Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de Computação
Departamento de Engenharia de Computação e Automação Industrial

Contribuições à Solução de Problemas de Escalonamento pela Aplicação Conjunta de Computação Evolutiva e Otimização com Restrições

RICARDO CONCILIO

Orientador: PROF. DR. FERNANDO JOSÉ VON ZUBEN
(DCA – FEEC – Unicamp)

Dissertação apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Elétrica.

Banca Examinadora: Prof. Dr. Ricardo Ribeiro Gudwin (DCA/FEEC/Unicamp)
Prof. Dr. Luis Gimeno Latre (DCA/FEEC/Unicamp)
Profa. Dra. Maria Teresa Moreira Rodrigues (DESQ/FEQ/Unicamp)

Campinas
Estado de São Paulo – Brasil
Dezembro de 2000

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

C748c

Concilio, Ricardo
Contribuições à solução de problemas de escalonamento pela aplicação conjunta de computação evolutiva e otimização com restrições / Ricardo Concilio.--Campinas, SP: [s.n.], 2000.

Orientador: Fernando José Von Zuben
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Algoritmos genéticos. 2. Otimização combinatória. 3. Inteligência artificial. I. Von Zuben, Fernando José. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

"Noutra acepção, considera-se a fé a confiança que se deposita na realização de determinada coisa, a certeza de se atingir um objetivo. ... estando apoiada na inteligência e na compreensão das coisas, tem a certeza de chegar ao fim."

RESUMO

Esta dissertação apresenta contribuições junto à solução de dois problemas de escalonamento: geração de turnos completos em torneios e definição de grade horária de professores em instituições de ensino (alocação de carga didática). Tratam-se de problemas de grande interesse prático, caracterizados por questões de factibilidade e uma explosão combinatória de candidatos à solução. Sendo assim, a atuação direta de um especialista e a aplicação de ferramentas convencionais de busca não conduzem a resultados satisfatórios.

A estratégia de solução proposta está baseada na aplicação conjunta de computação evolutiva, busca local e otimização baseada em restrições. Embora outras abordagens evolutivas já tenham sido propostas na literatura, a empregada nesta dissertação inova ao sugerir uma representação genética compacta aliada a um algoritmo de expansão de código, o qual emprega rotinas de reparação e de busca local.

Comparadas às soluções já implementadas para problemas reais de escalonamento, aquelas obtidas a partir da estratégia de solução proposta neste trabalho apresentaram melhor desempenho e a quantidade de recursos computacionais requeridos para produzir a solução é aceitável.

A aplicação conjunta de computação evolutiva, busca local e técnicas de otimização baseada em restrições pode ser estendida ao tratamento de outros problemas de escalonamento, assumindo a existência de uma codificação genética compacta e a disponibilidade de um algoritmo de otimização baseado em restrições.

ABSTRACT

This dissertation presents contributions to the solution of two assignment problems: generation of a complete set of rounds in tournaments, and definition of timetables in educational establishments (allocation of didactic charge). They represent practical problems of high interest, being characterized by feasibility aspects and a combinatorial explosion of solution candidates. In this case, the direct actuation of an expert and the use of conventional search tools guide to unsatisfactory results.

The proposed solution strategy is based on the joint application of evolutionary computation, local search and restriction-based optimization. Although other evolutionary approaches have already been proposed in the literature, the one considered here innovates, since it suggests a compact genetic codification in conjunction with an algorithm to expand the code, using repairing and local search routines.

When compared with the solutions already implemented to deal with real-world assignment problems, the ones obtained from the solution strategy proposed in this work presented a better performance and the required amount of computational resource to produce the solution was reasonable.

The joint application of evolutionary computation, local search and restriction-based optimization may be extended to deal with other assignment problems, assuming the existence of a compact genetic codification and the availability of an algorithm for restriction-based optimization.

AGRADECIMENTOS

São vários os nomes que deveriam ser citados neste momento, entretanto permito-me não fazê-lo. Certamente há o receio do esquecimento de alguém, o que seria algo imperdoável.

Também, é claro, que a vida é repleta de antagonismos e, dessa maneira, não poderia deixar de fazer um agradecimento especial ao meu orientador. O Prof. Dr. Fernando José Von Zuben demonstrou grande conhecimento técnico em várias áreas (o que é óbvio), norteando de maneira precisa os objetivos que deveriam e seriam atingidos. Não tão óbvio assim, por ser uma característica inata de cada pessoa, é a sua capacidade de inteligência emocional. O que significa que as suas palavras de incentivo foram sempre importantes e bem-vindas.

Basta, por último, manifestar a minha gratidão a todos aqueles que estiveram ao meu lado, lançando ou intuindo novas idéias para a resolução do que foi proposto. A ajuda desses amigos sempre foi de fundamental importância.

A todos, obrigado. E vamos para a próxima.

LISTA DE TABELAS

LISTA DE FIGURAS

ÍNDICE

CAPÍTULO 1

INTRODUÇÃO

Problemas de otimização de natureza numérica, sujeitos a múltiplas restrições, envolvem basicamente uma função-objetivo a ser otimizada e um conjunto de restrições que devem ser simultaneamente atendidas, na forma:

$$\begin{cases} \min\limits_{x} < \text{função - objetivo dependente de } x > \\ \text{s.a.} \quad < \text{conjunto de restrições impostas a } x > \end{cases}$$

Duas etapas estão envolvidas no processo de solução (SHERALI *et al.*, 1993):

1. formulação matemática do problema, procurando descrever todos os aspectos relevantes, seja em termos de objetivos a serem otimizados ou restrições a serem atendidas;
2. obtenção da solução pela aplicação de ferramentas de otimização, desenvolvidas a partir da formulação matemática do problema.

Dependendo das características do problema, a sua formulação leva a um processo de solução que pode ser expresso algebricamente, de forma fechada. Como exemplo, considere a seguinte instância de um problema de otimização multivariável, com restrições lineares de igualdade:

$$\begin{cases} \min\limits_{\mathbf{u}} \dfrac{1}{2}\left\|\mathbf{u}-\mathbf{z}\right\|_2^2, \text{ com } \mathbf{u}, \mathbf{z} \in \Re^n, \ A \in \Re^{m\times n} \ (m < n) \text{ e } \mathbf{b} \in \Re^m, \\ \text{s.a.} \quad A\mathbf{u} = \mathbf{b} \end{cases}$$

sendo:

- $\left\|\cdot\right\|_2$ a norma euclidiana, tal que $\left\|\mathbf{x}\right\|_2^2 = \sum_{i=1}^{n} x_i^2$ para $\mathbf{x} \in \Re^n$;
- $\mathbf{z}$ uma constante e A uma matriz de posto completo (indicando que AA^T é inversível);

Aplicando as condições necessárias de otimalidade, resulta a solução ótima:

$$\mathbf{u}^* = \mathbf{z} - A^T\left(AA^T\right)^{-1}\left(A\mathbf{z}-\mathbf{b}\right).$$

No entanto, grande parte dos problemas de otimização apresentam características (por exemplo, não-linearidades) que impedem a obtenção de uma solução analítica, requerendo a aplicação de processos iterativos de busca da solução, a partir de uma condição inicial. A busca iterativa é geralmente implementada em computador, devido ao elevado custo de processamento associado à sua execução.

A presença de restrições de diversas naturezas introduz uma complexidade adicional por dividir o espaço de candidatos à solução em duas classes: soluções factíveis (que atendem a todas as restrições simultaneamente) e soluções infactíveis (que violam uma ou mais restrições).

Além disso, outra questão a ser considerada é o comportamento de convergência do processo iterativo. Três condições são desejáveis:

- garantia de convergência;
- tempo de convergência compatível com as necessidades de cada aplicação;
- convergência para uma solução de boa qualidade.

Basicamente, quanto mais complexo o problema de otimização e quanto maior seu tamanho (número de variáveis envolvidas), menor a chance de ter atendidas estas três condições simultaneamente, principalmente quando o método de otimização empregado não é suficientemente poderoso na execução do processo de busca da solução.

Neste trabalho, serão considerados problemas com elevado grau de complexidade, apresentando múltiplos objetivos devidamente ponderados, múltiplas restrições e grande número de variáveis. Estas propriedades levam a uma explosão combinatória de candidatos à solução, de modo que uma busca exaustiva pela solução ótima, dentre as soluções candidatas, representa um procedimento computacionalmente intratável (AHUJA *et al.*, 1993).

A tratabilidade de problemas de otimização com este grau de complexidade é conseguida caso se abra mão da solução ótima, em prol da obtenção de uma solução de boa qualidade. Como muitas vezes não se têm condições de saber o quão distante da solução ótima se encontra a solução obtida, a qualidade desta é medida em relação às soluções candidatas anteriormente produzidas pelo processo iterativo, a partir de uma condição inicial.

Dentre outras técnicas, a computação evolutiva tem se destacado no tratamento deste tipo de problema (BÄCK *et al.*, 1997b), pelas seguintes razões (GOLDBERG, 1989):

- por trabalhar com múltiplos candidatos à solução ao mesmo tempo em uma dada geração;
- por privilegiar melhores soluções candidatas na composição das próximas gerações;
- por ser capaz de detectar regiões promissoras no espaço de busca e atingir soluções de boa qualidade sem a necessidade de avaliar um grande número de candidatos.

Segundo BEASLEY (1997), o termo computação evolutiva é relativamente novo, tendo surgido em 1991, e representa um esforço para unir os pesquisadores que trabalhavam com simulação computacional nas áreas ligadas à evolução. As técnicas de algoritmos genéticos, estratégias e programação evolutiva têm um ponto fundamental em comum: envolvem aplicação de operadores genéticos, avaliação e seleção de indivíduos da população. Essas características formam a essência do processo de evolução, seja ele em computador ou no mundo real.

Conforme FOGEL (1997), as aplicações de computação evolutiva podem ser distribuídas em várias áreas, sendo que cada uma delas possuirá uma grande quantidade de ramificações. Em geral, seriam as seguintes: planejamento, modelagem, identificação, controle e classificação. Esta técnica está sendo aplicada na solução de problemas cada vez mais complexos e de natureza diversificada.

A grande dificuldade na aplicação de computação evolutiva a problemas multi-objetivo e com múltiplas restrições é como chegar a uma representação genotípica das soluções candidatas e como obter novas soluções candidatas que sejam factíveis, a partir da aplicação de operadores genéticos (MICHALEWICZ, 1997, C5.5). Uma das contribuições deste trabalho está na proposição de uma representação genética compacta, associada a um algoritmo de expansão de código, representando uma iniciativa de se explorar conjuntamente a computação evolutiva e métodos de busca baseados no atendimento de restrições. Um gerador de números pseudo-aleatórios com a propriedade de repetitividade é utilizado para garantir que a expansão de código sempre produza o mesmo código expandido para cada código compacto possível.

A estratégia de solução que será utilizada envolverá heurísticas para a geração de populações factíveis para o processo evolutivo, pela aplicação de um algoritmo de reparação. Além disso, procedimentos de busca local são empregados para aumentar a velocidade de convergência para ótimos locais.

1.1 Estudo de Casos

Caso 1: Geração Automática de Turnos Completos em Torneios

Em torneios envolvendo disputas dois-a-dois entre os participantes, sendo cada disputa denominada um jogo do torneio, usualmente devem ser atendidas as seguintes restrições básicas:

- cada participante deve jogar contra todos os outros uma única vez;
- cada participante deve obrigatoriamente jogar uma única vez na mesma rodada, a qual representa o menor conjunto de jogos que inclua todos os participantes;
- assumindo que existe distinção de mando de jogo e considerando todas as rodadas, a diferença entre o número de jogos no domínio de cada participante e fora dele deve ser no máximo um.

Se n é o número de participantes, então um turno completo será composto por $n-1$ rodadas, sendo que existe uma explosão combinatória de candidatos factíveis à solução (turnos completos). É assumido aqui que n é par. Na eventualidade de n ser ímpar, basta introduzir um participante fictício, indicando que seu adversário naquela rodada irá folgar. Se o torneio envolver dois turnos, o segundo turno vai representar os jogos de volta do primeiro turno, já que existe uma distinção entre jogos dentro e fora do domínio de cada participante.

Em face da existência de um grande número de candidatos factíveis à solução, é necessário estabelecer os objetivos a serem otimizados, de modo que existam soluções factíveis de melhor qualidade, ou seja, que maximizam o atendimento dos objetivos, as quais devem ser encontradas por um processo de busca iterativa. Neste trabalho, os objetivos a serem atendidos são os seguintes:

- minimizar o número de vezes em que qualquer participante do torneio jogue r ou mais vezes seguidas dentro (ou fora) de seus domínios, sendo que o valor de r depende de n;

- minimizar a diferença entre o maior e o menor percurso a ser realizado pelos participantes do torneio para completar seus jogos. Uma medida de variância entre os percursos poderia também ser minimizada neste caso.

Caso 2: Definição de Grade Horária em Instituições de Ensino

O problema a ser abordado envolve a alocação de carga didática na definição de uma grade horária semanal em instituições de ensino. Os candidatos à solução serão evoluídos, geração a geração, pela aplicação de operadores genéticos devidamente adaptados ao contexto. Este tipo de problema mostra-se compatível com a estratégia de solução proposta, uma vez que é fácil perceber a explosão combinatória existente, além da necessidade de se atender a múltiplas restrições e objetivos. Em outras palavras, existirá uma grande quantidade de combinações para a formação da grade, sendo que este valor será função do número de professores e do número de aulas concomitantes para a definição do horário. Veja o exemplo simplificado a seguir e a Tabela 1.1, mostrando a explosão combinatória existente.

Exemplo: Considere uma grade horária que tenha um único período de aulas durante todos os cinco dias da semana (de segunda à sexta-feira). Adota-se, para esse exemplo, que 25 professores (*A*, *B*, *C*, ..., *X*, *Y*) devem ser alocados uma única vez durante toda a semana e que o número de aulas em paralelo em um mesmo período seja cinco. A Figura 1.1 mostra uma das soluções possíveis para este exemplo.

Esta é uma versão demasiadamente simples para um problema de alocação de carga didática, pois as únicas restrições a serem atendidas são:

1. disponibilidade de salas de aula;
2. ausência de repetição de professores durante o processo de alocação.

Na prática, como também nos casos a serem considerados neste trabalho, existem outras especificações que devem ser incorporadas ao problema:

- restrições adicionais a serem atendidas:

3. professores com cargas horárias múltiplas e distintas;
4. múltiplos períodos no mesmo dia;
5. atendimento das impossibilidades dos professores em certos períodos;

6. atendimento das demandas dos cursos, no sentido de ser necessário ministrar esta ou aquela disciplina neste ou naquele período, o que implica na necessidade de escolha de um professor vinculado àquela disciplina e não qualquer professor disponível;
7. atendimento da demanda com um número mínimo de professores.

- objetivo a ser otimizado:

1. atendimento das preferências dos professores por certos períodos e também por certas seqüências de períodos (por exemplo, professores que preferem dar aula apenas nos primeiros períodos do dia).

Neste trabalho, somente não serão abordados diretamente os itens 6 e 7 do elenco de restrições, uma vez que a solução elaborada está baseada no atendimento das preferências dos professores e não na demanda dos cursos. Os demais serão todos considerados explicitamente na formulação do problema, direcionando o processo de busca e caracterizando o problema como tendo múltiplos objetivos e múltiplas restrições. Dessa maneira, a estratégia de solução proposta realizará uma busca entre os possíveis candidatos à solução ótima (ou aproximadamente ótima), sabendo que a otimização dos objetivos pode levar à violação de restrições.

Segunda-feira	*W*	*P*	*F*	*R*	*A*
Terça-feira	*E*	*S*	*N*	*K*	*L*
Quarta-feira	*D*	*I*	*G*	*X*	*U*
Quinta-feira	*B*	*V*	*T*	*J*	*H*
Sexta-feira	*Y*	*O*	*C*	*M*	*Q*

Figura 1.1: Representação de uma solução possível.

Tabela 1.1: Número de combinações possíveis, conforme o número de professores, para o exemplo dado.

Número de professores	**Número de combinações**
10	$3{,}6288\times10^{6}$
15	$1{,}3077\times10^{12}$
20	$2{,}4329\times10^{18}$
25	$1{,}5511\times10^{25}$
30	$2{,}6525\times10^{32}$

1.2 Aspectos Gerais da Estratégia de Solução

Para viabilizar a aplicação da estratégia de solução proposta neste trabalho, tanto para o caso 1 como para o caso 2, serão desenvolvidos um código genético compacto e um algoritmo de expansão de código, ambos específicos para cada caso.

Além disso, o desempenho da estratégia de solução baseada na aplicação conjunta de computação evolutiva, busca local e otimização baseada em restrições, será comparada à aplicação da computação evolutiva diretamente ao código genético expandido, a qual também vai apresentar etapas de reparação e procedimentos de busca local.

É fundamental que se garanta a produção de um único e mesmo fenótipo para cada representação compacta, justificando a existência de uma única semente para cada indivíduo e da propriedade de repetitividade do gerador de números pseudo-aleatórios (veja Anexo A), a ser empregado no processo de expansão de código. Uma vez que o código genético compacto é sempre um indivíduo factível e que os operadores genéticos sejam aplicados somente a eles, os descendentes gerados também serão factíveis.

A solução empregada caracteriza-se, ainda, por ser salvacionista e parcialmente elitista. Isso significa que o melhor indivíduo de uma dada geração será preservado para a próxima (sobrevivência do elemento mais adaptado), de modo que não ocorra uma degradação de qualidade do melhor indivíduo ao longo das gerações. Acrescenta-se, que uma porcentagem dos melhores elementos de uma geração estará presente na seguinte, fazendo com que haja uma manutenção das boas características da população, sem comprometer de todo o seu nível de diversidade.

O processo de geração de soluções factíveis, pela aplicação do algoritmo de expansão de código, se dá pela implementação de sofisticados mecanismos de reparação, com convergência garantida. É importante ressaltar que o problema estudado tem características próprias que indicam o uso da computação evolutiva juntamente com processos de busca local, conduzindo aos algoritmos meméticos (MOSCATO, 1989). Sendo assim, além da expansão responsável por produzir um código factível, uma busca local é empregada para refinar o código na direção de ótimos locais. Uma vez que as soluções factíveis da geração corrente tenham sido obtidas e refinadas localmente, uma medida da sua qualidade poderá ser obtida e utilizada nas etapas subseqüentes do processo evolutivo.

