis, podemos produzir uma prole que combina ambas as características. Este princípio tem um forte argumento de apoio - por milênios ele tem sido aplicado com sucesso por criadores de plantas e gado para produzir espécies que dão maiores rendimentos ou têm outras características desejáveis. Algoritmos evolucionários criam uma série de descendentes por recombinação aleatória e esperamos que, embora alguns tenham combinações indesejáveis ​​de características, e a maioria não seja melhor ou pior que seus pais, alguns tenham características melhoradas. A biologia do planeta Terra, onde, com muito poucas exceções, organismos inferiores se reproduzem assexuadamente e organismos superiores se reproduzem sexualmente [288, 289], sugere que a recombinação é a forma superior de reprodução. No entanto, os operadores de recombinação em EAs são geralmente aplicados probabilisticamente, ou seja, com uma chance diferente de zero de não serem realizados.

É importante lembrar que os operadores de variação são dependentes da representação. Assim, para diferentes representações, diferentes operadores de variação devem ser definidos. Por exemplo, se os genótipos são cadeias de bits, a inversão de um bit pode ser usada como um operador de mutação. No entanto, se representarmos as soluções possíveis por estruturas semelhantes a árvores, outro operador de mutação é necessário.

**3.2.6 Mecanismo de Seleção de Sobrevivente (Substituição)**

Semelhante à seleção dos pais, o papel da seleção de sobreviventes ou seleção ambiental é distinguir entre os indivíduos com base em sua qualidade. No entanto, ele é usado em um estágio diferente do ciclo evolutivo - o mecanismo de seleção de sobreviventes é chamado após a criação da prole dos pais selecionados. Conforme mencionado na Seção 3.2.3, na CE o tamanho da população é quase sempre constante. Isso exige que seja feita uma escolha sobre quais indivíduos terão permissão para entrar na próxima geração. Essa decisão muitas vezes é baseada em seus valores de aptidão, favorecendo aqueles com maior qualidade, embora o conceito de idade também seja utilizado com frequência. Em contraste com a seleção dos pais, que é tipicamente estocástica, a seleção do sobrevivente é freqüentemente determinística. Assim, por exemplo, dois métodos comuns são o método baseado na aptidão de classificar o multiconjunto unificado de pais e filhos e selecionar o segmento superior, ou a abordagem tendenciosa por idade de selecionar apenas os filhos.

A seleção do sobrevivente também é freqüentemente chamada de estratégia de substituição. Em muitos casos, os dois termos podem ser usados ​​alternadamente, mas usamos o nome seleção de sobrevivente para manter a terminologia consistente: as etapas 1 e 5 na Fig. 3.1 são ambas seleção de nomes, diferenciados por um qualificador. Da mesma forma, se o algoritmo cria filhos excedentes (por exemplo, 500 filhos de uma população de 100), então usar o termo seleção de sobreviventes é claramente apropriado. Por outro lado, o termo “substituição” pode ser preferido se o número de filhos recém-criados for pequeno em comparação com o número de indivíduos na população. Por exemplo, um algoritmo de "estado estacionário" pode gerar dois filhos por iteração de uma população de 100. Neste caso, a seleção de sobreviventes significa escolher os dois indivíduos antigos que devem ser excluídos para dar espaço para os novos, então é mais eficiente para declarar que todos sobrevivem a menos que sejam excluídos e para escolher quem substituir. Ambas as estratégias podem ser vistas na natureza e têm seus proponentes na CE, portanto, no restante deste livro, seremos pragmáticos sobre esse assunto. Usaremos a seleção de sobreviventes nos cabeçalhos das seções por razões de generalidade e uniformidade, ao mesmo tempo em que usaremos a substituição, se for comumente usada na literatura para o procedimento dado que estamos discutindo.

**3.2.7 Inicialização**

A inicialização é mantida simples na maioria dos aplicativos EA; a primeira população é semeada por indivíduos gerados aleatoriamente. Em princípio, heurísticas específicas do problema podem ser usadas nesta etapa, para criar uma população inicial com maior aptidão. Se isso vale ou não o esforço computacional extra, depende muito do aplicativo em questão. Existem, no entanto, algumas observações gerais a respeito dessa questão que discutimos na Seção 3.5, e também retornaremos a esse assunto no Cap. 10

**3.2.8 Condição de Rescisão**

Podemos distinguir dois casos de uma condição de término adequada. Se o problema tem um nível de aptidão ótimo conhecido, provavelmente vindo de um ótimo conhecido da função objetivo dada, então, em um mundo ideal, nossa condição de parada seria a descoberta de uma solução com essa aptidão. Se sabemos que nosso modelo do problema do mundo real contém simplificações necessárias, ou pode conter ruído, podemos aceitar uma solução que atinge a adequação ideal com uma determinada precisão> 0. No entanto, os EAs são estocásticos e principalmente não há garantias de atingir esse ótimo, de forma que essa condição nunca seja satisfeita e o algoritmo nunca pare. Portanto, devemos estender essa condição com uma que certamente interrompa o algoritmo. As seguintes opções são comumente usadas para este propósito:

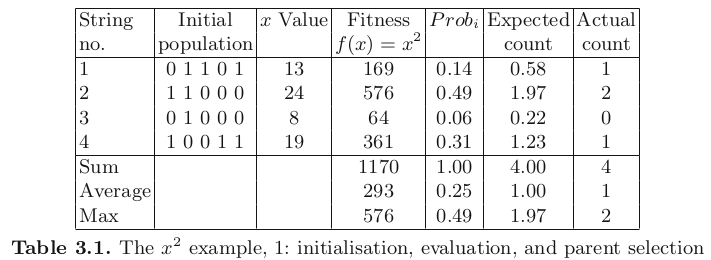
1. O tempo de CPU máximo permitido decorre.
2. O número total de avaliações de aptidão atinge um determinado limite.
3. A melhoria da aptidão permanece abaixo de um valor limite por um determinado período de tempo (ou seja, por uma série de gerações ou avaliações de aptidão).
4. A diversidade da população cai abaixo de um determinado limite.