1.3 Descrição do Conteúdo dos Demais Capítulos

O Capítulo 2 deste trabalho apresenta os aspectos principais relacionados à computação evolutiva, dando ênfase aos algoritmos genéticos. O Capítulo 3 fornece os conceitos básicos associados aos algoritmos meméticos. No Capítulo 4, é apresentada uma revisão teórica de problemas com restrições sob o enfoque da computação evolutiva, de modo a posicionar o trabalho desenvolvido nesta dissertação. Já o Capítulo 5 enfoca o início dos estudos de caso desenvolvidos, mais especificamente a geração automática de turnos completos em torneios. O tratamento apresentado no Capítulo 5 pode ser considerado como sendo uma etapa preliminar à abordagem do problema de alocação de carga didática em instituições de ensino, a ser tratado no Capítulo 6. O Capítulo 7 apresenta comentários conclusivos e aponta as perspectivas futuras da pesquisa. A seguir, o Anexo A apresenta alguns comentários a respeito de geradores de números aleatórios. O Anexo B mostra o questionário que foi elaborado para o obtenção das preferências dos professores no caso de alocação de carga didática. Por último, o Anexo C mostra o cálculo realizado para a definição do número de candidatos factíveis no caso de turnos completos em torneios.

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.

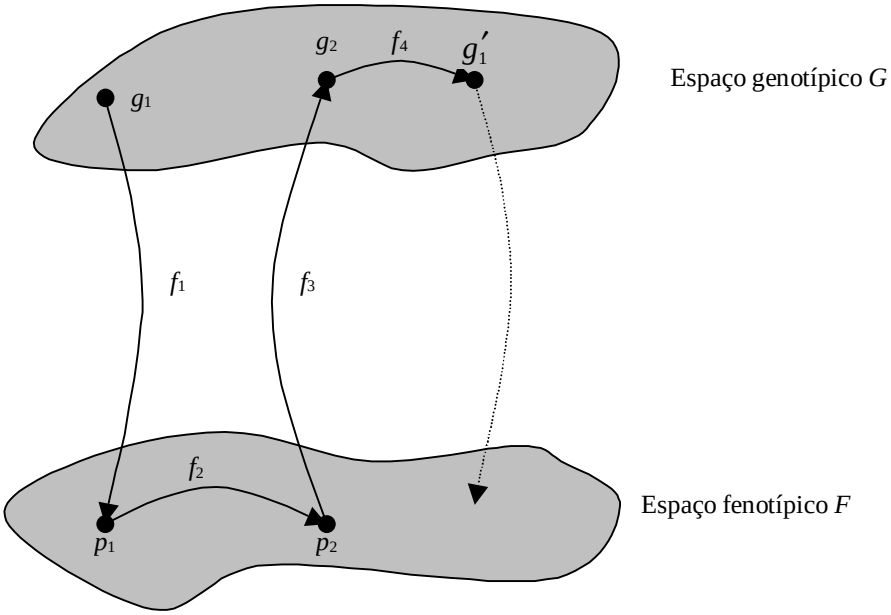


Figura 2.1: Evolução de uma população durante uma geração

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).

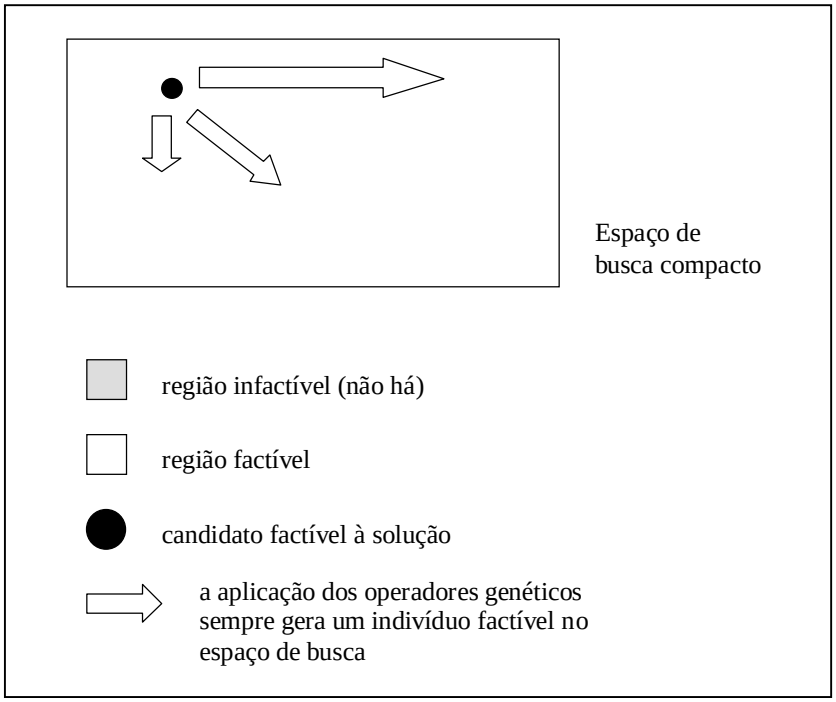


Figura 2.3: Produção de descendentes factíveis após a aplicação dos operadores genéticos em um espaço de busca compacto.

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).

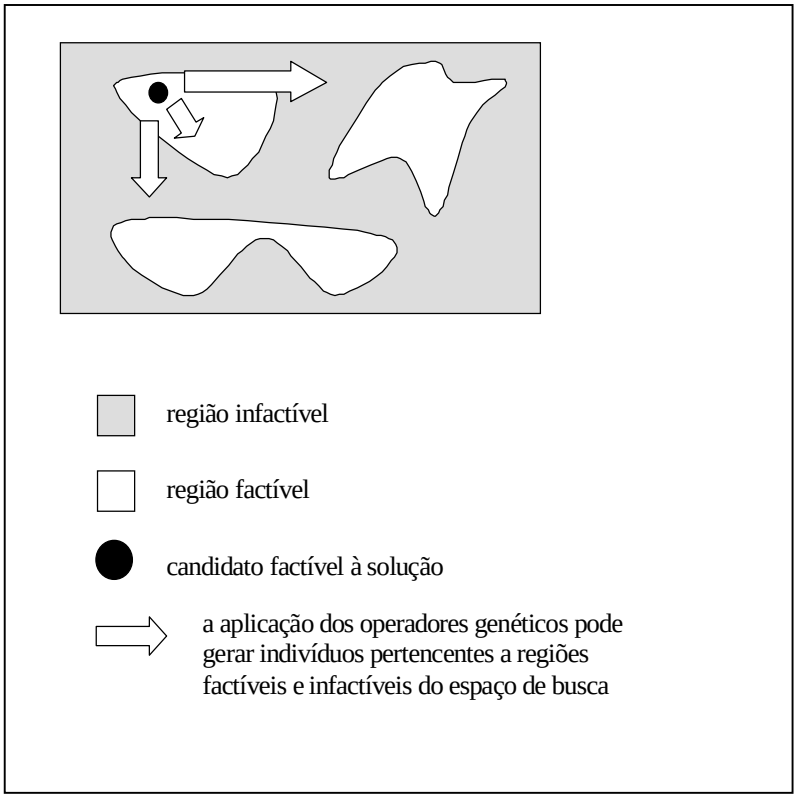


Figura 2.4: Produção de descendentes após a aplicação dos operadores genéticos em um espaço de busca expandido.

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

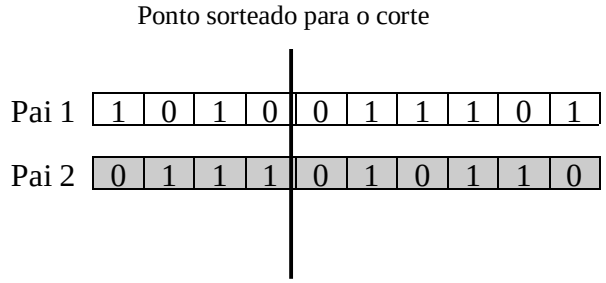
Os operadores genéticos mais freqüentemente utilizados são o *crossover* e a mutação. Nesta seção, são apresentados os principais aspectos relacionados a estes operadores.

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

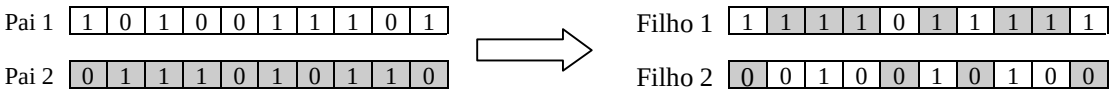
Outros tipos de *crossover* têm sido propostos na literatura. Uma extensão simples do *crossover* de um ponto é o *crossover* de dois pontos. Neste, dois pontos de corte são escolhidos e o material genético será invertido entre eles na posição de ruptura. Observe o exemplo a seguir:

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

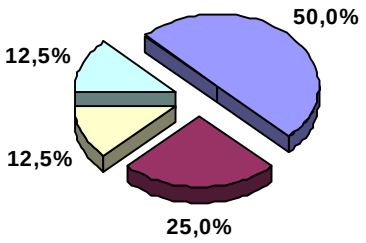


Figura 2.5: Exemplo de aplicação do *Roulette Wheel*: cada indivíduo em uma determinada geração recebe uma probabilidade de passar à próxima geração proporcional ao seu *fitness*, medido em relação ao *fitness* total da população.

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).

CAPÍTULO 3

ALGORITMOS MEMÉTICOS

3.1 Introdução

Os cinco anos em que Charles Darwin atuou como biólogo em sua viagem a bordo do navio Beagle, permitiram que ele coletasse uma grande quantidade de informação e material biológico. Toda essa massa de dados, em conjunto com as suas observações, certamente contribuíram para a formulação das suas idéias sobre os mecanismos da evolução (SILVA & SASSON, 1996). Pode-se inferir que Darwin, ao elaborar a sua Teoria da Evolução, também expôs suas idéias (um conjunto de observações, conhecimento adquirido, genialidade e conclusões) a um processo de seleção. Ele construiu uma "pré-história" sempre confrontando as conclusões obtidas com os fatos observados e com os dados coletados. Este método de trabalho fez com que as características em comum fossem mantidas, e as outras, descartadas. Na verdade, Darwin trabalhava com uma população de idéias, ou em outras palavras, "indivíduos de concepção mental" que passavam por um processo de busca pelo melhor entendimento dos dados coletados. É também provável que, com o conhecimento adquirido ao longo do tempo, ele tenha aperfeiçoado seus mecanismos de busca e seus critérios de avaliação. Este processo de busca pela melhor resposta somente pode ser concebido dentro dos limites do campo do conhecimento, na forma de um processo iterativo aplicado por um indivíduo para a melhoria continuada de suas idéias. Este processo sofisticado de refinamento do conhecimento se justifica no caso de problemas que envolvem um grande número de possibilidades de interpretação e formalização.

3.2 *Memes*

No campo da computação evolutiva, o refinamento do conhecimento pode ser incorporado como uma etapa de apoio ao processo evolutivo. MOSCATO & NORMAN (1992) introduziram o termo "algoritmo memético" para descrever um processo evolutivo que possua uma busca local como parte decisiva na evolução. Essa busca pode ser caracterizada

como sendo um refinamento local dentro de um espaço de busca. O termo *meme* foi idealizado por DAWKINS (1976) como sendo uma unidade de informação que se reproduz durante um processo argumentativo e de transmissão de conhecimento (RADCLIFFE & SURRY, 1994). Portanto, pode-se dizer que, enquanto o algoritmo memético está relacionado com a evolução cultural, o algoritmo genético está baseado na evolução biológica dos indivíduos.

Uma diferença marcante entre genes e *memes* está no processo de transmissão aos seus descendentes. Quando o *meme* é transmitido, ele será adaptado pela entidade que o recebe com base no seu conhecimento e para melhor atender suas necessidades. Quanto aos genes, no processo de evolução eles são transmitidos de uma maneira tal que o descendente gerado vai herdar muitas habilidades e características presentes em seus progenitores. Dessa maneira, o algoritmo genético é inspirado na tentativa de emulação computacional da evolução biológica e o algoritmo memético tenta fazer o mesmo em relação à evolução cultural.

É interessante lembrar que na evolução biológica a informação está codificada nos genes por uma seqüência de nucleotídeos, de modo que a transmissão é influenciada pela presença de mutação, recombinação e pela seleção natural em relação aos indivíduos geneticamente melhor adaptados. Na evolução cultural, a informação envolvida está nos *memes*, de modo que as alterações surgem pela combinação, criação e reorganização das representações mentais (conscientes ou não) e pela possível ineficácia dos mecanismos de transmissão de informação. A replicação (fenótipo) ocorre quando essas representações mentais são transformadas em ações passíveis de imitação ou expressas através de alguma linguagem. A incorporação dessa nova informação por parte de algum indivíduo certamente alterará a pressão seletiva e a influência vinculada às limitações impostas pelo ambiente.

3.3 Algoritmos Meméticos

De acordo com MOSCATO (1999), o uso genérico da denominação de Algoritmo Memético é feito para a identificação de uma classe de meta-heurísticas, constituindo um dos procedimentos de maior sucesso para problemas de otimização combinatória.

A característica principal, presente em muitas implementações que utilizam algoritmos meméticos, é o uso de processos de busca dedicados. Esses processos pretendem

utilizar toda a informação disponível sobre o problema, de modo que este conhecimento seja incorporado sob a forma de heurísticas, técnicas de busca local, operadores especializados de recombinação e muitas outras maneiras.

Em essência, os algoritmos meméticos podem ser interpretados como um conjunto de estratégias que implementam a competição e a cooperação entre diferentes mecanismos de otimização (MOSCATO & NORMAN, 1992). Sendo assim, o sucesso obtido pode ser explicado como sendo uma conseqüência direta da sinergia dos diferentes processos de busca utilizados.

A idéia geral dos algoritmos meméticos é a utilização dos operadores evolutivos que determinam regiões promissoras no espaço de busca, combinados com busca local nestas regiões (este processo tem sido aplicado com sucesso em vários problemas de otimização – MERZ & FREISLEBEN, 1999). Também pode-se dizer que algoritmos meméticos correspondem à união de um método de busca global e uma heurística local aplicada a cada indivíduo, de modo que um algoritmo memético é, na verdade, um tipo especial de *hill-climbing*.

Conforme MERZ & FREISLEBEN (1999), em um ambiente que emprega algoritmo memético, os operadores de recombinação e mutação agem como estratégias de diversificação. Os indivíduos da população podem estar localizados em uma região do espaço de busca contendo um ótimo local, chamada base de atração do ótimo local. Utilizando a informação contida na população, novos pontos de partida podem ser descobertos após a busca local. Os operadores de recombinação e mutação podem gerar indivíduos da população que estejam localizados em bases de atração de ótimos locais ainda não explorados, de modo que um novo pico deva ser alcançado (maximização) ou um vale deva ser explorado (minimização). Utilizando o conceito de superfície de adaptação (*fitness*), a Figura 3.1 ilustra estes eventos, no caso da maximização. Após a recombinação, o filho gerado pode possuir um *fitness* baixo, mas um grande potencial para crescimento, de modo que uma busca local pode levar o descendente a assumir um valor de *fitness* elevado. A mutação, por sua vez, pode levar a um pequeno aumento (ou decréscimo) do *fitness*, por representar uma perturbação local junto à representação do indivíduo.

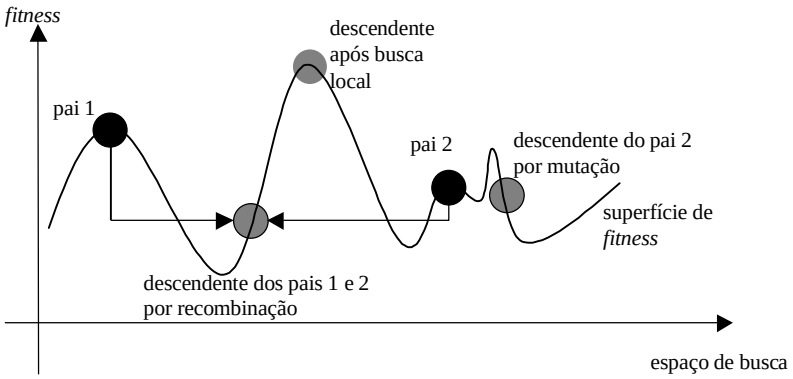


Figura 3.1: Operadores de recombinação e mutação agindo como estratégias de diversificação junto a algoritmos meméticos.

Na presença de restrições, os operadores genéticos de recombinação e mutação podem produzir soluções que estão fora da região factível do espaço de busca. Entretanto, um algoritmo de reparação (factibilização) pode ser elaborado para remeter o descendente gerado para uma região factível (RADCLIFFE & SURRY, 1994). Embora também pudesse ser caracterizado como um processo de busca local, este algoritmo de busca será denominado algoritmo de reparação, a ser descrito no capítulo 4. A razão para a distinção entre uma busca local realizada por um algoritmo memético e a busca realizada por um algoritmo de reparação é que este último não investe no aumento da qualidade da solução, mas apenas na garantia de sua factibilidade.

No caso de problemas de otimização irrestritos, a geração da população inicial é realizada em duas etapas. A primeira, consiste na geração de indivíduos que procuram explorar amplamente o espaço de busca, caracterizando-os como soluções candidatas. Na segunda etapa, procedimentos de busca local são implementados para a obtenção de um ótimo associado à região do espaço em que se encontra cada indivíduo. Esse mesmo processo de busca é empregado após a aplicação de qualquer operador genético (MERZ & FREISLEBEN, 1999), procurando remeter o descendente gerado a um ótimo local.

No caso de problemas de otimização com restrições, já na geração da população inicial podem surgir indivíduos infactíveis, não porque possuam um genótipo mal formado ou porque o algoritmo de solução utilizado contenha erros, mas sim por violarem restrições

impostas ao problema. Há, portanto, a necessidade de algum dispositivo de seleção que seja capaz de discernir entre indivíduos factíveis ou não. Em algumas aplicações, como é o caso deste trabalho, é possível implementar algoritmos de reparação para levar indivíduos à factibilidade (eventualmente à factibilidade mais próxima). Uma vez tendo essa necessidade atendida, é uma conseqüência natural o emprego de uma busca local sobre os indivíduos factíveis, fazendo com que eles possam atingir um ótimo local dentro da região factível. Portanto, resulta como abordagem de solução um algoritmo memético (ou algoritmo genético associado a mecanismos de busca local) aliado a etapas de reparação, tanto para conduzir os indivíduos à factibilidade como para aumentar o seu *fitness*.

3.4 Formalização do Processo de Busca Local

AARTS & VERHOEVEN (1997) consideram a busca local como sendo uma aproximação geral para problemas de otimização combinatória baseados na exploração de vizinhanças (para maiores detalhes veja AARTS & LENSTRA, 1997).

De maneira genérica, um algoritmo de busca local começa com um indivíduo $s_0 \in S$ e tenta continuamente encontrar melhores elementos dentre os vizinhos. Em outras palavras, a finalidade será remeter esse indivíduo para um local onde a função de *fitness* tenha um valor melhor que o atual (Figura 3.1). Essa busca deverá ser realizada até que uma condição de parada seja satisfeita, sendo que essa condição deve ser atendida sempre que o processo de busca não tenha mais a capacidade de melhorar a solução atual. Observe o algoritmo representativo de um processo de busca na Figura 3.2.

```
Procedimento busca local
    início
    repita
        final ← verdade
        para i ← 1 até número de indivíduos faça
            início
            elemento ← indivíduo(i)
            aplicar busca local (elemento)
            se fitness é melhor então final ← falso
            fim
    até final
    fim
```

Figura 3.2: Algoritmo genérico representativo de uma busca local.

MOSCATO (1999) apresenta uma definição formal de busca local que será apresentada a seguir.

Um problema computacional *P* tem domínio de entrada I_p, com $x \in I_p$, podendo ser estabelecido um conjunto $ans_p(x)$ de respostas correspondentes. Entretanto, é preciso garantir que exista um subconjunto $sol_p(x) \subseteq ans_p(x)$ que identifica as soluções factíveis de *P*. Um algoritmo soluciona um problema *P* se, para a entrada $x \in I_p$, apresentar como saída qualquer $y \in sol_p(x)$ – solução factível – ou, no caso de $sol_p(x) = \{\}$, indicando que não existe *y*. A otimização combinatória é um tipo especial de problema de busca, em que cada $x \in I_p$ tem um conjunto $sol_p(x)$ de cardinalidade finita e cada solução $y \in sol_p(x)$ tem um valor de *fitness* $m_p(y, x)$. A busca, nesse tipo de problema, será responsável por encontrar uma solução factível $y^* \in sol_p(x)$ que maximize o *fitness* $m_p(y, x)$.

CAPÍTULO 4

ABORDAGEM EVOLUTIVA PARA PROBLEMAS COM RESTRIÇÕES

4.1 Considerações Iniciais

Conforme EIBEN & RUTTKAY (1997, C5.7), a aplicação de algoritmos evolutivos na solução de problemas de satisfação de restrições (*constraint-satisfaction problem – CSP*) é interessante sob dois pontos de vista. Por um lado, não se pode esperar que um algoritmo clássico de busca possa alcançar com facilidade a solução de um *CSP*. Determinadas heurísticas de busca apresentam-se bastante eficazes em certos casos, mas falham em outros por restringir o escopo da busca. Ainda segundo EIBEN & RUTTKAY (1997, C5.7), algumas instâncias de *CSPs* podem ser resolvidas pela diversificação da busca, pela manutenção de vários candidatos à solução em paralelo e pela aplicação de heurísticas que agreguem mecanismos de construção aleatória de novos candidatos à solução. Como esses princípios são essenciais dentro de algoritmos evolutivos, a sua aplicação para a resolução de *CSPs* apresenta-se promissora.

Por outro lado, algoritmos evolutivos são tradicionalmente utilizados em problemas de otimização que não apresentam nenhum tipo de restrição. Como os operadores convencionais de recombinação e mutação não tratam diretamente restrições, a aplicação de algoritmos evolutivos para problemas com estas características não é imediata. Uma extensão desse fato é a impossibilidade de garantir que pais geneticamente factíveis gerem descendentes também factíveis.

4.2 Definições e Notação Utilizada

A seguir, serão apresentadas algumas definições para posicionar a terminologia utilizada na solução de problemas envolvendo restrições (para maiores detalhes, veja EIBEN & RUTTKAY, 1997, C5.7).

Definição 1: $S = D_1 \times ... \times D_n$ é denominado espaço de busca livre, sendo obtido a partir de um produto cartesiano dos conjuntos D_i, i=1,...,n.

Definição 2: Um problema de otimização sem restrições (*free optimization problem - FOP*) é formado por um par *(S,f)*, sendo *S* um espaço de busca livre e *f* uma função-objetivo em *S*, devendo ser otimizada (minimizada ou maximizada). A solução de um *FOP* é um elemento $s \in S$ com um valor ótimo de *f*.

Definição 3: Um problema de satisfação de restrições (*constraint-satisfaction problem - CSP*) é um par *(S,ϕ)*, sendo *S* um espaço de busca livre e ϕ uma função booleana em *S*. A solução de um *CSP* é um $s \in S$ com $\phi(s)$= verdadeiro.

Definição 4: Um problema de otimização com restrições (*constraint optimization problem - COP*) é uma tripla *(S,f,ϕ)*, sendo *S* um espaço de busca livre, *f* uma função-objetivo em *S* e ϕ uma função booleana em *S*. A solução de um *COP* é um $s \in S$, com um valor ótimo de *f* tal que $\phi(s)$= verdadeiro.

Definição 5: Para o caso de *CSPs* e *COPs*, ϕ será chamada de condição de factibilidade e o conjunto $\{s \in S / \phi(s) =$ verdadeiro$\}$ será o espaço de busca factível.

Por intermédio dessas definições, pode-se representar os problemas *FOP*, *CSP* e *COP* respectivamente pelas seguintes notações: *(S,f,•)*, *(S,•,ϕ)* e *(S,f,ϕ)*, sendo que • implica a ausência do argumento correspondente.

4.3 Transformação de *CSPs* em Problemas que Admitem Solução via Algoritmos Evolutivos

É sabido que a presença de uma função-objetivo (função de *fitness*) para ser otimizada é essencial em algoritmos evolutivos. Pela Definição 3 do item anterior, percebe-se que para um *CSP* (representado pela ênupla $(S,\bullet,\phi)$) isto não ocorre.

Para que um algoritmo evolutivo seja aplicado nesse caso, antes será necessário transformá-lo em um *FOP* - $(S,f,\bullet)$ - ou um *COP* - (S,f,ϕ) -, embora seja sabido que esta transformação nem sempre será possível ou viável.

Definição 6: Considere os problemas P_1 e P_2 como sendo $(S,f,\bullet)$, $(S,\bullet,\phi)$ ou (S,f,ϕ). P_1 e P_2 são equivalentes se:

$$\forall\ s \in S:\ s \text{ é uma solução de } P_1 \Leftrightarrow s \text{ é uma solução de } P_2.$$

Diz-se que P_1 está contido em P_2 se:

$$\forall\ s \in S:\ s \text{ é uma solução de } P_1 \Rightarrow s \text{ é uma solução de } P_2.$$

Assim, a solução de um *CSP* por um algoritmo evolutivo requer que este seja transformado em um *FOP* ou um *COP* que o contém e, em seguida, resolver este novo problema.

Vale o comentário de que *FOPs* permitem uma busca livre, de modo que não apresentam dificuldades para os algoritmos evolutivos. Em contra partida, *COPs* são de difícil solução quando se recorre apenas aos algoritmos evolutivos em sua forma tradicional.

4.4 Espaços de Busca com Restrições

MICHALEWICZ (1997, C5.1) argumenta que muitos dos problemas que apresentam um domínio discreto, tais como *Knapsack problem*, *set covering problem*, *vehicle routing problem* e todos os tipos de escalonamento, *scheduling* e *timetabling*, contêm uma série de restrições que devem ser atendidas simultaneamente.

Acrescenta, ainda, que em geral um espaço de busca *S* consiste de dois subconjuntos disjuntos de subespaços: factíveis (*F* – atendem a todas as restrições simultaneamente) e infactíveis (*U* – violam uma ou mais restrições), sendo que no decorrer do processo de

busca, a população de indivíduos candidatos à solução pode conter elementos que sejam tanto factíveis como infactíveis. A Figura 4.1 ilustra candidatos à solução caracterizados como factíveis, infactíveis e uma solução ótima.

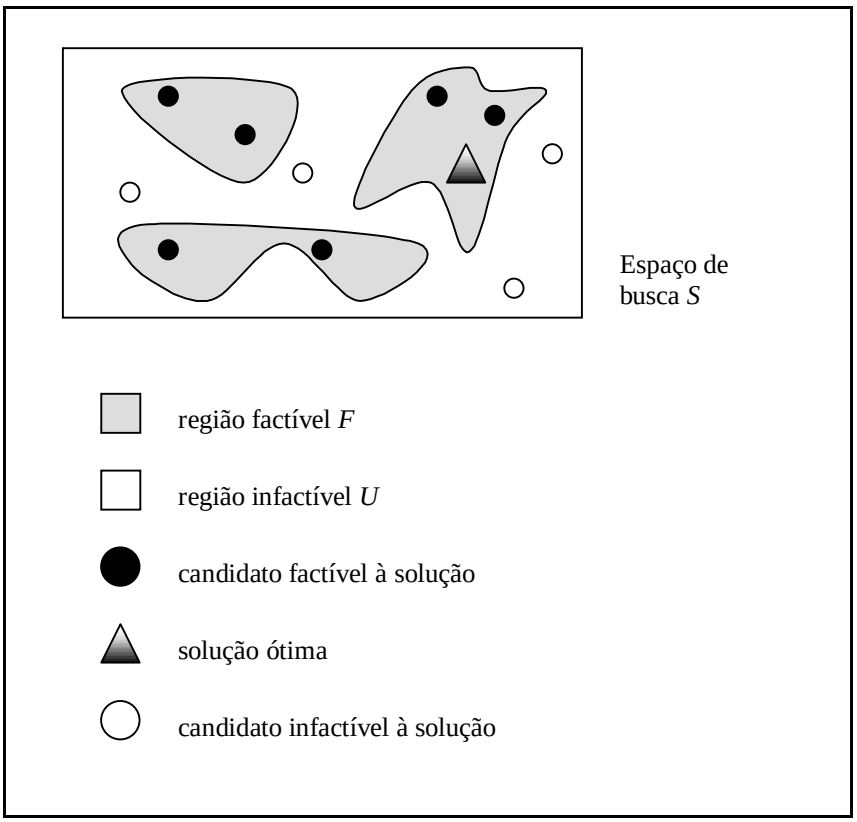


Figura 4.1: Representação pictórica do espaço de busca *S*.

Conforme MICHALEWICZ (1997, C5.1), o problema de como trabalhar com indivíduos infactíveis está longe de ser trivial. De modo geral, duas funções de avaliação devem ser elaboradas: uma delas fará referência ao domínio das soluções factíveis ($eval_f$) e a outra às infactíveis ($eval_u$), tal que:

$$eval_f : F \to \Re \quad \text{e} \quad eval_u : U \to \Re .$$

Dentro desse contexto, há várias técnicas de como trabalhar com espaços de busca restritivos, sendo que cada uma delas apresenta uma abordagem distinta. A seguir, será apresentada uma breve discussão dessas técnicas.

4.4.1 Funções com Penalidade

Segundo SMITH & COIT (1997, C5.2), existem dois tipos básicos de funções de penalidade:

- função de penalidade externa (aplicada às soluções infactíveis);
- função de penalidade interna (aplicada às soluções factíveis).

Para o caso de função de penalidade interna, a principal idéia envolvida é o fato do problema apresentar uma solução ótima tal que pelo menos uma restrição seja ativada, o que implica em dizer que a solução ótima está no limiar entre a factibilidade e a infactibilidade. Conhecendo-se essa característica, uma penalidade será aplicada às soluções factíveis quando a restrição não for ativada. Ressalta-se que, para o caso de múltiplas restrições, esse tipo de implementação torna-se bastante complexa.

Para funções de penalidade externa, há três tipos de solução:

- método no qual nenhuma solução infactível será considerada;
- função de penalidade parcial, sendo que uma punição é aplicada nas proximidades do limite de factibilidade;
- função de penalidade global, sendo que uma punição é aplicada na região de infactibilidade.

Dado um problema de otimização, a formulação a seguir caracteriza-se por ser a mais geral:

$$\min_x f(x) \text{ tal que } x \in A \text{ e } x \in B,$$

sendo x um vetor de variáveis de decisão e tal que restrições $x \in A$ são relativamente fáceis de serem atendidas, enquanto restrições $x \in B$ são relativamente difíceis. Como exemplo, o conjunto A pode representar variáveis contínuas e o conjunto B, variáveis discretas.

Assim, o problema pode ser reformulado como:

$$\min_x f(x) + p(d(x,B)) \text{ tal que } x \in A,$$

sendo $d(x,B)$ uma métrica descrevendo a distância do vetor solução x à região B e $p(.)$ sendo uma função de penalidade monotonicamente não-decrescente tal que $p(0) = 0$. Portanto se $x \in B$, então $d(x, B) = 0$, o que leva à ausência de punição.

Conforme SMITH & COIT (1997, C5.2), várias funções para *p(.)* têm sido estudadas, como também diferentes tipos de métricas para *d(.)*, incluindo uma contagem do número de restrições violadas, distância euclidiana entre *x* e *B*, somatório linear das restrições não atendidas ou somatório de funções exponenciais envolvendo as restrições não atendidas.

Encontrar uma função de penalidade que seja eficiente é uma tarefa difícil. Grande parte dessa dificuldade encontra-se no fato de que a solução ótima freqüentemente está no limiar de uma região de factibilidade. Acrescenta-se, ainda, que ao restringir a busca somente às soluções factíveis, ou ao se impor penalidades muito severas, acaba-se dificultando o encontro da solução ótima. Por outro lado, se a punição não for forte o suficiente, tornará a região de busca extremamente ampla e boa parte do tempo do processo será utilizado na exploração de regiões distantes da região de otimalidade.

4.4.2 Decodificadores

MICHALEWICZ (1997, C5.3) afirma que decodificadores oferecem uma opção interessante para as técnicas de computação evolutiva, sendo que um cromossomo (representação do genótipo) fornecerá instruções para um decodificador ou representará a condição inicial para a elaboração de uma solução factível (representação do fenótipo).

Mais formalmente, um decodificador é uma transformação *T* de um espaço de elementos codificados (onde não há infactibilidade) para uma região de factibilidade do espaço de soluções candidatas (onde há regiões factíveis e infactíveis). A Figura 4.2 mostra um exemplo de decodificação, onde a transformação *T* decodifica um ponto *d* em um representante *s* de um subespaço factível.

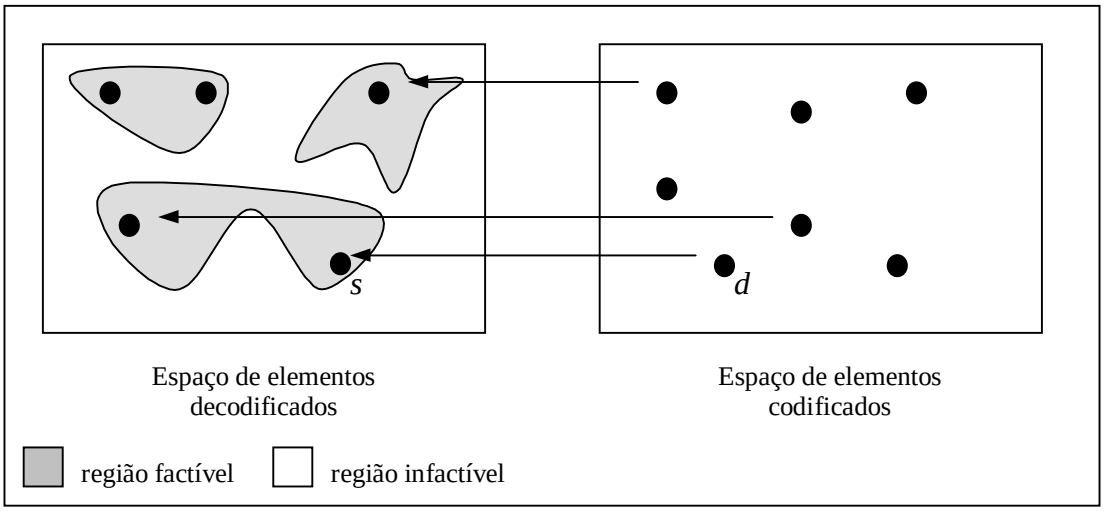


Figura 4.2: Transformação *T* que leva elementos codificados a soluções factíveis.

Será importante considerar que esse processo de decodificação sempre satisfaz as seguintes condições (*F* é o espaço de soluções factíveis):

- para cada elemento $s \in F$, há pelo menos um elemento *d* do espaço de elementos codificados;
- a cada elemento *d* do espaço de elementos codificados corresponde uma única solução factível $s \in F$, ou seja, a transformação *T* é tal que para cada *d* está associado um único $s \in F$.

Também é desejável que a transformação *T*:

- seja computacionalmente rápida;
- deva possuir a característica de que pequenas alterações em elementos do espaço de elementos codificados resultem em pequenas alterações no elemento correspondente do espaço de factibilidade.

4.4.3 Algoritmos de Reparação

Segundo MICHALEWICZ (1997, C5.4), a aplicação de algoritmos de reparação é freqüentemente adotada pela comunidade científica que trabalha com computação evolutiva, acrescentando que para muitos problemas de otimização combinatória é possível fazer com que um indivíduo infactível torne-se factível.

Em relação à função de avaliação, a nova configuração será a seguinte: $eval_u(y) = eval_f(z)$, sendo z a versão factível de y após a aplicação do algoritmo de reparação.