Tecnicamente, o critério de encerramento real em tais casos é uma disjunção: acerto de valor ótimo ou condição X satisfeita. Se o problema não tem um ótimo conhecido, não precisamos de disjunção. Precisamos simplesmente de uma condição da lista acima ou de uma semelhante que com certeza interrompa o algoritmo. Voltaremos à questão de quando encerrar um EA na Seção. 3,5.

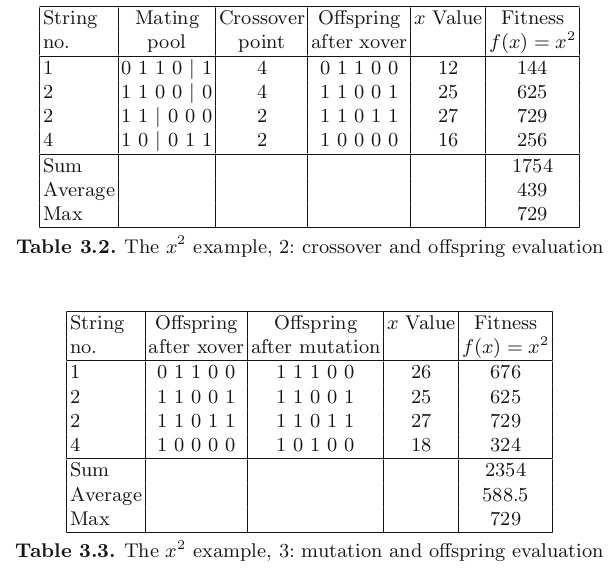
**3.3 Um Ciclo Evolutivo à Mão**

Para ilustrar o funcionamento de um EA, mostramos os detalhes de um ciclo de seleção-reprodução em um problema simples após Goldberg [189], o de maximizar os valores de x 2 para inteiros no intervalo de 0–31. Para executar um ciclo evolutivo completo, devemos tomar decisões de design com relação à representação dos componentes da EA, seleção de pais, recombinação, mutação e seleção de sobreviventes.

Para a representação, usamos uma codificação binária simples de cinco bits mapeando inteiros (fenótipos) para cadeias de bits (genótipos). Para a seleção dos pais, usamos um mecanismo proporcional de aptidão, em que a probabilidade p i de um indivíduo i na população P ser escolhido para ser um dos pais é p i = f (i) / j∈P f (j). Além disso, podemos decidir substituir toda a população de uma vez pela prole criada a partir dos pais selecionados. Isso significa que nosso operador de seleção de sobreviventes é muito simples: todos os indivíduos existentes são removidos da população e todos os novos indivíduos são adicionados a ela sem comparar os valores de aptidão. Isso implica que iremos criar tantos descendentes quantos forem os membros da população. Dada a nossa representação escolhida, os operadores de mutação e recombinação podem ser mantidos simples. A mutação é executada gerando um número aleatório (de uma distribuição uniforme no intervalo [0, 1]) em cada posição de bit e comparando-o a um limite fixo, geralmente chamado de taxa de mutação. Se o número aleatório estiver abaixo dessa taxa, o valor do gene na posição correspondente é invertido. A recombinação é implementada pelo crossover clássico de um ponto. Esse operador é aplicado a dois pais e produz dois filhos escolhendo um ponto de cruzamento aleatório ao longo das strings e trocando os bits dos pais após esse ponto.



Depois de tomar as decisões essenciais de design, podemos executar um ciclo completo de seleção-reprodução. A Tabela 3.1 mostra uma população inicial aleatória de quatro genótipos, os fenótipos correspondentes e seus valores de aptidão. O ciclo então começa com a seleção dos pais para semear a próxima geração. A quarta coluna da Tabela 3.1 mostra o número esperado de cópias de cada indivíduo após a seleção dos pais, sendo f i / f ̄, onde f ̄ denota a aptidão média (os valores exibidos são arredondados para cima). Como pode ser visto, esses números não são inteiros; em vez disso, eles representam uma distribuição de probabilidade, e o pool de acasalamento é criado fazendo escolhas aleatórias para obter uma amostra dessa distribuição. A coluna "Contagem real" representa o número de cópias no pool de acasalamento, ou seja, mostra um resultado possível.



Em seguida, os indivíduos selecionados são emparelhados aleatoriamente e, para cada par, um ponto aleatório ao longo da corda é escolhido. A Tabela 3.2 mostra os resultados do cruzamento no pool de acasalamento dado para pontos de cruzamento após o quarto e segundo genes, respectivamente, juntamente com os valores de aptidão correspondentes. A mutação é aplicada à prole entregue por cruzamento. Mais uma vez, mostramos um resultado possível dos desenhos aleatórios e a Tabela 3.3 mostra os "mutantes" feitos à mão. Neste caso, as mutações demonstradas causaram mudanças positivas na aptidão, mas devemos enfatizar que nas gerações posteriores, à medida que o número de 1 na população aumenta, a mutação será em média (mas nem sempre) deletéria. Embora projetado manualmente, este exemplo mostra um progresso típico: a aptidão média cresce de 293 para 588,5, e a melhor aptidão na população de 576 para 729 após o cruzamento e a mutação.