O processo de reparação dos indivíduos pode ser relacionado com a combinação de aprendizado e evolução (efeito *Baldwin* - WHITLEY *et al.*, 1994). O aprendizado é, em geral, uma busca local pela solução factível mais próxima e as modificações ocorridas (o indivíduo infactível é levado para uma região de factibilidade do espaço) serão incorporadas pelo indivíduo ou, então, o indivíduo original não é alterado e um novo elemento será incluído na população com este novo código genético factível.

A desvantagem desse método é a dependência clara em relação ao problema estudado. Em outras palavras, cada problema terá o seu próprio algoritmo de reparação, não existindo um que possa ser generalizado para todos os casos. Acrescenta-se, também, que a heurística utilizada pelo algoritmo deve variar de caso para caso, buscando aquela que melhor se adapta ao contexto.

MICHALEWICZ (1997, C5.4) diz que, para alguns problemas, esse processo de reparação pode ser tão complexo quanto a busca pela solução ótima do próprio problema (em muitos casos, é o que ocorre com problemas de *scheduling*, *timetabling* e outros).

A substituição de indivíduos está relacionada com o lamarckismo (WHITLEY *et al.*, 1994), que assume que um indivíduo passa por um processo de adaptação no decorrer da sua vida no sentido de aumentar a sua adaptabilidade, sendo o resultado deste processo incorporado ao seu código genético. Ainda segundo WHITLEY *et al.* (1994), resultados analíticos e empíricos indicam que estratégias lamarckistas representam um tipo de busca eficaz.

4.4.3.1 Efeito Baldwin

Aprendizado e evolução são considerados processos adaptativos, sendo que o primeiro ocorre durante o ciclo de vida de um indivíduo e é caracterizado pela adaptação a mudanças rápidas no ambiente. Já o segundo processo ocorre durante a história da existência da espécie, sendo caracterizado pela adaptação a mudanças mais lentas no ambiente ou a ambientes estacionários.

Embora seja consenso que não exista interação direta entre aprendizado e evolução, em 1896 James Baldwin propôs um novo fator evolutivo pelo qual características adquiridas poderiam ser herdadas indiretamente. Esse novo fator estaria vinculado ao aprendizado (habilidade que um organismo tem em adaptar-se a seu meio durante a sua vida), sendo denominado de plasticidade fenotípica.

Segundo TURNEY *et al*. (1996), o efeito Baldwin opera em duas etapas distintas:

1. sinergia entre plasticidade fenotípica e evolução: em primeiro lugar, a plasticidade fenotípica permite que um indivíduo se adapte a uma mutação parcialmente bem sucedida (de outro modo seria inútil ao indivíduo). Se essa mutação aumentar a adaptabilidade ao meio, ela tenderá a proliferar-se na população.
2. assimilação genética: ocorrendo a etapa anterior, será dado um certo tempo ao processo evolutivo para encontrar um mecanismo definitivo para a substituição da plasticidade fenotípica.

Como extensões do Efeito Baldwin, percebe-se o seguinte:

- se o aprendizado auxiliar na sobrevivência do organismo, então indivíduos com mais capacidade de aprendizado terão um maior número de descendentes, conduzindo a um aumento da freqüência de genes responsáveis pela habilidade do aprendizado;
- uma vez que o ambiente seja suficientemente estável para que a habilidade mais importante a ser aprendida permaneça a mesma e, ainda, sendo a aprendizagem desta característica custosa (alta pressão seletiva), o processo seletivo poderá indiretamente conduzir à codificação genética desta habilidade (MITCHELL & FORREST, 1995). Em outras palavras, será dado tempo (estabilidade do ambiente) e motivação continuada (custo elevado de aprendizado) para que o processo evolutivo empregue sua criatividade na codificação genética da habilidade desejada. Enquanto essa codificação não ocorre, o indivíduo deverá empregar uma quantidade considerável de recursos para obter esta peculiaridade via aprendizado.
- um comportamento que foi, em um primeiro momento, aprendido pode tornar-se parte de um instinto por intermédio desse processo (desde que seja preservado um nível mínimo de estabilidade no ambiente e que o custo de aprendizagem seja elevado).

De maneira geral, pode-se notar que o efeito Baldwin assemelha-se à teoria lamarckista. Entretanto, no primeiro não há alteração direta do genótipo a partir de modificações do fenótipo do indivíduo. Finalizando, verifica-se que o aprendizado pode afetar a evolução, mesmo quando o que for aprendido não puder ser incorporado diretamente ao genótipo.

4.4.4 Operadores que Preservam as Restrições

Várias pesquisas têm levado à aplicação bem sucedida de operadores genéticos dedicados, tendo a função de preservar a factibilidade dos indivíduos gerados após a sua aplicação (MICHALEWICZ, 1997, C5.5). Percebe-se que essa classe de operadores incorpora um conhecimento específico do problema. Segundo SURRY *et al.* (1995), essa proposta tem a finalidade de construção e utilização de operadores genéticos que tenham a capacidade de atendimento das restrições, de modo que nunca produzam soluções infactíveis.

As principais desvantagens apresentadas por esse processo são:

- operadores genéticos dedicados são aplicáveis somente aos problemas para os quais foram projetados;
- uma análise formal desse tipo de estratégia torna-se difícil, embora uma grande quantidade de experimentos evidenciem a aplicabilidade desta técnica.

4.4.5 Outros Métodos para Trabalhar com Restrições

Várias outras heurísticas para trabalhar com restrições em conjunto com técnicas evolutivas têm sido propostas nos últimos anos. Na maioria das vezes é difícil a classificação desses métodos, uma vez que são a combinação de outros ou são baseados em idéias originais ainda não completamente formalizadas. Dentre essas técnicas podem ser citadas as seguintes (para maiores detalhes veja MICHALEWICZ, 1997, C5.6):

- métodos de otimização multi-objetivo;
- modelos de co-evolução;
- algoritmos meméticos;
- algoritmos genéticos segregados;
- Genocop V (MICHALEWICZ, 2000).

4.5 Abordagens Propostas para o Tratamento de Otimização com Restrições

No desenvolvimento deste trabalho, duas abordagens foram propostas e comparadas no tratamento de problemas de otimização com restrições, particularmente dois problemas de escalonamento. Conforme já apresentado no Capítulo 1, os problemas a serem abordados nos capítulos 5 e 6 caracterizam-se por serem multi-objetivo e por apresentarem múltiplas restrições. De acordo com a nomenclatura adotada neste capítulo, tratam-se de problemas do tipo *COP*: problema de otimização com restrições (*constraint optimization problem*).

A seguir, as duas abordagens utilizadas são apresentadas e posicionadas conforme o contexto deste capítulo (maiores detalhes serão apresentados nos Capítulos 5 e 6).

4.5.1 Codificação Compacta com Algoritmo de Expansão de Código

Na aplicação de algoritmos evolutivos junto a problemas do tipo *COP*, o emprego de uma codificação restrita a um espaço totalmente factível, seguida de decodificação na produção de um elemento factível do espaço original, já foi utilizada na literatura (COSTA, 1999). Essa abordagem será generalizada aqui, com o uso de uma codificação compacta seguida pela aplicação de um algoritmo de expansão de código.

Como é sabido, por um lado, que algoritmos evolutivos em sua composição tradicional encontram dificuldades no tratamento de problemas de otimização com restrições, também é fato, por outro lado, que a aplicação isolada de técnicas de otimização baseada em restrições podem encontrar sérias dificuldades em virtude da existência de ótimos locais de baixa qualidade. Sendo assim, a aplicação conjunta de ambas as metodologias mostra-se promissora pelas seguintes razões:

- o processo evolutivo vai poder atuar somente sobre o código compacto, junto ao qual não há infactibilidade (ou então de modo que a factibilidade possa ser prontamente conquistada), fazendo com que sua eficácia não seja reduzida pela existência de etapas infactíveis;
- embora as técnicas de otimização baseada em restrições, que irão desempenhar o papel de expansão de código, continuem produzindo apenas soluções factíveis com otimalidade local, a existência de uma população de candidatos à solução sendo

evoluída, juntamente com a aplicação de procedimentos de busca local, aumenta significativamente as chances de se obter ótimos locais de boa qualidade.

No entanto, para que esta conjugação de técnicas produza melhores resultados que a aplicação isolada de cada uma delas, é preciso estabelecer critérios para definir o nível de compactação da representação genética, de modo a equilibrar o papel a ser desempenhado por ambas as técnicas. Por exemplo, se a compactação for além do que poderia ser considerado como adequado, o espaço de busca para o processo evolutivo será muito reduzido e o papel da expansão de código passaria a ser muito mais importante. E vice-versa. Existe portanto uma solução de compromisso para o processo de definição do nível de compactação.

Funções de penalidade (seção 4.4.1)

É utilizada uma função de penalidade externa, que não irá permitir nenhuma solução infactível como solução candidata. Este tipo de punição está entre as mais severas, podendo dificultar o encontro da solução ótima caso não venha acompanhada de ferramentas de apoio, como algoritmos de reparação e de busca local.

Decodificadores (seção 4.4.2)

A representação genética compacta irá desempenhar o papel de condição inicial para um algoritmo de expansão de código, que corresponde essencialmente a um decodificador (veja Figura 4.2).

Vale aqui um comentário a respeito da seguinte propriedade que deve estar presente em um decodificador para garantir sua completitude:

- para cada elemento factível $s \in F$ no espaço de elementos decodificados, deve existir pelo menos uma condição inicial no espaço de elementos codificados (representação compacta) de modo que o decodificador seja capaz de produzi-lo.

Como será evidenciado nos capítulos 5 e 6, o algoritmo de expansão de código, que fará o papel do decodificador, não necessita da propriedade de completitude para operar convenientemente, pois existe uma função-objetivo a ser otimizada. Com isso, já durante o processo de expansão de código (e não apenas ao final da decodificação), parte dos

elementos factíveis será descartada por representar um distanciamento da condição de otimalidade. Além disso, ao término da expansão de código, procedimentos de busca local são empregados, de modo que vai haver uma exploração local da região factível, podendo levar a elementos factíveis que melhor atendam aos objetivos, mesmo que estes não sejam diretamente reprodutíveis a partir da decodificação.

O processo de expansão de código pode ser visto como uma busca em uma árvore de decisão, com etapas de *backtracking*. Como a tomada de decisão vai depender da saída de um gerador de números pseudo-aleatórios com a propriedade de repetitividade, então a completitude do decodificador estaria associada à possibilidade de recorrer a todos os possíveis geradores de números psudo-aleatórios, ou seja, a todas as possíveis seqüências pseudo-aleatórias, de modo que qualquer seqüência de ramos da árvore de decisão pudesse ser adotada.

Algoritmos de reparação (seção 4.4.3)

O algoritmo de expansão de código emprega procedimentos de reparação durante o processo de decodificação, e não ao seu final. Portanto, a reparação está incluída no processo de expansão de código. Além disso, procedimentos de busca local são também aplicados imediatamente após o término da etapa de decodificação.

Neste caso, o efeito Baldwin pode representar um papel importante no processo de obtenção da solução, pois é possível optar por condições iniciais ou então geradores de números pseudo-aleatórios que reduzam o custo das etapas de reparação e de busca local.

4.5.2 Codificação Expandida

A busca de solução para problemas *COP* empregando codificação expandida se dá pelos mecanismos usuais de obtenção de indivíduos factíveis via técnicas de otimização baseada em restrições e pela aplicação de operadores evolutivos diretamente sobre o código expandido, seguida da aplicação de procedimentos de reparação e busca local.

São etapas idênticas ao caso anterior, envolvendo expansão de código:

- a técnica de otimização baseada em restrições. No entanto, aqui esta técnica é aplicada uma única vez para cada indivíduo, não necessitando ser reaplicada toda vez que o código genético é alterado, como feito no caso anterior;
- os procedimentos de busca local.

Por outro lado, as diferenças são as seguintes:

- os operadores genéticos aqui só se aplicam ao código expandido, podendo gerar indivíduos infactíveis, enquanto que no caso anterior estavam restritos ao código compacto, gerando sempre indivíduos factíveis. Sendo assim, o genótipo aqui equivale ao fenótipo do caso anterior.
- os procedimentos de reparação aqui só se aplicam sob o código expandido, enquanto no caso anterior eles entram em ação durante a expansão de código.

A expectativa é que o uso de codificação expandida produza técnicas de busca menos eficientes, quando comparadas à técnica de busca baseada em codificação compacta e expansão de código. A razão para tal é simplesmente o fato de não haver uma evolução de condições iniciais para o algoritmo de otimização baseada em restrições, e pelo fato do processo evolutivo estar operando em um espaço com elementos factíveis e infactíveis, o que impede a aceitação de boa parte das operações genéticas propostas, reduzindo em muito a taxa média de progresso junto ao processo evolutivo.

CAPÍTULO 5

CASO 1: GERAÇÃO DE TURNOS COMPLETOS EM TORNEIOS

5.1 Motivação Inicial

Como motivação inicial para o encaminhamento desta dissertação, abordou-se o caso de geração automática de turnos completos em torneios. Esse estudo de caso apresenta-se com características análogas às presentes no problema da produção da grade horária, sendo que ambos possuem múltiplos objetivos e múltiplas restrições, embora as condições de contorno impostas sejam outras (peculiaridades de cada problema). Apresenta, ainda, uma explosão combinatória de candidatos à solução, de modo que uma busca exaustiva pela solução ótima representaria um procedimento computacionalmente intratável.

5.2 Formulação do Problema

Uma das principais contribuições deste trabalho está na solução do problema de representação e na produção de soluções factíveis para o caso específico de geração de turnos completos em torneios de competição, baseados em partidas entre dois participantes adversários (CONCILIO & VON ZUBEN, 2000a; CONCILIO & VON ZUBEN, 2000b), tais como campeonatos esportivos, maratonas, competições escolares, etc. As principais restrições a serem atendidas são:

- cada participante deve enfrentar todos os seus adversários uma única vez durante o torneio;
- todos os *n* participantes, a menos no caso de constituírem um número ímpar, devem jogar a cada rodada.

Portanto, o objetivo é elaborar uma tabela de jogos completa, formando um turno com *n*-1 rodadas, sendo que esta tabela será evoluída geração a geração pela aplicação de operadores genéticos devidamente adaptados ao contexto. O problema apresentado acima mostra-se compatível com o tipo de solução a ser proposta, uma vez que é fácil perceber a

explosão combinatória existente. Em outras palavras, existirá uma grande quantidade de combinações para a formação dos jogos, sendo que este valor será função do número de participantes. Dessa maneira, o algoritmo a ser proposto realizará uma busca entre os possíveis candidatos à solução ótima (ou aproximadamente ótima).

A Tabela 5.1 mostra o número de combinações possíveis para os jogos, de modo a compor um turno completo. Nesse caso, assume-se haver distinção de mando de jogo. Caso o torneio tenha turno e returno, a solução não muda, pois basta produzir o turno, já que o returno seria dado pela inversão simples de mando. O número de turnos completos factíveis para n participantes é dado pela seguinte expressão: $(n-1)!.(n-3)!.(n-5)!.....(n-(n-1))!.\ 2^{(n-1)\times\frac{n}{2}}$ (veja Anexo C). Esta expressão fornece o número de candidatos presentes no espaço de busca da representação expandida.

Tabela 5.1: Número de combinações possíveis para os jogos na composição de um turno completo.

Número de participantes	**Número de combinações**
2	2
4	384
6	$2{,}36\times10^{7}$
8	$9{,}7410\times10^{14}$
10	$4{,}6331\times10^{25}$
12	$3{,}8785\times10^{39}$
14	$8{,}1039\times10^{56}$
16	$5{,}6893\times10^{77}$
18	$1{,}7383\times10^{102}$
20	$2{,}9062\times10^{130}$

O problema de representação compacta foi resolvido a partir da proposição de uma codificação genética original, que utiliza um algoritmo de geração de todas as rodadas (atendendo às duas restrições citadas) apenas a partir da primeira rodada. Portanto, a codificação genética envolve apenas a descrição da primeira rodada. Dessa maneira, a Tabela 5.2 mostra os valores calculados para a definição do número de candidatos factíveis para essa representação genética compacta. Salienta-se que o que importa na primeira rodada são os participantes que a comporão e qual a ordem dos competidores nos pares para a definição dos jogos, indicando quem está no seu domínio ou fora dele. Outro ponto a

ser considerado é que não interessa qual a seqüência de jogos dos pares, tanto na primeira rodada como nas demais.

No exemplo a seguir, para $n = 6$, existirão 6! permutações possíveis, sendo 3! rodadas equivalentes para cada tripla de jogos. Especificamente para o caso dos jogos envolverem os competidores 2 4, 1 6 e 5 3, todas as seguintes rodadas serão equivalentes:

2 4 1 6 5 3
2 4 5 3 1 6
1 6 2 4 5 3
1 6 5 3 2 4
5 3 2 4 1 6
5 3 1 6 2 4

Essas considerações levam à conclusão de que o número de diferentes primeiras rodadas factíveis vai ser dado por: $\frac{n!}{\left(\frac{n}{2}\right)!}$, sendo esta a cardinalidade do espaço de busca na representação compacta.

Tabela 5.2: Número de combinações possíveis para os jogos na composição de uma rodada.

Número de participantes	**Número de combinações**
2	2
4	12
6	120
8	1680
10	$3{,}024\times10^{4}$
12	$6{,}6528\times10^{5}$
14	$1{,}7297\times10^{7}$
16	$5{,}1891\times10^{8}$
18	$1{,}7643\times10^{10}$
20	$6{,}7044\times10^{11}$

Por outro lado, o problema de geração de soluções factíveis pela aplicação de operadores genéticos junto às soluções existentes se dá pela implementação de sofisticados mecanismos de reparação, com convergência garantida.

Dado o número *n* de participantes, serão necessárias *n*–1 rodadas para que as restrições impostas sejam atendidas, de modo que um candidato à solução do problema será formado pelo conjunto das *n*–1 rodadas que satisfazem as condições especificadas.

O *fitness* será calculado pela soma ponderada de dois termos. O primeiro termo da função (FC_1) contabiliza o número de participantes do torneio que realizam *r* jogos consecutivos dentro (ou fora) de seu domínio. O valor de *r* é função de *n*, sendo que nenhum participante pode realizar mais que *r* jogos consecutivos dentro (ou fora) de seu domínio, pois existe uma restrição que impede esta ocorrência. O segundo termo da função-objetivo (FC_2) é uma medida da diferença entre as distâncias máxima e mínima percorridas pelos participantes do torneio considerando todas as rodadas. É desejável que, para completar o torneio, todos os participantes tenham percorrido uma distância parecida, de modo a não privilegiar este ou aquele participante em termos de custo e tempo de deslocamento. Vale lembrar que cada jogo sempre ocorre no domínio de um dos dois participantes.

Dessa maneira, o problema de otimização apresentado pode ser definido como a seguir, sendo dados os valores de *n* e *r*, e assumindo que $x \in CR_n$ (CR_n representa todos os possíveis conjuntos de *n*–1 rodadas):

$$\max_x \quad w_1 \cdot \frac{1}{FC_1(x)} + w_2 \cdot \frac{1}{FC_2(x)}$$

sujeito a:

- $w_1, w_2 > 0$ e $w_1 + w_2 = 1$;
- cada um dos participantes joga contra todos os outros;
- cada participante joga uma única vez em cada rodada;
- para cada participante, a diferença do número de jogos em seu domínio e fora dele é no máximo um;
- número máximo de jogos consecutivos no domínio do participante (ou fora) não é superior a *r*.

5.3 Exemplo ilustrativo

Considere o seguinte exemplo para cálculo de FC_1 e FC_2, adotando $n = 8$, $r = 3$ e $w_1 = w_2 = 0{,}5$. A matriz D contém as distâncias entre o domínio de cada participante em relação ao domínio dos demais.

$$D = \begin{bmatrix} 0 & 0 & 0 & 435 & 435 & 1123 & 1123 & 2783 \\ 0 & 0 & 0 & 435 & 435 & 1123 & 1123 & 2783 \\ 0 & 0 & 0 & 435 & 435 & 1123 & 1123 & 2783 \\ 435 & 435 & 435 & 0 & 0 & 1558 & 1558 & 2491 \\ 435 & 435 & 435 & 0 & 0 & 1558 & 1558 & 2491 \\ 1123 & 1123 & 1123 & 1558 & 1558 & 0 & 0 & 3906 \\ 1123 & 1123 & 1123 & 1558 & 1558 & 0 & 0 & 3906 \\ 2783 & 2783 & 2783 & 2491 & 2491 & 3906 & 3906 & 0 \end{bmatrix}$$

Rodada 1: 4 7 2 6 5 8 1 3
Rodada 2: 6 4 8 3 5 1 7 2
Rodada 3: 2 1 8 6 7 5 3 4
Rodada 4: 4 2 3 7 6 5 1 8
Rodada 5: 8 2 7 6 5 3 1 4
Rodada 6: 3 6 4 8 7 1 2 5
Rodada 7: 6 1 8 7 5 4 2 3

Figura 5.1: Exemplo de candidato factível à solução.

Tabela 5.3: Análise do candidato factível apresentado na Figura 5.1em relação ao *fitness*.

Participante	**Penalidade em relação a *r***	**Distância percorrida**
1	0	2681
2	0	4341
3	0	3218
4	0	2428
5	0	3551
6	0	6152
7	0	6587
8	0	7765
	Somatória= 0	Menor= 2428 e Maior= 7765

O *fitness* é então obtido na forma:

$$\frac{1}{FC_1} = \frac{1}{Somatória + 1} = 1 \qquad \frac{1}{FC_2} = \frac{1}{\frac{Maior}{Menor}} = 0{,}313$$

$$Fitness = w_1 \frac{1}{FC_1} + w_2 \frac{1}{FC_2}$$

$$Fitness = 0{,}5 + 0{,}157 = 0{,}657$$

A penalidade adotada para *r* neste exemplo será sempre acrescida de uma unidade quando, no candidato factível à solução, qualquer participante jogar três ou mais vezes consecutivas dentro ou fora do seu domínio. No candidato à solução apresentado na Figura 5.1 esse fato não ocorre, portanto a penalidade em relação a *r* na Tabela 5.3 será zero.

5.4 Estratégia de Solução

A estratégia de solução permite a elaboração de uma tabela para um número arbitrário de participantes. Outra característica é o fato de que a quantidade de participantes deve ser sempre um número par. Se esse número for ímpar, define-se um competidor fictício, de modo que seu adversário será o participante do torneio que irá folgar naquela rodada.

Para o cálculo do número de jogos necessários, utiliza-se a combinação de *n* participantes dois a dois. Dessa maneira, aplica-se:

$$C_{n,2} = \frac{n!}{(n-2)!2!}.$$

Para exemplificar, tomando o número de participantes igual a 20, resulta:

$$C_{20,2} = \frac{20!}{(20-2)!\,2!} = 190 \text{ jogos}.$$

Como, por hipótese, todos os participantes jogam em todas as $n-1$ rodadas, cada uma delas terá $\frac{n}{2}$ jogos. Para o caso de 20 participantes, serão necessárias 19 rodadas para que todos os jogos sejam realizados, totalizando 190 jogos.

5.4.1 Codificação Compacta e Expansão de Código

A codificação genética utilizada para a composição de um cromossomo em representação compacta (será composto apenas pela primeira rodada), é uma permutação de números entre 1 e *n*. Cada um desses números (genes) representa um participante dentro da tabela gerada. Ressalta-se que essa formação cromossômica compacta vai atender ao elenco de restrições associado a uma rodada específica.

Para a geração da segunda rodada em diante, ou seja, o fenótipo do indivíduo, aplica-se um algoritmo de expansão de código. Para tanto, escolhe-se uma semente do processo de geração aleatória, de modo que todos os outros pares de genes (jogos) sejam produzidos a partir dela. O gerador pseudo-aleatório apresenta a propriedade de repetitividade. Percebe-se, portanto, que a cada primeira rodada gerada está atrelada uma semente, a partir da qual serão codificadas de forma única as demais rodadas. Assim, dada uma primeira rodada e uma semente, o algoritmo de geração da tabela completa vai sempre produzir um mesmo turno, representando o fenótipo associado ao genótipo em representação compacta. É por isso que é dito que a representação genética é compacta, pois a cada conjunto de primeira rodada e semente existe sempre um único turno completo, que vai representar uma solução-candidata factível.

Por exemplo, tomando $n = 16$, começa-se com uma primeira rodada correspondente a uma permutação arbitrária dos números entre 1 e 16, tal como:

16 1 9 11 7 15 4 2 6 10 3 14 12 13 5 8

que dará origem ao seguinte código genético:

16	1	9	11	7	15	4	2	6	10	3	14	12	13	5	8

Esta lista de 16 genes, cada um com um valor distinto, representa um cromossomo, o qual será a entrada para o procedimento de expansão de código, responsável pela geração das demais 14 rodadas. Para tanto, o procedimento de expansão de código irá empregar um gerador de números pseudo-aleatórios que apresente a propriedade de repetitividade, e cada cromossomo contará com um gene adicional que representará a semente da geração (Figura 5.2). Para um mesmo cromossomo e uma mesma semente, sempre irá resultar uma mesma seqüência de rodadas.

16	1	9	11	7	15	4	2	6	10	3	14	12	13	5	8	semente

Figura 5.2: Representação genética compacta (genes e semente).

Após a elaboração da primeira rodada, a produção da seqüência de jogos que comporão as $n-2$ rodadas restantes será realizada com o auxílio de um algoritmo de reparação e outro de busca local. A rotina de reparação será responsável pela aceitação ou rejeição de cada um dos números aleatórios gerados, correspondente a cada gene (participante do torneio).

Tomando como exemplo $n = 16$ e assumindo que a primeira rodada é a apresentada anteriormente, se o gerador de números aleatórios produzir 11 como uma primeira saída, a conseqüência será uma segunda rodada com a seguinte configuração inicial:

11 _ _ _ _ _ _ _ _ _ _ _ _ _ _.

Nesse caso, o único valor que não poderá ser aceito como próxima saída do gerador será 9, pois na primeira rodada já existe o jogo entre os participantes 11 e 9, embora com o domínio invertido.

Com o aumento do número de jogos já definidos, a determinação de uma rodada válida pode se tornar mais custosa computacionalmente, pois aumenta a probabilidade de que a saída do gerador pseudo-aleatório não seja aceita. Este processo de montagem do turno completo a partir da primeira rodada, atendendo a múltiplas restrições, pode ser visto como um processo de busca em uma árvore de decisão, sendo necessário retornar a nós anteriores dessa árvore (*backtracking*) caso não exista possibilidade de progresso a partir das condições existentes. Além desses procedimentos de reparação, mecanismos de busca local são aplicados imediatamente após a conclusão do processo de expansão de código, visando aumentar o *fitness* do indivíduo gerado.

Toda vez que se atinge um folha desta árvore de decisão, a partir do nó raiz, obtém-se uma solução factível. Para um dado gerador de números pseudo-aleatórios, mesmo que seja possível adotar qualquer uma das sementes para iniciar a geração, a propriedade de repetitividade e a conseqüente periodicidade do gerador (veja Anexo A) impedem que exista a possibilidade de se percorrer qualquer ramo factível da árvore de decisão, no caso de haver mais de uma opção. Para que qualquer folha da árvore de decisão, representando

uma solução factível, possa ser obtida como resultado da busca, haveria a necessidade de se recorrer a um elenco de geradores de números pseudo-aleatórios, todos eles apresentando a propriedade de repetitividade. Cada expansão de código seria então realizada por um único gerador do elenco de geradores. Assim, podendo variar a semente e também o gerador, seria possível garantir a completitude do processo de expansão de código.

5.4.1.1 Algoritmo de Expansão de Código para Produção de Soluções Factíveis

Para a codificação compacta, a partir de um conjunto de primeira rodada e semente, será elaborado o restante das rodadas pelo emprego de um algoritmo de expansão de código. Salienta-se, novamente, que todas as rodadas devem sempre atender às restrições apresentadas para a sua formação. Se não forem observadas estas restrições, certamente será gerada uma solução infactível. Dessa maneira, lança-se mão de um algoritmo de expansão de código, basicamente composto dos seguintes passos:

- verificar se o jogo (par de participantes) sorteado, ou o seu inverso, já não faz parte de alguma rodada da tabela;
- se não fizer parte, validar esse par e proceder com o sorteio de outro jogo. Caso já esteja presente em alguma rodada, rejeitar o par e realizar outro sorteio para ocupar essa mesma posição.
- este processo se repete até que sejam atendidas as restrições. Caso se esgotem as possibilidades sem levar ao atendimento das restrições, todos os jogos dessa rodada serão invalidados e uma nova seqüência será sorteada;
- outra situação verificada pelo algoritmo de reparação é o número de vezes que uma mesma rodada é invalidada. Se esse valor exceder um determinado número, todas as rodadas (a menos da primeira) são rejeitadas e todo o processo se repete a partir da segunda rodada.

Como a árvore de decisão possui um número finito de nós, é certo que o algoritmo de reparação vai chegar a uma solução factível, embora não se tenha controle sobre o custo computacional específico associado a cada vez que o algoritmo é aplicado. A convergência

ocorreu em todos os casos simulados, para várias instâncias de número total *n* de participantes.

O algoritmo de expansão de código representa uma etapa fundamental na geração de soluções factíveis a partir de uma representação compacta, sendo aplicado toda vez que a representação compacta sofrer qualquer tipo de modificação. Além disso, no caso da representação expandida, este algoritmo também exerce um papel importante, pois a população inicial, assim como novos indivíduos factíveis que sejam gerados ao longo do processo evolutivo, acabam empregando este mesmo algoritmo para se obter conjuntos de *n*−1 rodadas que sejam factíveis.

5.4.2 Codificação expandida

No caso da codificação expandida, os operadores genéticos vão atuar diretamente sobre o conjunto de *n*−1 rodadas. Para se produzir conjuntos completos de *n*−1 rodadas, o mesmo procedimento de expansão de código adotado na seção anterior será necessário. No entanto, ele somente é aplicado na geração da população inicial e de novos candidatos factíveis a serem evoluídos.

No capítulo 4, todas as principais distinções entre as codificações expandida e compacta foram salientadas. Como aspectos comuns, ambas as codificações fazem uso dos mesmos mecanismos de reparação e busca local, os quais serão detalhados a seguir.

5.4.3 Algoritmo de Busca Local

O processo de busca local é aplicado a todos os indivíduos, sejam eles gerados aleatoriamente para a composição da população ou após a aplicação dos operadores genéticos. O mecanismo é o mesmo tanto para representação compacta como para representação expandida.

A busca local permite a geração de candidatos factíveis e a produção de ótimos locais. Em outras palavras, desde que não haja infactibilidade da solução e para a melhoria do valor do *fitness* de um dado conjunto de *n*−1 rodadas, será possível fazer uma inversão do domínio de cada uma das partidas envolvidas. Outra possibilidade existente é a reordenação das rodadas já definidas.

No caso da representação compacta, todas as vezes em que o procedimento de expansão do código for aplicado em conjunto com o algoritmo de busca local, sua execução a partir de um mesmo código genético compacto levará sempre a um mesmo fenótipo.

5.4.4 Operação do Algoritmo Genético

A descrição que se segue também será válida, com exceção de alguns módulos específicos, tanto para a representação compacta quanto a expandida. Em linhas gerais, o programa calcula o *fitness* de cada indivíduo de uma população inicial gerada. Essa população é ordenada levando-se em conta o *fitness* obtido para, a seguir, receber a aplicação do algoritmo de *Roulette Wheel* e, em seqüência, os operadores genéticos. Novamente a população é ordenada para receber uma seleção parcialmente elitista.

Após a escolha dos melhores cromossomos, novos indivíduos serão gerados para a complementação da população. Dessa maneira, a próxima geração está pronta para sofrer os passos já descritos. Ao término de todas as gerações será apresentada a solução que alcançar o melhor resultado de *fitness* segundo os critérios especificados.

A seguir, são brevemente descritos os módulos que compõem o programa computacional que foi implementado.

- **Mostra_Distâncias**: contém as distâncias entre as localidades dos participantes do torneio.
- **Rodadas_Repetidas**: faz uma busca, verificando se os participantes têm *r* jogos consecutivos dentro do seu domínio (mando de jogo) ou fora.
- **Pop_Inicial**: faz a geração da população inicial.
- **Expansão_Código:** responsável pela geração da segunda rodada em diante (emprega o código genético da primeira rodada mais uma semente).
- **Mostra_Tabelas**: faz a exibição da tabela com todas as rodadas definidas.
- **Mostra_Estatística**: faz a verificação se as rodadas estão distribuídas com equilíbrio entre jogos no domínio e fora, em relação aos mandos de jogos dos participantes.
- **Robin_Hood**: de acordo com os resultados de saída do módulo **Mostra_Estatística**, faz a inversão automática de alguns mandos de jogos para equilibrar a tabela.

- **Verifica_Mando**: faz uma verificação dos mandos de jogo dos participantes. Caso haja *r* consecutivos, a solução candidata sofrerá uma punição (degradação do *fitness*).
- **Verifica_Fora**: faz uma verificação do número de jogos fora do domínio dos participantes. Caso haja *r* consecutivos, a solução candidata sofrerá uma punição (degradação do *fitness*).
- **Fit_Mando**: faz o cálculo da segunda parcela do *fitness* de cada indivíduo (solução candidata) da população, levando em consideração os resultados dos módulos **Verifica_Mando** e **Verifica_Fora**.
- **Fit_Distância**: faz o cálculo do *fitness* de cada indivíduo (solução candidata) da população, levando em consideração as distâncias percorridas pelos participantes quando for necessário viajar até o local da partida.
- **Ordenação**: faz a ordenação decrescente em relação ao *fitness* da população.
- **Guarda_Melhor_Indivíduo**: armazena o melhor indivíduo da população para aplicação do processo de seleção salvacionista na produção da próxima geração.
- **Verifica_Melhor_Indivíduo**: o melhor indivíduo da geração anterior será colocado na atual, caso seu *fitness* seja melhor que o de todos os indivíduos da geração atual.
- **Roulette_Wheel**: aplica o algoritmo da roleta para executar o sorteio dos cromossomos-pais que serão utilizados pelos operadores genéticos.
- **Crossover_Ox**: aplica como operador genético o *order crossover* para a geração dos cromossomos-filhos.
- **Mutação_Inversiva**: aplica como operador genético a mutação inversiva, fazendo a troca de posição entre dois genes de um mesmo cromossomo.
- **Seleção_Elitista**: mantém compulsoriamente um terço dos melhores indivíduos da geração atual na nova geração.
- **Guarda_Melhor_Solução**: armazena a melhor solução gerada a partir do indivíduo com o maior *fitness* de todas as gerações.
- **Recupera_Melhor_Tabela**: faz a recuperação da tabela que contém o maior *fitness* de todas as gerações.

5.4.5 Discussão

É importante ressaltar que o problema estudado tem características próprias que indicam o uso da computação evolutiva, mais precisamente, os conceitos de algoritmos genéticos com busca local, também conhecidos na literatura como algoritmos meméticos (MOSCATO, 1989). Essa abordagem depende, no entanto, do atendimento de restrições de diversas naturezas, conforme já discutido.

O algoritmo de reparação é responsável por validar o conjunto de $n-1$ rodadas, tornando a solução candidata factível. Uma vez que as soluções factíveis da geração corrente tenham sido definidas seguindo os critérios mencionados, uma medida da sua qualidade poderá ser obtida, o que equivale a uma formalização matemática do *fitness* de cada indivíduo. Essa medida pode levar em conta mais de um critério, os quais podem ser ponderados de acordo com a importância atribuída a cada um. Em nosso caso, foram estabelecidos dois critérios igualmente importantes, de modo que os dois valores calculados contribuem igualmente para compor a medida do *fitness* de cada indivíduo.