**3.4 Aplicativos de Exemplo**

**3.4.1 O Problema das Oito Rainhas**

Este é o problema de colocar oito rainhas em um tabuleiro de xadrez 8 × 8 normal, de modo que nenhuma delas possa se checar. Este problema pode ser naturalmente generalizado, gerando o problema das N-rainhas descrito na Seção 1.3. Existem muitas abordagens clássicas de inteligência artificial para esse problema, que funcionam de maneira construtiva ou incremental. Eles começam colocando uma rainha, e depois de terem colocado n rainhas, eles tentam colocar a (n + 1 )ª em uma posição viável onde a nova rainha não verifique nenhuma outra. Normalmente, algum tipo de mecanismo de retrocesso é aplicado; se não houver posição viável para a (n + 1) ª rainha, a enésima é movida para outra posição.

Uma abordagem evolucionária para esse problema é drasticamente diferente, pois não é incremental. Nossas soluções candidatas são configurações de tabuleiro completas (em vez de parciais), que especificam as posições de todas as oito rainhas. O espaço do fenótipo P é o conjunto de todas essas configurações. Claramente, a maioria dos elementos deste espaço são inviáveis, violando a condição de rainhas que não verificam. A qualidade q (p) de qualquer fenótipo p ∈ P pode ser simplesmente quantificada pelo número de pares de rainhas de verificação. Quanto menor for essa medida, melhor será o fenótipo (configuração da placa), e um valor zero, q (p) = 0, indica uma boa solução. A partir dessa observação, podemos formular uma função objetivo adequada a ser minimizada, com um valor ótimo conhecido. Mesmo que não tenhamos definido os genótipos neste ponto, podemos afirmar que a adequação (a ser maximizada) de um genótipo g que representa o fenótipo p é algum inverso de q (p). Existem muitas maneiras possíveis de especificar que tipo de inverso desejamos usar aqui. Por exemplo, 1 / q (p) é uma opção fácil, mas tem a desvantagem de tentar a divisão por zero ser um problema para muitos sistemas de computação. Poderíamos contornar isso observando q (p) = 0 e dizendo que, quando isso ocorrer, teremos uma solução, ou adicionando um pequeno valor, ou seja, 1 / (q (p) +). Outras opções são usar −q (p) ou M - q (p), onde M é um número suficientemente grande para tornar todos os valores de aptidão positivos, por exemplo, M ≥ max {q (p) | p ∈ P}. Esta função de aptidão herda a propriedade de q de ter um M ótimo conhecido.

Para projetar um EA para pesquisar o espaço P, precisamos definir uma representação dos fenótipos de P. A ideia mais direta é usar uma representação de matriz de elementos de P diretamente como genótipos, o que significa que devemos projetar operadores de variação para essas matrizes. Neste exemplo, no entanto, definimos uma representação mais inteligente da seguinte maneira. Um genótipo, ou cromossomo, é uma permutação dos números 1,. . . , 8, e um dado g = (i1,..., I8) denota a configuração (única) do tabuleiro, onde a enésima coluna contém exatamente uma rainha colocada na enésima linha. Por exemplo, a permutação g = (i1,..., I8) representa um tabuleiro onde as rainhas são colocadas ao longo da diagonal principal. O espaço do genótipo G é agora o conjunto de todas as permutações de 1,. . . , 8 e também definimos um mapeamento F: G → P.

É fácil ver que, ao usar tais cromossomos, restringimos a pesquisa às configurações do tabuleiro onde não ocorrem violações de restrição horizontal (duas rainhas na mesma linha) e violações de restrição vertical (duas rainhas na mesma coluna). Em outras palavras, a representação garante metade dos requisitos de uma solução - o que resta a ser minimizado é o número de violações de restrição diagonais. De uma perspectiva formal, escolhemos uma representação que não é sobrejetiva, uma vez que apenas parte de P pode ser obtida pela decodificação de elementos de G. Embora em geral isso possa acarretar o perigo de soluções ausentes em P, em nosso presente exemplo esse não é o caso , pois sabemos a priori que aqueles fenótipos de P \ F (G) nunca podem ser soluções.

A próxima etapa é definir operadores de variação adequados (mutação e cruzamento) para a nossa representação, ou seja, trabalhar em genótipos que são permutações. A característica crucial de um operador adequado é que ele não conduz para fora do espaço G. Na linguagem comum, a progênie de permutações deve ser ela própria permutações. Mais tarde, nas Seitas. 4.5.1 e 4.5.2, discutiremos esses operadores detalhadamente. Aqui, descrevemos apenas uma mutação adequada e um operador de cruzamento para fins de ilustração. Para mutação, podemos usar um operador que seleciona aleatoriamente duas posições em um determinado cromossomo e troca os valores encontrados nessas posições. Um bom cruzamento para permutações é menos óbvio, mas o mecanismo descrito na Fig. 3.3 criará duas permutações filho de dois pais.