Em relação aos operadores genéticos (*order* c*rossover* e mutação inversiva) é realizado um sorteio pelo algoritmo da roleta para encontrar os cromossomos que serão modificados. Para essa nova população de soluções candidatas, é calculado novamente o *fitness* (seguindo os critérios já abordados). A seguir, aplica-se a ordenação da população (em função do *fitness*), sendo que a partir deste momento ela é submetida ao processo de seleção parcialmente elitista. É mantido um terço da população atual para a próxima geração e todo o processo descrito se repete até que seja alcançada a última geração.

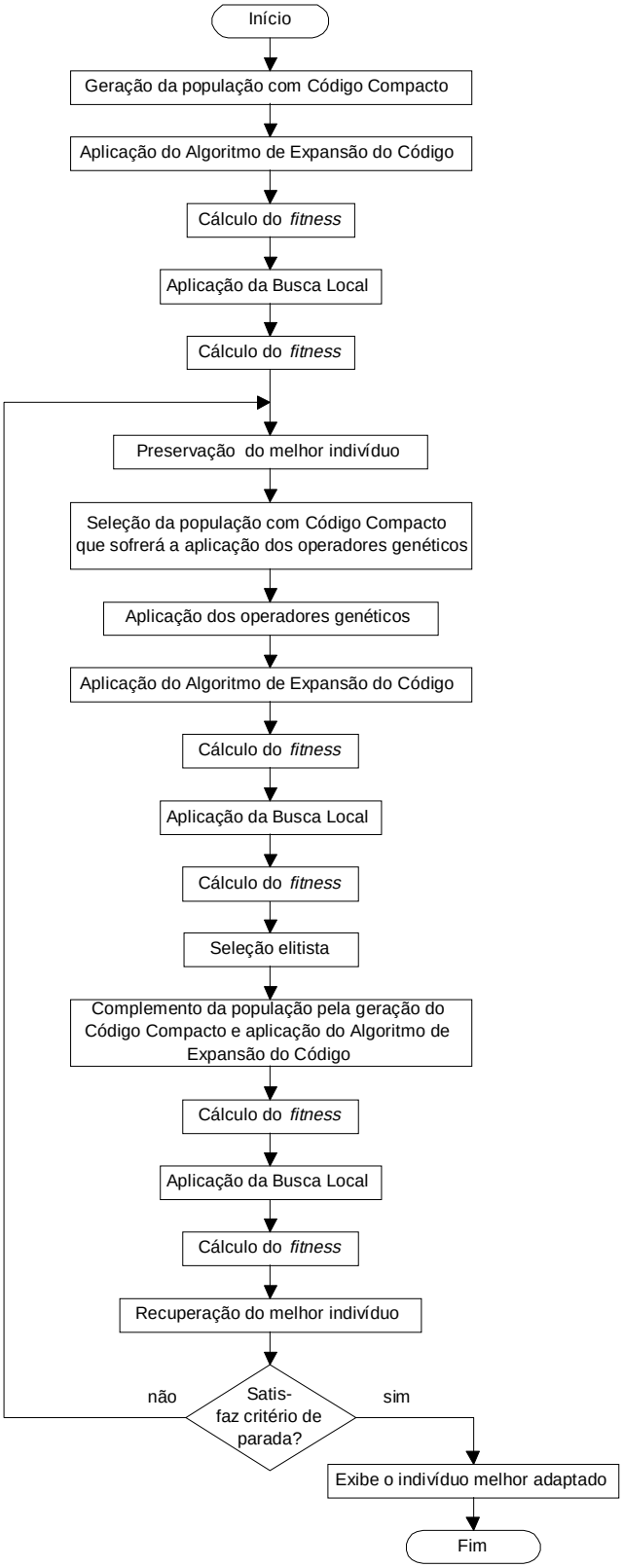
No momento da elaboração da próxima geração, é verificado se o melhor indivíduo da geração anterior está presente na atual. Caso não esteja, ele será introduzido para que não ocorra uma perda do melhor indivíduo e, por conseqüência, uma queda do *fitness*.

Após cada aplicação de operadores genéticos, é ativado o procedimento de busca local. Como resultado final, o programa apresenta a tabela que alcançou o maior valor para o *fitness* e qual é este valor.

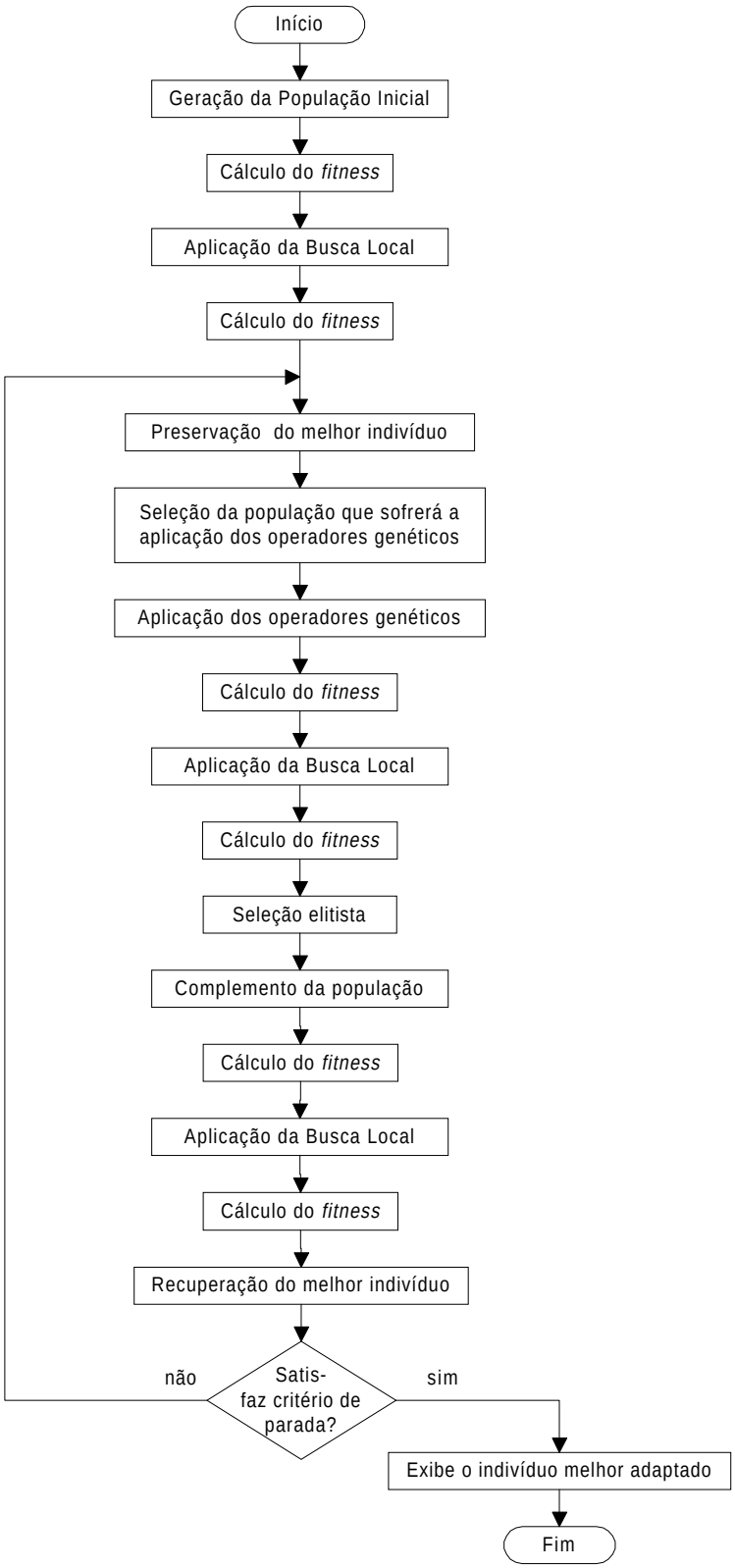
O custo computacional para a geração da população aumenta proporcionalmente aos custos associados ao algoritmo de busca local. Entretanto, resultados muito superiores para o *fitness*, tanto para a codificação compacta quanto para a expandida, foram alcançados

com essa nova metodologia (quando comparados aos valores obtidos sem sua aplicação), por requererem um número reduzido de gerações junto ao processo evolutivo.

5.5 Fluxograma Representativo do Processo Evolutivo com Representação Compacta



5.6 Fluxograma Representativo do Processo Evolutivo com Representação Expandida



5.7 Resultados

Como resultados, apresentam-se na Tabela 5.4 simulações para alguns valores de número de participantes (*n*). O programa computacional foi desenvolvido com a linguagem de programação *Pascal* e os tempos obtidos foram para o processamento em um microcomputador *Pentium* III 500 MHz, com 128Mbytes de memória *RAM*.

Para efeito de comparação com a solução fornecida por um especialista em um problema do mundo real, são apresentados a seguir os resultados obtidos pelo programa desenvolvido neste estudo e a tabela utilizada no Campeonato Paulista de Futebol de 1997, em sua Divisão A1. Utilizou-se a tabela deste ano, porque de 1998 em diante o modo de disputa foi alterado, não atendendo mais à restrição de que cada time deveria jogar contra todos os demais adversários.

Tabela 5.4: Resultados de simulações para diferentes números de participantes (*n*) e diferentes parâmetros.

	Número de participantes	**Número de Gerações**	**Tamanho da população**	**Crossover (%)**	**Mutação (%)**	**Melhor *Fitness***	**Tempo de execução**
1	10	20	40	65	2	0,738	3 min
2	12	20	40	65	2	0,707	19 min
3	12	20	60	65	2	0,717	30 min
4	12	10	40	65	2	0,737	11 min
5	12	20	40	65	4	0,709	13 min
6	12	20	40	80	2	0,718	19 min
7	12	20	40	80	4	0,706	19 min
8	16	20	40	65	2	0,666	1h 08 min
9	16	30	20	65	2	0,629	1h 05 min
10	16	10	30	65	2	0,629	19 min

Nota-se que, ao comparar o *fitness* da Tabela 5.6, elaborada pelo programa, com o da Tabela 5.5, utilizada no Campeonato Paulista de 1997, resultados superiores foram obtidos a partir da aplicação da estratégia proposta neste trabalho. Vale ressaltar que não há conhecimento por parte dos autores deste trabalho acerca do procedimento adotado pelo especialista na produção da Tabela 5.5. Pode ter havido apenas a preocupação da geração de uma solução factível, seguida ou não de etapas de refinamento em busca do atendimento de objetivos, não necessariamente os mesmos adotados neste trabalho. Sendo assim, as duas soluções que estão sendo comparadas podem ter sido obtidas a partir de problemas de otimização com formulação distinta. Como, neste trabalho, o problema foi formulado procurando-se considerar aspectos de grande interesse prático, justifica-se a comparação

realizada com uma solução adotada na prática, independente dos detalhes de implementação desta última.

Tabela 5.5: Solução fornecida pelo especialista (n=16).

***Fitness* 1/FC$_1$= 0,333** ***Fitness* 1/FC$_2$= 0,233**
***Fitness* total= 0,283 (w$_1$=w$_2$=0,5)**
Melhor tabela (solução factível)

Rodada 1=>	1 2	3 4	5 6	7 8	9 10	11 12	13 14	15 16
Rodada 2=>	15 14	4 5	2 8	1 7	6 3	10 11	16 13	12 9
Rodada 3=>	6 4	5 7	8 1	3 2	11 13	14 16	9 15	12 10
Rodada 4=>	4 8	2 5	1 6	7 3	10 14	15 11	16 12	13 9
Rodada 5=>	4 7	5 1	8 3	10 13	11 16	14 9	12 15	6 2
Rodada 6=>	15 10	2 4	5 8	3 1	6 7	11 14	12 13	9 16
Rodada 7=>	3 5	8 6	7 2	1 4	16 10	9 11	14 12	13 15
Rodada 8=>	7 14	2 10	3 12	5 13	4 9	6 15	8 16	1 11
Rodada 9=>	12 2	13 3	9 5	14 4	15 7	16 6	11 8	10 1
Rodada 10=>	2 13	3 9	5 14	7 16	6 11	8 10	1 12	4 15
Rodada 11=>	9 2	14 3	15 5	16 4	11 7	10 6	12 8	13 1
Rodada 12=>	7 10	2 14	3 15	5 16	4 11	6 12	8 13	1 9
Rodada 13=>	15 2	16 3	11 5	10 4	12 7	13 6	9 8	14 1
Rodada 14=>	2 16	3 11	5 10	7 13	6 9	8 14	1 15	4 12
Rodada 15=>	11 2	10 3	12 5	13 4	9 7	14 6	15 8	16 1

Tabela 5.6: Solução fornecida pelo algoritmo genético com representação compacta e busca local (n=16).

número de gerações=20 **tamanho da população=40**
taxa de crossover= 65% **taxa de mutação= 2%**
tempo de execução=1h 08min
***Fitness* 1/FC$_1$= 1,000** ***Fitness* 1/FC$_2$= 0,333**
***Fitness* total= 0,666 (w$_1$=w$_2$=0,5)**
Melhor tabela (solução factível)

Rodada 1=>	16 1	9 11	7 15	4 2	6 10	3 14	12 3	5 8
Rodada 2=>	15 13	10 2	5 9	11 7	6 1	4 16	8 14	3 12
Rodada 3=>	1 3	10 15	9 4	8 16	14 7	12 11	13 6	2 5
Rodada 4=>	6 4	7 8	12 9	11 3	13 14	16 10	15 5	1 2
Rodada 5=>	7 2	14 1	10 13	3 16	8 11	4 12	9 15	5 6
Rodada 6=>	15 8	3 7	5 13	11 16	9 1	4 10	2 14	6 12
Rodada 7=>	11 6	12 5	15 3	13 9	8 2	7 10	16 14	1 4
Rodada 8=>	10 9	13 4	7 6	1 12	16 5	8 3	14 11	2 15
Rodada 9=>	12 2	4 7	9 8	6 16	11 13	5 14	15 1	3 10
Rodada 10=>	7 13	14 9	2 3	4 11	10 12	16 15	5 1	6 8
Rodada 11=>	12 7	1 10	16 9	11 5	14 15	8 4	13 2	3 6
Rodada 12=>	1 8	4 14	9 3	10 11	15 12	2 6	7 5	13 16
Rodada 13=>	3 13	5 4	2 9	12 8	11 1	16 7	14 10	6 15
Rodada 14=>	13 1	5 3	11 2	8 10	7 9	16 12	14 6	4 15
Rodada 15=>	13 8	3 4	12 14	10 5	9 6	15 11	1 7	2 16

Existe uma sensível diferença em relação aos dois valores de penalidade para *r* calculados para as Tabelas 5.5 e 5.6. Para o caso da Tabela 5.6, $\frac{1}{FC_1} = 1{,}000$, indicando que nenhum participante do torneio jogou três vezes consecutivas dentro ou fora do seu domínio. Já para a Tabela 5.5, $\frac{1}{FC_1} = 0{,}333$, o que mostra que houve a participação de alguns competidores em pelo menos três rodadas consecutivas dentro ou fora do seu domínio. Em uma análise mais cuidadosa, pode-se verificar que o participante 3 jogou três vezes consecutivas no seu domínio nas rodadas seis, sete e oito. O participante 10, também nessas mesmas rodadas, fez três partidas consecutivas fora do seu domínio. Essa situação implica que o somatório em relação à *r* será dois, portanto $\frac{1}{FC_1} = \frac{1}{2+1} = 0{,}333$. Caso se procure eliminar diretamente esta seqüência de três jogos consecutivos para os participantes 3 e 10, no domínio e fora dele respectivamente, pela simples troca de mando de jogo, por exemplo, na rodada oito (substituição do jogo 2 10 por 10 2 e do jogo 3 12 por 12 3) resolve-se o problema imediato para os participantes 3 e 10, mas criam-se outros problemas, como:

- a violação da restrição de que cada participante deva ter no máximo a diferença de um entre jogos dentro e fora do domínio;
- com estas trocas, o participante 2 passou a ter três jogos consecutivos fora do seu domínio.

Conclui-se, portanto, que a diferença de desempenho das duas soluções comparadas não pode ser eliminada de forma elementar, fato que conduz à conclusão de que há uma grande diferença qualitativa entre as duas soluções sob comparação.

Analisando-se $\frac{1}{FC_2}$, verifica-se que quanto maior for esta razão, a diferença entre a menor distância percorrida e a maior será um valor menor. Isso implica em dizer que o torneio está mais equilibrado em relação à distância percorrida pelos participantes.

Para o caso da Tabela 5.6, a Figura 5.3 mostra a situação do melhor *fitness* e do *fitness* médio ao longo das gerações. Nota-se na figura apresentada, que já para a primeira geração o melhor *fitness* é maior que o obtido pelo especialista (Tabela 5.5), entretanto este valor ainda será evoluído ao longo das gerações. Este melhor *fitness* obtido já a partir da primeira rodada se deve à adoção de mecanismos de busca local, fazendo com que cada indivíduo que compõe a primeira geração já corresponda a um ótimo local. A distância existente entre o *fitness* médio e aquele produzido pelo melhor indivíduo demonstra a existência de um nível significativo de diversidade na população ao longo das gerações.

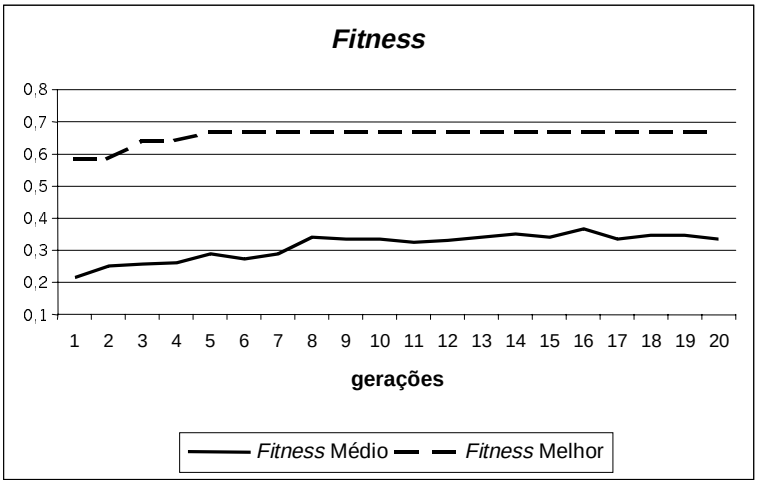


Figura 5.3: Evolução dos *fitness* da média da população e do melhor indivíduo a cada geração, resultando na solução apresentada na Tabela 5.6.

A Tabela 5.7 mostra a média de cinco resultados obtidos em relação ao *fitness* para a codificação genética expandida quando comparada com a compacta. Ressaltando que o termo codificação genética expandida faz referência ao cromossomo que possui no seu genoma todos os genes representativos do torneio. Veja o seguinte exemplo, considerando n=10:

- para a versão compacta, o cromossomo (código genético) terá um tamanho de 10 genes e o seu fenótipo será composto por 90 genes, ou seja, $n \times (n\text{-}1)$;
- para a versão expandida, o cromossomo será composto por 90 genes, sendo que toda esta carga genética será evoluída geração a geração.

Tabela 5.7: Testes comparativos envolvendo a codificação genética compacta e expandida com número de gerações igual a 20 e tamanho da população de 30.

Número de Participantes	**Codificação Expandida**	**Codificação Compacta**
	Fitness médio alcançado	*Fitness* médio alcançado
8	0,598	0,654
10	0,615	0,637
12	0,599	0,607
14	0,612	0,629
16	0,611	0,615
18	0,578	0,600

Analisando-se a tabela anterior, verifica-se que o uso de codificação expandida produz técnicas de busca menos eficientes, quando comparadas à técnica de busca baseada em codificação compacta e expansão de código. A razão para tal é simplesmente o fato de não haver uma evolução de condições iniciais para o algoritmo de otimização baseado em restrições, e pelo fato do processo evolutivo estar operando em um espaço com elementos factíveis e infactíveis, o que impede a aceitação de boa parte das operações genéticas propostas, reduzindo em muito a taxa média de progresso junto ao processo evolutivo.

A próxima tabela mostra o tempo de execução do programa computacional desenvolvido para os mesmos casos da tabela anterior. Os tempos medidos foram obtidos para o processamento em um microcomputador *Pentium* III 650MHz com 128Mbytes de memória *RAM*. O programa computacional foi desenvolvido no ambiente de programação *Delphi*.

Tabela 5.8: Testes comparativos em relação ao tempo de execução do programa com número de gerações igual a 20 e tamanho da população de 30.

Número de participantes	**8**	**10**	**12**	**14**	**16**	**18**
Codificação Compacta	14s	59s	6min 45s	20min 55s	29mim 15s	1h 50min 20s
Codificação Expandida	4s	22s	2min 11s	9min 20s	15min 22s	46min 40s

Nota-se que o tempo de execução para a codificação genética compacta é maior que o da expandida em todos os testes realizados. Justifica-se esse fato pela presença do algoritmo de expansão de código, que faz a geração das demais rodadas do torneio tomando como ponto de partida a primeira rodada e mais uma semente.

CAPÍTULO 6

CASO 2: DEFINIÇÃO DE GRADE HORÁRIA EM INSTITUIÇÕES DE ENSINO

Neste capítulo, a primeira seção apresenta o problema de definição de grade horária em instituições de ensino, mais precisamente um problema de alocação de carga didática. As demais seções tratam das estratégias empregadas para solução do problema e análise dos resultados.

6.1 Apresentação do Estudo de Caso

Uma grade horária de uma instituição de ensino, embora não pareça ou não se perceba, é algo que influi de forma notável na vida de todo o corpo docente, discente e dos funcionários da instituição. Uma vez elaborada e implementada essa grade, ela será válida durante todo o período letivo, o que fará, por várias vezes, com que alunos e professores tenham que se adaptar a ela, já que não necessariamente atende a todos os interesses e disponibilidades dos envolvidos.

As pessoas que recebem o seu horário pronto, acabam não tendo a oportunidade de verificar o trabalho que está envolvido em sua elaboração. Uma técnica utilizada pelos encarregados da sua confecção é, uma vez feita a grade do ano anterior, promover apenas uma atualização para o ano seguinte. Assumindo que realmente exista uma grade anteriormente elaborada, essa técnica mostra-se válida, desde que seja suficiente realizar pequenas adaptações naquilo que já era funcional. Entretanto, somente poderá ser empregada se não existirem alterações profundas nas demandas e disponibilidades. Se isso ocorrer, será necessário começar praticamente do zero, o que certamente consumirá um tempo elevado e sem garantia de se produzir uma solução satisfatória.

Outro ponto a ser considerado são situações como a variação do número de alunos, influindo na inclusão ou exclusão de turmas e, por conseqüência, na necessidade de mais ou menos professores e salas de aula. Percebe-se que o problema de escalonamento encontrado aqui não se limita a este caso, sendo possível citar como outros exemplos os horários de

partida e chegada de aviões, de trens e de ônibus, a elaboração de tabelas de jogos em torneios (CONCILIO & VON ZUBEN, 2000a; CONCILIO & VON ZUBEN, 2000b), discutida no capítulo 5, a criação de cardápios com a quantidade de calorias controlada, a definição de escalas de trabalhadores em empresas com mais de um turno de trabalho e várias outras situações em que existam recursos limitados, demandas a serem atendidas e múltiplas possibilidades de alocação.

Um processo de alocação de recursos sujeito a múltiplas restrições deve fornecer uma solução factível que otimize uma dada função-objetivo, a qual, dependendo do caso em estudo, precisará ser maximizada ou minimizada.

Segundo BEASLEY (1997), escalonamento é uma descrição do ordenamento de recursos no tempo ou no espaço, tentando alcançar com sucesso um determinado objetivo e respeitando o conjunto de limitações impostas. Logo, uma solução factível é aquela que satisfaz todo o conjunto de restrições associadas ao problema.

Uma dificuldade encontrada neste tipo de situação é que, devido às inúmeras particularidades, o processo de solução adotado em um problema geralmente não poderá ser aplicado a outros fins.

Para o estudo de caso deste capítulo, as restrições apresentadas abaixo representam o conjunto de condições que devem ser obedecidas na produção da grade horária:

- um professor somente pode estar em uma única sala em um mesmo período;
- deve ser respeitado o critério do número de períodos de aula do dia e da semana;
- deve ser obedecido o número de salas que podem ser utilizadas durante um mesmo período (admite-se que o número de salas disponíveis é suficiente para atender a demanda requerida);
- deve ser respeitado o número de aulas semanais de cada professor.

A função-objetivo para esse problema deverá ser maximizada e leva em conta as preferências dos professores pelos dias da semana e períodos de aula. É assumida a existência de pelo menos uma solução factível, ou seja, o atendimento das restrições não é mutuamente exclusivo.

A Tabela 6.1 apresenta uma grade horária gerada por meio da codificação expandida e seguindo as especificações apresentadas acima. A simulação foi realizada com

um total de cinco salas de aula por período, dez períodos na semana e envolve um número de vinte professores (designados pelos números de 1 a 20).

Tabela 6.1: Exemplo de grade horária gerada (cinco salas concomitantes) seguindo as especificações apresentadas e conforme carga horária para cada professor indicada pela Tabela 6.2 e grau de preferência mostrado na Tabela 6.3.

2ª feira	Período 1	19	17	15	14	10
	Período 2	19	17	15	14	18
3ª feira	Período 1	13	3	12	6	2
	Período 2	13	3	12	20	18
4ª feira	Período 1	17	11	7	9	5
	Período 2	17	11	7	20	1
5ª feira	Período 1	20	19	8	5	4
	Período 2	20	19	8	18	16
6ª feira	Período 1	18	15	7	5	16
	Período 2	18	15	7	5	4

A Tabela 6.2 mostra o número de aulas semanais de cada professor. Esses valores devem ser obrigatoriamente seguidos para que a grade horária gerada seja factível. A Tabela 6.1 foi elaborada conforme a especificação apresentada na Tabela 6.2 e tenta atender as preferências por período de cada um dos professores, indicada na Tabela 6.3.

Tabela 6.2: Número de aulas semanais de cada membro do corpo docente envolvidos na grade horária.

Professor	**Carga horária semanal**
1	1
2	1
3	2
4	2
5	4
6	1
7	4
8	2
9	1
10	1
11	2
12	2
13	2
14	2
15	4
16	2
17	4
18	5
19	4
20	4

Tabela 6.3: Indica o grau de preferência dos professores em relação a cada um dos períodos de aula.

	Professor	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2ª feira	Período 1	0	0	0	0	0	0	0	0	0	5	0	0	0	5	5	0	5	5	5	0
	Período 2	0	0	0	0	0	0	0	0	0	0	0	0	0	5	5	0	5	5	5	0
3ª feira	Período 1	0	5	5	0	5	5	0	0	0	0	5	5	5	0	0	0	0	5	0	5
	Período 2	0	5	5	0	3	0	0	0	0	0	5	5	5	0	0	0	0	5	0	5
4ª feira	Período 1	5	0	0	0	5	0	5	0	5	0	5	0	0	0	0	0	5	0	0	5
	Período 2	5	0	0	0	3	0	5	0	3	0	5	0	0	0	0	0	5	0	0	5
5ª feira	Período 1	0	0	5	5	5	0	0	5	0	0	5	0	4	0	0	5	0	5	5	5
	Período 2	0	0	5	5	3	0	0	5	0	0	5	0	4	0	0	5	0	5	5	5
6ª feira	Período 1	0	0	0	5	5	0	5	0	0	0	0	0	0	0	5	5	0	5	0	0
	Período 2	0	0	0	5	3	0	5	0	0	0	0	0	0	0	5	5	0	5	0	0

Neste novo exemplo a Tabela 6.4 apresenta uma grade horária gerada por intermédio da codificação expandida e seguindo as mesmas especificações já apresentadas. Para essa nova grade a simulação foi realizada com um total de onze salas de aula por período, doze períodos na semana e envolve um número de quarenta professores (designados pelos números de 1 a 40).

Tabela 6.4: Exemplo de grade horária gerada (onze salas concomitantes) seguindo as especificações apresentadas e conforme carga horária para cada professor indicada pela Tabela 6.5 e grau de preferência mostrado na Tabela 6.6.

2ª feira	Período 1	39	23	40	10	30	29	34	32	17	15	9
	Período 2	36	18	30	32	14	9	29	15	34	17	19
3ª feira	Período 1	20	18	27	5	11	34	40	4	3	2	21
	Período 2	4	2	12	22	21	11	13	34	15	18	33
4ª feira	Período 1	9	5	31	34	26	38	7	40	11	20	17
	Período 2	26	20	40	29	7	1	31	11	17	22	34
5ª feira	Período 1	38	9	39	35	5	19	8	18	1	26	37
	Período 2	13	20	8	4	39	18	1	19	26	29	11
6ª feira	Período 1	7	28	5	18	40	12	15	16	30	36	35
	Período 2	30	40	35	28	12	7	16	3	29	36	26
Sábado	Período 1	39	7	5	33	37	36	25	9	40	4	24
	Período 2	21	24	39	23	7	14	12	19	35	38	6

Tabela 6.5: Número de aulas semanais de cada membro do corpo docente envolvidos na grade horária.

Professores	Carga Horária Semanal
1	3
2	2
3	2
4	4
5	5
6	1
7	6
8	2
9	5
10	1
11	5
12	4
13	2
14	2
15	4
16	2
17	4
18	6
19	4
20	4
21	3
22	2
23	2
24	2
25	1
26	5
27	1
28	2
29	5
30	4
31	2
32	2
33	2
34	6
35	4
36	4
37	2
38	3
39	5
40	7

Tabela 6.6: Indica o grau de preferência dos professores
em relação a cada um dos períodos de aula.

	Professor	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2ª feira	Período 1	5	0	0	0	1	0	0	0	5	5	0	3	4	5	5	0	5	5	5	3
	Período 2	5	0	0	0	1	0	0	0	5	1	3	3	4	5	5	0	5	5	5	3
3ª feira	Período 1	1	5	5	5	5	5	0	0	2	0	5	5	5	0	5	0	0	5	1	5
	Período 2	1	5	5	5	3	0	0	0	2	0	5	5	5	0	5	0	0	5	1	5
4ª feira	Período 1	5	0	4	0	5	0	5	0	5	0	5	0	0	0	1	0	5	0	0	5
	Período 2	5	0	5	0	3	0	5	0	3	0	5	0	0	0	1	0	5	0	0	5
5ª feira	Período 1	5	3	5	5	5	0	0	5	5	0	5	0	4	0	5	5	0	5	5	5
	Período 2	5	3	5	5	3	0	0	5	3	0	5	0	4	0	5	5	0	5	5	5
6ª feira	Período 1	4	0	4	5	5	0	5	0	1	0	0	5	0	0	5	5	0	5	0	2
	Período 2	4	0	5	5	3	0	5	0	1	0	0	5	0	0	5	5	0	5	0	2
Sábado	Período 1	5	0	3	5	5	0	3	0	1	0	0	5	0	0	5	0	0	0	0	0
	Período 2	3	0	4	5	4	0	4	0	1	0	0	5	0	0	5	0	0	0	0	0

	Professor	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
2ª feira	Período 1	0	0	5	0	0	0	2	0	5	5	0	5	0	5	0	2	0	0	5	5
	Período 2	0	0	5	0	0	0	2	0	5	5	0	5	0	5	0	5	0	0	0	5
3ª feira	Período 1	5	0	0	0	0	5	5	0	5	0	0	5	1	5	0	0	0	5	0	5
	Período 2	5	5	0	0	0	5	5	0	5	0	0	5	1	5	0	0	0	3	0	5
4ª feira	Período 1	0	0	5	0	0	5	3	0	5	2	5	0	0	5	0	0	0	5	0	5
	Período 2	0	5	5	0	0	5	2	0	5	2	5	0	0	5	0	0	0	3	0	5
5ª feira	Período 1	0	0	0	0	0	5	4	0	1	0	0	0	0	0	5	0	0	5	5	0
	Período 2	0	0	0	0	0	5	4	0	5	0	0	0	0	0	5	0	0	3	5	0
6ª feira	Período 1	0	0	0	0	0	0	5	5	5	5	0	0	0	0	5	5	0	0	0	5
	Período 2	0	0	0	0	0	0	5	5	5	5	0	0	0	0	5	5	0	0	0	5
Sábado	Período 1	0	0	3	5	5	0	4	0	1	0	0	0	5	0	0	5	5	0	5	0
	Período 2	5	0	5	5	5	0	4	0	1	0	0	0	5	0	0	5	5	0	5	0

Além da preferência dos professores por determinados períodos, outras situações poderiam ser consideradas, como o tipo de disciplina a ser ministrada, a divisão da grade horária por matérias afins (biológicas, exatas e humanas), restrições de disciplinas em seqüência, a necessidade de salas especiais (por exemplo com recursos de mídia) e várias outras condições que certamente afetariam a grade horária. De modo similar ao adotado no capítulo anterior para o caso de um número ímpar de participantes, caso haja mais salas de aula do que carga didática de professores, basta criar a figura de um ou mais professores fictícios que absorvam toda a carga horária não ocupada.

Outra questão a ser considerada é o caso de que, embora existam heurísticas refinadas para se construir manualmente uma instância das tabelas 6.1 e 6.4, trata-se de um problema de explosão combinatória e que certamente vai ganhar soluções de melhor

qualidade caso se implemente em computador processos automáticos e eficientes de busca no espaço de soluções candidatas.

WEARE (1995) diz na conclusão de sua tese de doutorado que os algoritmos genéticos representam um caminho promissor para a solução de uma grade horária. Destaca as suas características de busca, a capacidade para trabalhar com heurísticas específicas, a incorporação de restrições com garantias de factibilidade dos indivíduos e o fato de ser um algoritmo baseado na população. Todas essas peculiaridades permitem que sejam geradas várias grades horárias, representando potenciais candidatos à solução.

NEWALL (2000) investigou e desenvolveu vários métodos híbridos aplicados à problemas de grade horária. Fez uma comparação entre esses algoritmos, discutindo as suas vantagens e desvantagens. Utilizou a combinação dos seguintes algoritmos:

- algoritmos evolutivos;
- busca local;
- quatro diferentes heurísticas:
  - com operadores genéticos;
  - com elementos aleatórios;
  - para a inicialização da população;
  - decomposição do problema em outros problemas de mais fácil solução.

Acrescenta, ainda, que há outras áreas que deveriam ser abordadas em trabalhos futuros, comentando que há a necessidade de métodos para o tratamento das restrições que estão envolvidas em problemas de grade horária.

6.2 Codificação

A codificação de um indivíduo por um cromossomo é um dos pontos principais que determinam o sucesso ou o fracasso dos algoritmos genéticos. Uma codificação competente deve constar das seguintes características (PALMER, 1994):

- ao aplicar o operador de cruzamento, os descendentes gerados devem ser caracterizados como soluções candidatas válidas (factíveis);
- a representação adotada deve permitir a exploração de todo o espaço de soluções candidatas, não podendo restringir o processo de busca.

Conforme discutido no capítulo 4, a primeira característica citada revela-se bastante crítica para problemas de otimização com restrições, notadamente naqueles em que a solução possui uma estrutura particular e, por conseqüência, difícil de ser mantida após a recombinação para a formação dos descendentes. Para contornar esse fato, são possíveis quatro alternativas:

- descartar os descendentes inválidos e repetir a seleção dos pais, gerando novos filhos até que sejam factíveis;
- atribuir um valor de adequação baixo aos cromossomos que representam soluções inválidas (tal procedimento forçará a sua provável eliminação pelo processo de seleção);
- transformar as soluções inválidas em factíveis aplicando rotinas de factibilidade (por exemplo, algoritmos de reparação);
- utilizar operadores genéticos dedicados, de modo que somente possam ser gerados descendentes viáveis.

Uma das contribuições deste trabalho, também descrita no capítulo 4 e já empregada no capítulo 5, é a apresentação de uma quinta alternativa para contornar o problema de factibilidade de descendentes após a aplicação do operador de cruzamento. Ela é apresentada a seguir:

- utilização de uma codificação genética compacta (veja Figura 6.1) aliada a algoritmos de decodificação e reparação para a expansão do código, de modo que o código gerado seja sempre factível. Aplicação dos operadores genéticos apenas sobre o código compacto produzindo descendentes também factíveis. Por fim, emprego de um procedimento de busca local para aumentar a adaptabilidade da solução factível gerada.