1. Selecione uma posição aleatória, o ponto de cruzamento, i ∈ {1,. . . , 7}

2. Corte ambos os pais em dois segmentos nesta posição

3. Copie o primeiro segmento do pai 1 no filho 1 e o primeiro segmento do pai 2 no filho 2

4. Escaneie o pai 2 da esquerda para a direita e preencha o segundo segmento do filho 1 com valores do pai 2, pulando aqueles que já contém

5. Faça o mesmo para o pai 1 e o filho 2

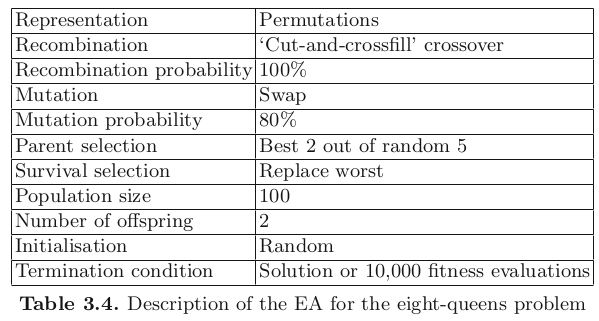
Fig. 3.3. Crossover de corte e preenchimento cruzado

O importante sobre esses operadores de variação é que a mutação causa uma pequena mudança não direcionada e o cruzamento cria filhos que herdam material genético de ambos os pais. Deve-se notar, porém, que pode haver grandes diferenças de desempenho entre os operadores, por exemplo, um EA usando a mutação A pode encontrar uma solução rapidamente, enquanto um usando a mutação B pode nunca encontrar uma solução. Os operadores que esboçamos aqui não são necessariamente eficientes; eles servem apenas como exemplos de operadores que são aplicáveis ​​à representação dada.

A próxima etapa na configuração de uma EA é decidir sobre os mecanismos de seleção e atualização da população. Vamos escolher um esquema simples de gestão da população. Em cada ciclo evolutivo, selecionaremos dois pais, produzindo dois filhos, e a nova população de tamanho n conterá o melhor n dos n + 2 indivíduos resultantes (a população antiga mais as duas novas).

A seleção dos pais (etapa 1 na Fig. 3.1) será feita escolhendo cinco indivíduos aleatoriamente da população e considerando os dois melhores como pais. Isso garante um viés para o uso de pais com aptidão relativamente alta. A seleção de sobreviventes (etapa 5 na Fig. 3.1) verifica quais indivíduos antigos devem ser excluídos para dar lugar aos novos - desde que os novos sejam melhores. Seguindo a convenção de nomenclatura discutida na Seção 3.2.6 definimos uma estratégia de substituição. A estratégia que usaremos funde a população e a descendência, depois os classifica de acordo com a aptidão e exclui os dois piores.

Para obter uma especificação completa, podemos decidir preencher a população inicial com permutações geradas aleatoriamente e encerrar a pesquisa quando encontrarmos uma solução, ou quando 10.000 avaliações de aptidão tiverem decorrido, o que ocorrer primeiro. Além disso, podemos decidir usar um tamanho de população de 100 e usar os operadores de variação com uma certa frequência. Por exemplo, sempre aplicamos crossover aos dois pais selecionados e em 80% dos casos aplicamos mutação à prole. Juntando tudo isso, obtemos um EA conforme resumido na Tabela 3.4.



**3.4.2 O problema da mochila**

O problema da mochila de 0-1, uma generalização de muitos problemas industriais, pode ser descrito resumidamente como segue. Recebemos um conjunto de n itens, cada um dos quais possui algum valor v i e algum custo c i. A tarefa é selecionar um subconjunto desses itens que maximize a soma dos valores, enquanto mantém o custo somado dentro de alguma capacidade C máx. Assim, por exemplo, ao embalar uma mochila para uma viagem de volta ao mundo, devemos equilibrar a utilidade provável dos itens com o fato de que temos um volume limitado (os itens escolhidos devem caber em uma mala) e peso (companhias aéreas impor taxas para bagagem acima de um determinado peso).

É uma ideia natural representar as soluções candidatas para esse problema como cadeias binárias de comprimento n, em que 1 em uma determinada posição indica que um item está incluído e um 0 que foi omitido. O espaço de genótipo correspondente G é o conjunto de todas essas cadeias de tamanho 2 n, que aumenta exponencialmente com o número de itens considerados. Usando este G, fixamos a representação no sentido de estrutura de dados e, em seguida, precisamos definir o mapeamento de genótipos em fenótipos.

A primeira representação (no sentido de um mapeamento) que consideramos considera o espaço do fenótipo P e o espaço do genótipo idênticos. A qualidade de uma dada solução p, representada por um genótipo binário g, é assim determinada pela soma de n os valores dos itens incluídos, ou seja, q (p) = i = 1 v i · g i. No entanto, essa representação simples nos leva a alguns problemas imediatos. Ao usar um mapeamento um-para-um entre o espaço do genótipo G e o espaço do fenótipo P, os genótipos individuais podem corresponder a soluções inválidas que têm um custo associado n maior do que a capacidade, ou seja, i = 1 ci · gi> C max . Essa questão é típica de uma classe de problemas aos quais retornaremos no capítulo 13, e vários mecanismos foram propostos para lidar com isso.

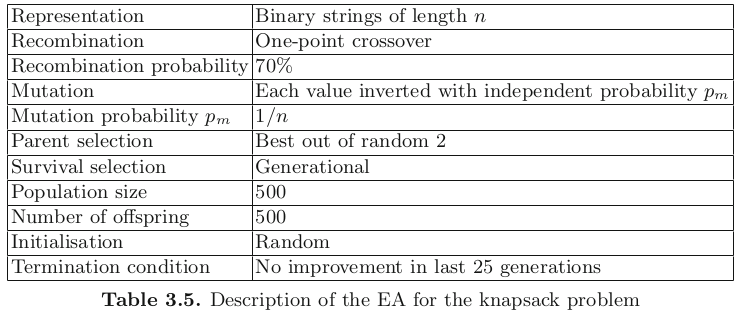
A segunda representação que delineamos aqui resolve este problema empregando uma função decodificadora, que quebra a correspondência um-para-um entre o espaço de genótipos G e o espaço de solução P. Em essência, nossa representação de genótipo permanece a mesma, mas ao criar uma solução, lemos da esquerda para a direita ao longo da string binária e mantemos um registro contínuo do custo dos itens incluídos. Quando encontramos um valor 1, primeiro verificamos se a inclusão do item quebraria nossa restrição de capacidade. Em outras palavras, em vez de interpretar um valor 1 como significando incluir este item, nós o interpretamos como significando incluir este item SE isso não nos levar ao longo da restrição de custo. O efeito desse esquema é fazer o mapeamento do genótipo para o fenótipo no espaço muitos-para-um, uma vez que uma vez que a capacidade tenha sido atingida, os valores de todos os bits à direita da posição atual são irrelevantes, já que nenhum item mais será adicionado à solução. Além disso, esse mapeamento garante que todas as strings binárias representem soluções válidas com uma adequação exclusiva (a ser maximizada).

Tendo decidido por uma representação binária de comprimento fixo, podemos agora escolher os operadores de variação disponíveis na literatura de GA, porque a representação da sequência de bits é "padrão" lá. Um operador de recombinação adequado (mas não necessariamente ótimo) é o chamado cruzamento de um ponto, em que alinhamos dois pais e escolhemos um ponto aleatório ao longo de seu comprimento. Os dois filhos são criados trocando as caudas dos pais nesse ponto. Vamos aplicar isso com 70% de probabilidade, ou seja, para cada par de pais há 70% de chance de criarmos dois filhos por cruzamento e 30% de que os filhos serão apenas cópias dos pais. Um operador de mutação adequado é chamado de inversão de bits: em cada posição invertemos o valor com uma pequena probabilidade p m ∈ [0, 1).

Nesse caso, criaremos o mesmo número de descendentes que temos em nossa população inicial. Conforme observado acima, criamos dois descendentes de cada um dos pais, portanto, selecionaremos esse número de pais e os emparelharemos aleatoriamente. Usaremos um torneio para seleção dos pais, onde cada vez escolhemos dois membros da população ao acaso (com reposição), e aquele com o maior valor q (p) vence o torneio e se torna um pai. Iremos instituir um esquema geracional para a seleção de sobreviventes, ou seja, toda a população em cada iteração é descartada e substituída por seus descendentes.

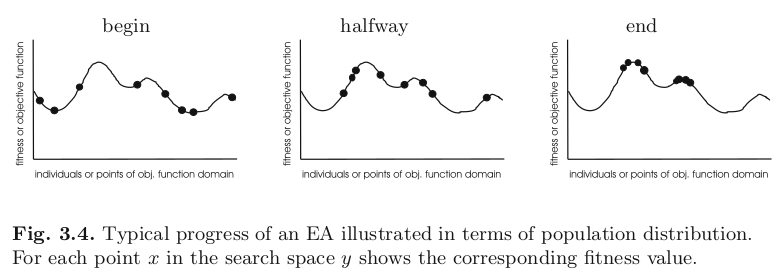
Finalmente, devemos considerar a inicialização (o que faremos por escolha aleatória de 0 e 1 em cada posição de nossa população inicial) e o término. Nesse caso, não sabemos o valor máximo que podemos atingir, então vamos executar nosso algoritmo até que nenhuma melhora na aptidão do melhor membro da população tenha sido observada por 25 gerações.

Já definimos nossa probabilidade de cruzamento como 0,7; trabalharemos com um tamanho de população de 500 e uma taxa de mutação de p m = 1 / n, ou seja, isso mudará em média um valor em cada prole. Nosso algoritmo evolutivo para lidar com este problema pode ser especificado como abaixo na Tabela 3.5.

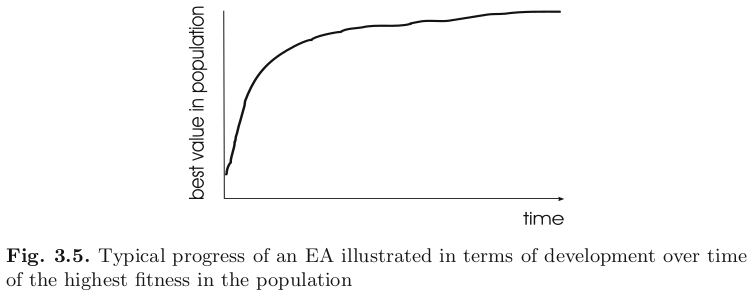


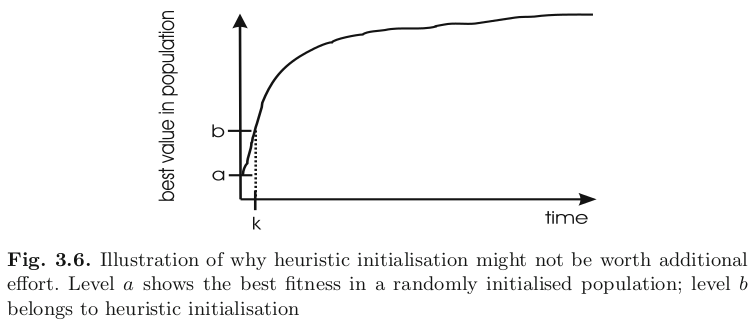
**3.5 A Operação de um Algoritmo Evolucionário**

Algoritmos evolucionários têm algumas propriedades bastante gerais sobre como funcionam. Para ilustrar como um EA normalmente funciona, assumiremos uma função objetivo unidimensional a ser maximizada. A Figura 3.4 mostra três estágios da busca evolutiva, mostrando como os indivíduos podem estar tipicamente distribuídos no início, em algum lugar na metade e no final da evolução. No primeiro estágio, diretamente após a inicialização, os indivíduos são espalhados aleatoriamente por todo o espaço de busca (Fig. 3.4, à esquerda). Depois de apenas algumas gerações, essa distribuição muda: por causa dos operadores de seleção e variação, a população abandona as regiões de baixa aptidão e começa a subir as colinas (Fig. 3.4, no meio). Mais tarde (próximo ao final da pesquisa, se a condição de término for definida apropriadamente), toda a população está concentrada em torno de alguns picos, alguns dos quais podem ser subótimos. Em princípio, é possível que a população suba a colina errada, deixando todos os indivíduos posicionados em torno de um ótimo local, mas não global. Embora não haja uma definição rigorosa universalmente aceita dos termos exploração e exploração, essas noções são frequentemente usadas para categorizar fases distintas do processo de pesquisa. Grosso modo, a exploração é a geração de novos indivíduos em regiões ainda não testadas do espaço de busca, enquanto a exploração significa a concentração da busca na vizinhança de boas soluções conhecidas. Os processos de pesquisa evolucionária são freqüentemente referidos em termos de um trade-off entre exploração e exploração. Muito do primeiro pode levar a uma pesquisa ineficiente, e muito do último pode levar a uma propensão de focar a pesquisa muito rapidamente (ver [142] para uma boa discussão dessas questões). A convergência prematura é o conhecido efeito de perder a diversidade populacional muito rapidamente e ficar preso a um ótimo local. Este perigo está geralmente presente em algoritmos evolutivos, e as técnicas para evitá-lo são discutidas no Cap. 5



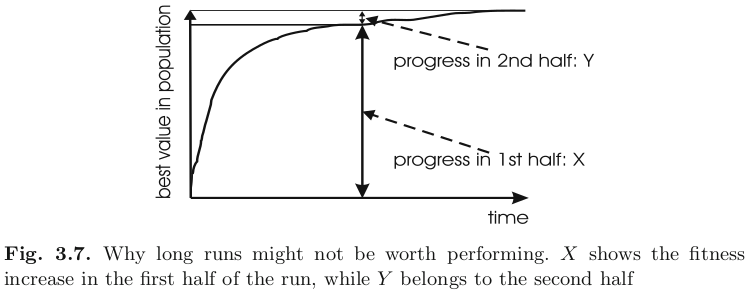
O outro efeito que queremos ilustrar é o comportamento dos EAs a qualquer momento, traçando o desenvolvimento do melhor valor de aptidão da população ao longo do tempo (Fig. 3.5). Esta curva mostra um progresso rápido no início e um nivelamento mais tarde. Isso é típico de muitos algoritmos que funcionam por meio de melhorias iterativas na (s) solução (ões) inicial (is). O nome ‘anytime’ vem da propriedade de que a pesquisa pode ser interrompida a qualquer momento, e o algoritmo terá alguma solução, mesmo que não seja o ideal. Com base nesta curva a qualquer momento, podemos fazer algumas observações gerais sobre a inicialização e a condição de término para EAs. Na seção 3.2.7 questionamos se vale a pena colocar esforço computacional extra na aplicação de heurísticas inteligentes para semear a população inicial com indivíduos melhores do que aleatórios. Em geral, pode-se dizer que a curva de progresso típica de um processo evolutivo o torna desnecessário. Isso é ilustrado na Fig. 3.6. Como a figura indica, o uso da inicialização heurística pode iniciar a pesquisa evolucionária com uma melhor população. No entanto, normalmente algumas (k na figura) gerações são suficientes para atingir esse nível, tornando o esforço extra questionável. No cap. 10 voltaremos a este assunto.

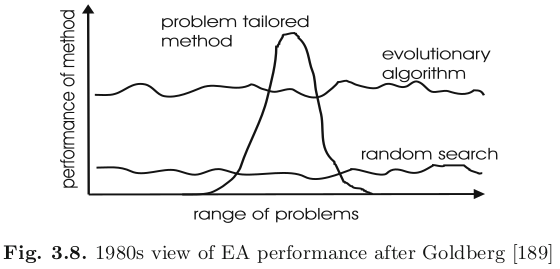




O comportamento a qualquer momento também dá algumas indicações gerais sobre a escolha das condições de terminação para EAs. Na Fig. 3.7, dividimos a corrida em duas seções igualmente longas. Como a figura indica, o progresso em termos de aumento de aptidão na primeira metade da corrida (X) é significativamente maior do que na segunda metade (Y). Isso sugere que pode não valer a pena permitir execuções muito longas. Em outras palavras, devido ao comportamento frequente dos EAs a qualquer momento, podemos supor que o esforço despendido após um certo tempo (número de avaliações de aptidão) provavelmente não resultará em melhor qualidade de solução.

Encerramos esta análise do comportamento da EA observando o desempenho da EA de uma perspectiva global. Ou seja, em vez de observar uma execução do algoritmo, consideramos o desempenho dos EAs para uma ampla gama de problemas. A Fig. 3.8 mostra a vista dos anos 1980 após Goldberg [189]. O que a figura indica é que os EAs apresentam um desempenho razoavelmente bom em uma ampla gama de problemas. Esse padrão de desempenho pode ser comparado à pesquisa aleatória e aos algoritmos adaptados a um tipo de problema específico. EAs são sugeridos para superar claramente a pesquisa aleatória. Em contraste, um algoritmo adaptado ao problema tem um desempenho muito melhor do que um EA, mas apenas no tipo de problema para o qual foi projetado. À medida que nos afastamos desse tipo de problema para problemas diferentes, o algoritmo específico do problema perde desempenho rapidamente. Nesse sentido, os EAs e os algoritmos específicos do problema formam dois extremos opostos. Essa percepção desempenhou um papel importante no posicionamento dos EAs e enfatizando a diferença entre a pesquisa evolucionária e a aleatória, mas mudou gradualmente na década de 1990 com base em novos insights da prática e da teoria. A visão contemporânea reconhece a possibilidade de combinar os dois extremos em um algoritmo híbrido. Esta questão é tratada em detalhes no cap. 10, onde também apresentamos a versão revisada da Fig. 3.8. Quanto às considerações teóricas, o teorema No Free Lunch mostrou que (sob algumas condições) nenhum algoritmo de caixa preta pode superar o passeio aleatório quando calculado sobre "todos" os problemas [467]. Ou seja, mostrar a linha do EA sempre acima da pesquisa aleatória é fundamentalmente incorreto. Isso é discutido mais adiante no Cap. 16

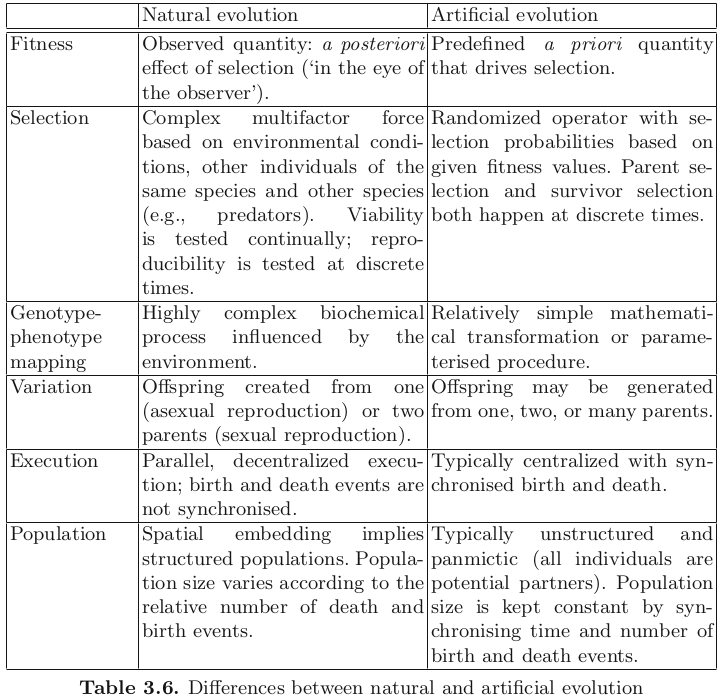




**3.6 Evolução Natural Versus Artificial**

Do ponto de vista do substrato subjacente, o surgimento da computação evolucionária pode ser considerada uma grande transição dos princípios evolutivos do wetware, o reino da biologia, para o software, o reino dos computadores. Isso foi possível com o uso de computadores como instrumentos para criar mundos digitais que são muito flexíveis e muito mais controláveis ​​do que a realidade física em que vivemos. Junto com o aumento da compreensão dos mecanismos genéticos por trás da evolução, isso trouxe a oportunidade de nos tornarmos ativos. mestres de processos evolutivos que são totalmente projetados e executados por experimentadores humanos de cima.

Pode-se argumentar que os algoritmos evolutivos não são modelos fiéis de evolução natural. No entanto, eles certamente são uma forma de evolução. Como expresso por Dennett [116]: Se você tem variação, hereditariedade e seleção, então você deve obter evolução. Na Tabela 3.6, comparamos a evolução natural e a evolução artificial conforme usadas nos algoritmos evolutivos contemporâneos.



**3.7 Computação Evolutiva, Otimização Global e Outros Algoritmos de Pesquisa**

No cap. 2 notamos que algoritmos evolutivos são freqüentemente usados ​​para otimização de problemas. É claro que os EAs não são a única técnica de otimização conhecida, portanto, nesta seção, explicamos onde os EAs se enquadram na classe geral de métodos de otimização e por que são cada vez mais interessantes.

Em um mundo ideal, possuiríamos a tecnologia e os algoritmos que poderiam fornecer uma solução comprovadamente ótima para qualquer problema que pudéssemos apresentar adequadamente ao sistema. Na verdade, esses algoritmos existem: uma enumeração exaustiva de todas as soluções possíveis para um problema é claramente um desses algoritmos. Além disso, para muitos problemas que podem ser expressos em uma formulação matemática adequada, técnicas muito mais rápidas e exatas, como pesquisa de ramificação e limite, são bem conhecidas. No entanto, apesar do rápido progresso na tecnologia de computação, e mesmo se não houver uma suspensão da Lei de Moore, muitas vezes os tipos de problemas colocados pelos usuários excedem em suas demandas a capacidade da tecnologia para respondê-los.

Décadas de pesquisa em ciência da computação nos ensinaram que muitos problemas do mundo real podem ser reduzidos em sua essência a formas abstratas bem conhecidas, para as quais o número de soluções potenciais cresce muito rapidamente com o número de variáveis ​​consideradas. Por exemplo, muitos problemas de transporte podem ser reduzidos ao conhecido problema do caixeiro viajante (TSP): dada uma lista de destinos, construa o passeio mais curto que visite cada destino exatamente uma vez. Se tivermos n destinos, com distâncias simétricas entre eles, o número de passeios possíveis é n! / 2 = n · (n − 1) · (n − 2) ·. . . · 3, que é exponencial em n. Para alguns desses problemas abstratos, métodos exatos são conhecidos, cuja complexidade de tempo escala linearmente (ou pelo menos polinomialmente) com o número de variáveis ​​(consulte [212] para uma visão geral). No entanto, é amplamente aceito que para muitos tipos de problemas encontrados, nenhum algoritmo existe - como foi discutido na Seção 1.4. Assim, apesar do aumento do poder de computação, além de um determinado tamanho de problema, devemos abandonar a busca por soluções comprovadamente ótimas e buscar outros métodos para encontrar boas soluções.

O termo otimização global refere-se ao processo de tentativa de encontrar a solução com o valor ideal para alguma função de aptidão. Na terminologia matemática, estamos tentando encontrar a solução x ∗ fora de um conjunto de soluções possíveis S, tal que x = x ∗ ⇒ f (x ∗) ≥ f (x) ∀x ∈ S. Aqui assumimos a problema de maximização - a desigualdade é simplesmente revertida para minimização. Conforme observado acima, uma série de algoritmos determinísticos existem que, se permitidos

Conforme observado acima, existem vários algoritmos determinísticos que, se forem executados até a conclusão, têm a garantia de encontrar x ∗. O exemplo mais simples é, claro, a enumeração completa de todas as soluções em S, que pode levar um tempo exponencialmente longo à medida que o número de variáveis ​​aumenta. Uma variedade de outras técnicas, conhecidas coletivamente como decomposição em caixa, baseiam-se na ordenação dos elementos de S em algum tipo de árvore e, em seguida, no raciocínio sobre a qualidade das soluções em cada ramo para decidir se devem investigar seus elementos. Embora métodos como branch e bound possam às vezes fazer um progresso muito rápido, no pior caso causado pela pesquisa em uma ordem subótima), a complexidade de tempo dos algoritmos ainda é a mesma que a enumeração completa.

Outra classe de métodos de pesquisa é conhecida como heurística. Isso pode ser considerado um conjunto de regras para decidir qual solução potencial de S deve ser gerada e testada em seguida. Para algumas heurísticas aleatórias, como o recozimento simulado [2, 250] e certas variantes de EAs, as provas de convergência existem de fato, ou seja, elas são garantidas para encontrar x ∗. Infelizmente, esses algoritmos são bastante fracos, no sentido de que não identificarão x ∗ como sendo globalmente ideal, ao invés de simplesmente a melhor solução vista até agora.

Uma importante classe de heurísticas é baseada na ideia de usar operadores que impõem algum tipo de estrutura aos elementos de S, de forma que cada ponto x tenha associado a ele um conjunto de vizinhos N (x). Na Fig. 2.2, as variáveis ​​(traços) X e Y foram consideradas com valor real, o que impõe uma estrutura natural a S. O leitor deve notar que, para aqueles tipos de problema em que cada variável assume um de um conjunto finito de valores (a chamada otimização combinatória), existem muitas estruturas de vizinhança possíveis. Como um exemplo de como a paisagem 'vista' por um algoritmo de busca local depende de sua estrutura de vizinhança, o leitor pode desejar considerar como seria um tabuleiro de xadrez se o reordenássemos, de modo que os quadrados que são possíveis próximos movimentos para o peça do cavaleiro estavam adjacentes uma à outra. Assim, pontos que são localmente ótimos (mais adequados do que todos os seus vizinhos) na paisagem induzida por uma estrutura de vizinhança podem não ser para outra. No entanto, por sua definição, o ótimo global x ∗ sempre será mais adequado do que todos os seus vizinhos em qualquer estrutura de vizinhança.

Os chamados algoritmos de busca local [2] e suas muitas variantes funcionam tomando uma solução inicial x, e então procurando as soluções candidatas em N (x) para um x que tem um desempenho melhor do que x. Se tal solução existe, então ela é aceita como a nova solução incumbente, e a busca prossegue examinando as soluções candidatas em N (x). Esse processo acabará por levar à identificação de um ótimo local: uma solução superior a todas as de sua vizinhança. Esses algoritmos (frequentemente chamados de hill climbing para problemas de maximização) foram bem estudados ao longo das décadas. Eles têm a vantagem de serem frequentemente rápidos em identificar uma boa solução para o problema, que às vezes é tudo o que é necessário em aplicações práticas. No entanto, a desvantagem é que os problemas freqüentemente exibirão vários ótimos locais, alguns dos quais podem ser significativamente piores do que o ótimo global, e nenhuma garantia pode ser oferecida para a qualidade da solução encontrada.

Uma série de métodos foram propostos para contornar este problema mudando o cenário de busca, seja mudando a estrutura de vizinhança (por exemplo, busca de vizinhança variável [208]), ou atribuindo temporariamente baixa adequação a boas soluções já vistas ( por exemplo, pesquisa Tabu [186]). No entanto, a base teórica por trás desses algoritmos ainda está em gestação.

Existem vários recursos dos EAs que os distinguem dos algoritmos de busca local, relacionados principalmente ao uso de uma população. A população fornece ao algoritmo um meio de definir uma função de distribuição de probabilidade não uniforme (p.d.f.) que rege a geração de novos pontos de S. Este p.d.f. reflete possíveis interações entre pontos em S que estão atualmente representados na população. As interações surgem da recombinação de soluções parciais de dois ou mais membros da população (pais). Este p.d.f. potencialmente complexo contrasta com a distribuição globalmente uniforme da busca aleatória cega e a distribuição localmente uniforme usada por muitos outros algoritmos estocásticos, como recozimento simulado e vários algoritmos de escalada.

A capacidade dos EAs de manter um conjunto diversificado de pontos fornece não apenas um meio de escapar dos ótimos locais, mas também um meio de lidar com espaços de busca grandes e descontínuos. Além disso, como será visto em capítulos posteriores, se várias cópias de uma solução podem ser geradas, avaliadas e mantidas na população, isso fornece uma maneira natural e robusta de lidar com problemas onde há ruído ou incerteza associada à atribuição de uma pontuação de aptidão para uma solução candidata.