6.3 Base de Dados

A base de dados deve conter o conhecimento prévio da carga horária de cada professor envolvido e da sua respectiva disponibilidade. Para o escopo deste trabalho, o conceito de disponibilidade envolve o dia da semana e o período de aulas que é da preferência do professor. Para formalizar essa preferência foi apresentado um grau de

satisfação graduado de 0 a 5, de modo que o maior grau indica plena satisfação e o menor mostra a insatisfação máxima com aquele determinado período (veja Anexo B).

6.4 Geração da População Compacta e Semente

Obedecendo aos critérios de formação para os cromossomos, uma população inicial com o código genético compacto será gerada. O código compacto caracteriza-se por ter apenas um número reduzido de genes quando comparado a uma solução completa. Mais precisamente, esse código é formado pelo número de genes que representam dois períodos para alocação de professores (veja o exemplo da Figura 6.1, que considera uma grade horária com onze salas concomitantes por período). Como parte integrante dessa codificação compacta, será escolhida (também aleatoriamente) uma semente para cada um dos indivíduos da população (veja o exemplo da Figura 6.1).

A geração da população inicial está também associada a mecanismos de reparação utilizados para factibilizar cada indivíduo. No entanto, como os cromossomos são de tamanho reduzido, esta etapa de reparação apresenta um custo computacional baixo.

período 1	39	23	40	10	30	29	34	32	17	15	9	
período 2	36	18	30	32	14	9	29	15	34	17	19	semente

Figura 6.1: Exemplo de codificação compacta para um candidato à solução da população.

6.5 Procedimento de Expansão do Código

Quando da expansão de código desta representação genética compacta, todos os outros genes que deverão compor uma solução completa serão gerados a partir da semente (ela é característica de cada indivíduo) e de um gerador pseudo-aleatório que apresente a propriedade de repetitividade. É claro que a expansão da população deve seguir criteriosamente o conjunto de restrições para a sua formação, requerendo a aplicação de algoritmos de reparação, de modo que o cromossomo gerado seja um candidato factível à solução. Outro aspecto que será levado em conta no momento da geração aleatória é a preferência de cada professor. Sendo assim, professores com maior preferência por certos períodos terão maior probabilidade de serem selecionados. Com isso, a árvore de decisão a

ser percorrida pelo algoritmo de expansão de código é polarizada, já que alguns ramos têm prioridade sobre os demais associados a um dado nó da árvore.

Assim, com o conhecimento da base de dados e do código compacto de cada indivíduo da população, será possível fazer a expansão de código que irá levar a uma grade horária completa. Esta expansão do código deve ter como característica primordial a geração de indivíduos factíveis e, ainda, um único indivíduo a partir de cada semente selecionada.

No contexto do caso em estudo, para que um indivíduo (cromossomo) seja factível é necessário que cada gene (representando um professor) apareça somente uma vez em um mesmo período da grade horária e, ainda, que cada professor esteja representado nesse cromossomo de modo a atender sua respectiva carga horária. Por exemplo, um professor nunca poderá ter carga horária de uma aula e a ele serem atribuídas três (ou seja, o gene correspondente aparece três vezes no mesmo cromossomo). Observe o exemplo das Tabelas 6.7 e 6.8 representando a carga horária gerada de maneira incorreta para os professores *A* e *G*, acarretando na infactibilidade do respectivo cromossomo.

Tabela 6.7: Carga horária estabelecida para os professores.

Professor	Carga Horária
A	**1**
B	2
C	2
D	2
E	3
F	2
G	**3**

Tabela 6.8: Carga horária incorreta produzida pela solução candidata.

Professor	Carga Horária
A	**3**
B	2
C	2
D	2
E	3
F	2
G	**1**

As Figuras 6.2, 6.3 e 6.4 apresentam exemplos de cromossomos factíveis e infactíveis elaborados conforme a carga horária indicada pelas Tabelas 6.7 e 6.8.

A	*B*	*C*	*D*	*E*	período 1
A	*C*	*D*	*E*	*F*	período 2
A	*B*	*E*	*F*	*G*	período 3

Figura 6.2: Exemplo de cromossomo infactível,
levando à carga horária da Tabela 6.8.

A	*B*	*C*	*D*	*E*	período 1
G	*C*	*D*	*E*	*F*	período 2
G	*B*	*E*	*F*	*G*	período 3

Figura 6.3: Exemplo de cromossomo infactível (o mesmo professor
está designado mais de uma vez para o mesmo período).

A	*B*	*D*	*E*	*G*	período 1
G	*C*	*D*	*E*	*F*	período 2
G	*B*	*E*	*F*	*C*	período 3

Figura 6.4: Exemplo de cromossomo factível conforme a Tabela 6.7
(gerado seguindo as leis de formação especificadas).

Para que a população inicial seja gerada com a garantia de factibilidade dos indivíduos e com a melhor qualidade possível, utilizam-se algoritmos de reparação e de busca local. O primeiro deles faz a verificação da repetição ou não de um determinado professor no mesmo período. Se ele já estiver presente, será rejeitado e outro gene sorteado. Verificação análoga ocorre em relação ao atendimento da carga didática.

O segundo executa uma busca por um ótimo local. Tenta-se fazer uma melhoria do *fitness* associado à grade horária, pela troca de professores entre períodos adjacentes de um mesmo cromossomo. No entanto, para que essa troca seja possível será necessária a utilização de um procedimento de validação, verificando a possibilidade da troca. Ou seja, será necessário que os genes possam ser invertidos, sem ocasionar a infactibilidade do cromossomo.

A Figura 6.5 apresenta um exemplo de busca local que pode ser realizada, uma vez que o professor *B* do período 1 e o professor *F* do 2, podem ser trocados sem que haja

infactibilidade do cromossomo. Em outras palavras, o professor *F* somente fará parte do período 1 após o procedimento de busca local. O mesmo acontecendo com o professor *B* em relação ao período 2.

Situação anterior à busca local:

A	*B*	*C*	*D*	*E*	período 1
A	*C*	*D*	*E*	*F*	período 2

Situação posterior à busca local:

A	*F*	*C*	*D*	*E*	período 1
A	*C*	*D*	*E*	*B*	período 2

Figura 6.5: Exemplo de busca local que pode ser realizada.

A Figura 6.6 apresenta um exemplo de busca local que não pode ser aceita, uma vez que, caso o professor *C* do período 1 e o professor *F* do 2 sejam invertidos, ocasionará a infactibilidade do cromossomo (professor *C* estará duas vezes no mesmo período). Ou seja, o professor *C* já está presente no período 2 antes da realização do procedimento de busca local. Caso seja invertido de posição com o professor *F*, o *C* estará presente duas vezes no período 2, ocasionando a infactibilidade do cromossomo.

Situação anterior à busca local:

A	*B*	*C*	*D*	*E*	período 1
A	*C*	*D*	*E*	*F*	período 2

Situação posterior à busca local:

A	*B*	*F*	*D*	*E*	período 1
A	*C*	*D*	*E*	*C*	período 2

Figura 6.6: Exemplo de busca local que não pode ser realizada.

Após todo o processo de elaboração da população inicial, esta será evoluída geração a geração até alcançar o seu valor final. Todos os processos definidos a partir da seção 6.6 serão executados no decorrer dessa evolução.

6.6 Procedimento de Sorteio

O procedimento de sorteio é realizado por intermédio de roletas. A utilização desse método de sorteio faz com que os elementos de maior probabilidade sejam priorizados, embora todos os indivíduos estejam envolvidos nesse processo.

A primeira roleta é elaborada levando-se em consideração graus de satisfação dos professores em relação a um determinado período. Uma vez montada a roleta, será escolhido um valor aleatoriamente. O próximo passo será verificar a qual faixa da roleta o valor sorteado pertence, de modo que o intervalo encontrado indicará o grau de satisfação. A partir desse valor será elaborada uma segunda roleta para o mesmo período, levando em consideração a carga horária dos professores com no mínimo o grau de satisfação sorteado pela primeira roleta. Uma vez que a segunda roleta esteja montada, novamente será sorteado um valor aleatório e procurado a qual faixa ele pertence. O intervalo encontrado representa um professor que será alocado no cromossomo (se atender aos critérios de restrição local e global já mencionados) no período que está sendo tratado.

A cada nova iteração do processo, as duas roletas serão refeitas de modo que os elementos já alocados no cromossomo terão a sua probabilidade de sorteio diminuída ou zerada (se já estiverem com toda a carga horária designada). Portanto, há uma clara prioridade para professores com preferência alta e carga didática ainda pouco atendida.

O exemplo a seguir mostra a elaboração das roletas de um período qualquer utilizadas no sorteio dos elementos que comporão o cromossomo.

Exemplo: São considerados 20 professores divididos entre as preferências apresentadas na Figura 6.7. Para cada grau de preferência, é atribuído um determinado valor de peso. Com base nesses dados é construída a primeira roleta. Considere o valor R_1=0,974611 sorteado aleatoriamente, o que indica que ele pertence ao último intervalo da primeira roleta (intervalo associado ao grau de preferência 5). A partir desse valor será construída a segunda roleta, conforme mostra a Figura 6.8. Considere, agora, R_2=0,24135 sorteado aleatoriamente. Esse valor pertence ao segundo intervalo da nova roleta, indicando que o professor escolhido será o de número 2 (coincidentemente é aquele com menor probabilidade de ser escolhido).

Período 1 da 2ª feira

Número de professores	Preferência	Número professores x Peso	Probabilidade da roleta
5	0	5	5/70
1	1	2	2/70
4	2	12	12/70
3	3	12	12/70
3	4	15	15/70
4	5	24	24/70

Somatória= 70

Preferência	Peso
0	1
1	2
2	3
3	4
4	5
5	6

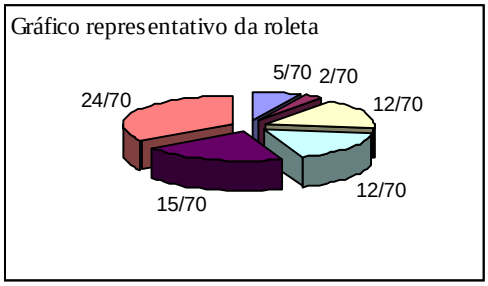


Figura 6.7: Elaboração da primeira roleta.

Período 1 da 2ª feira e Número de professores= 4

Professor	Preferência	Carga Horária	Probabilidade da roleta
1	5	2	2/10
2	5	1	1/10
3	5	4	4/10
4	5	3	3/10

Somatória= 10

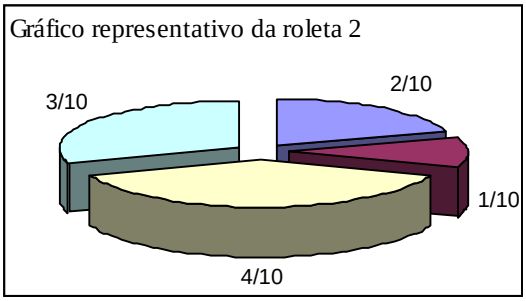


Figura 6.8: Elaboração da segunda roleta.

6.7 Preservação do Melhor Indivíduo

Para evitar a perda do indivíduo mais adaptado ao meio (aquele que apresenta o maior *fitness*), este cromossomo será preservado a cada geração, já que existe uma probabilidade, embora baixa, de que ele seja perdido. Sempre se fará a seguinte verificação: se o melhor indivíduo da geração anterior possuir um *fitness* maior que o melhor indivíduo da geração atual, ele será introduzido nessa geração; caso contrário será preservado o da geração atual. Esta estratégia evita a degradação do *fitness* do melhor indivíduo da população, mantendo o elemento melhor adaptado no decorrer do processo evolutivo, até que surja um elemento que o supere.

6.8 Operadores Genéticos

Para a aplicação dos operadores genéticos, será necessário antes de mais nada fazer uma seleção pelo algoritmo de *Roulette Wheel* dos cromossomos envolvidos (cromossomos-pai que, pela combinação da sua carga genética, irão gerar seus descendentes). Pela utilização da roleta, sabe-se que a probabilidade de seleção de um cromossomo (codificação genética da solução candidata) é diretamente proporcional ao seu valor de *fitness*. Isto significa que os indivíduos mais adaptados (melhor *fitness*) têm uma probabilidade maior de repassar a sua carga genética aos seus descendentes (terceira conclusão do darwinismo).

Quando a aplicação dos operadores genéticos é feita sobre o código compacto, uma vez que esse código pertence a um espaço de dimensão reduzida onde a aplicação de algoritmos de reparação é mais simples, o descendente gerado após a aplicação da recombinação e da mutação também será caracterizado por ser um candidato factível à solução (Figura 6.9).

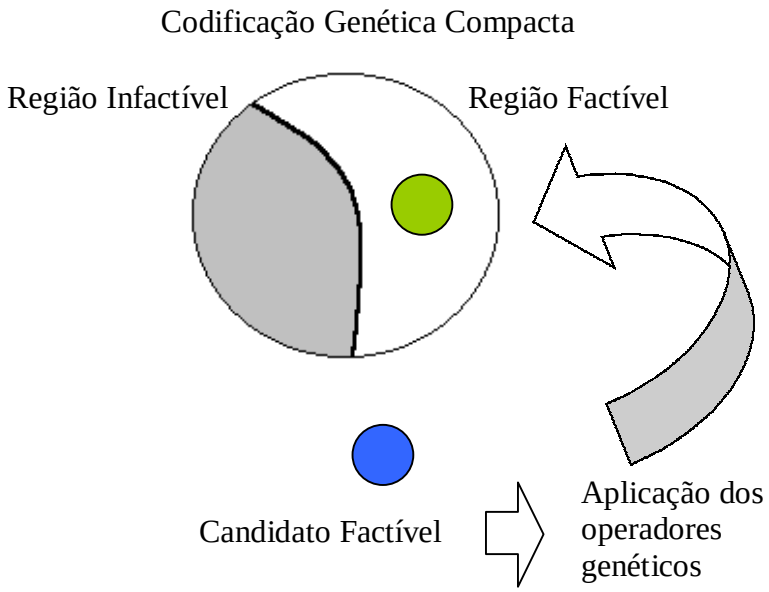


Figura 6.9: Factibilidade dos descendentes após
a aplicação dos operadores genéticos.

Por outro lado, se os operadores genéticos forem aplicados sobre o código já expandido, não há nenhuma garantia de que o descendente gerado seja um indivíduo factível. Isso significa que o elemento gerado poderá ser remetido para uma região de infactibilidade, de modo que se isso ocorrer, mais recursos computacionais serão necessários para trazê-lo para uma região de indivíduos factíveis (Figura 6.10).

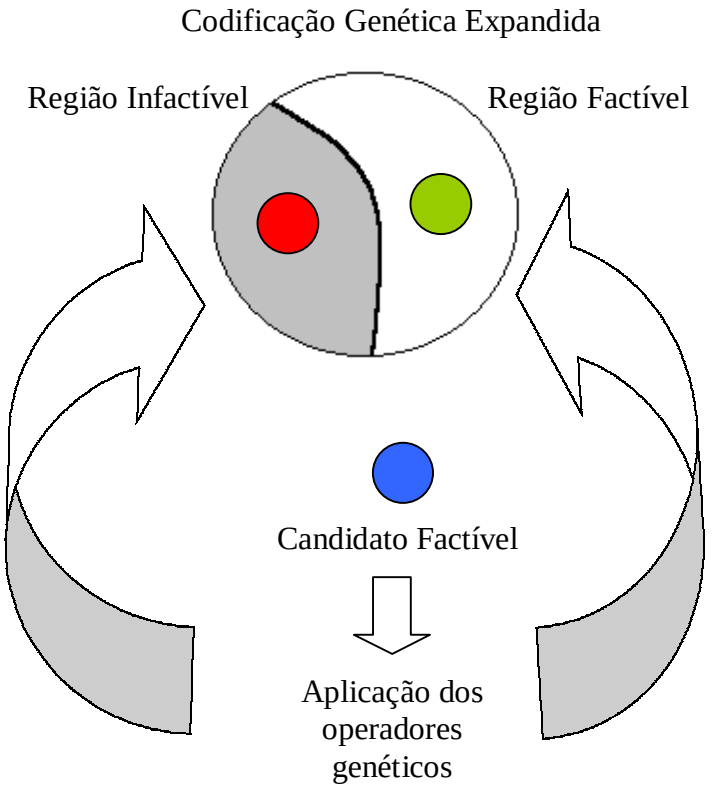


Figura 6.10: Geração de descendentes factíveis ou infactíveis após a aplicação dos operadores genéticos.

6.8.1 Operador Genético de Recombinação

Para todos os pares de pais formados, existirá uma probabilidade p_c de ocorrência de *crossover*. No desenvolvimento deste trabalho, verificou-se que esta fase é bastante crítica, uma vez que será necessária a criação de cromossomos-filho que sejam factíveis (portanto devem seguir os critérios de formação já apresentados) após a junção de características dos seus ascendentes.

Para tanto, será sorteado um período qualquer de um dos cromossomos-pai, sendo repassado integralmente ao cromossomo-filho. A partir da próxima posição final mais um do primeiro cromossomo, pegam-se os genes do segundo cromossomo-pai, repassando-os (quando possível) para o descendente que está sendo gerado. Para que essa operação seja possível, são necessárias algumas condições:

- o professor que é representado pelo gene que está sendo transmitido ao descendente, não poderá estar presente no cromossomo-filho com toda a sua carga horária esgotada;

- o gene passado não poderá estar presente no período que o estará recebendo.

A transgressão dessas condições fará com que o descendente gerado seja infactível. Nesse sentido, o algoritmo de recombinação tem um tratamento completo para esses casos, tentando viabilizar o cromossomo-filho (mantê-lo factível). Essa rotina é executada um certo número de vezes. Excedido esse limite, todo o processo é reiniciado.

As figuras a seguir ilustram a operação de recombinação conforme a carga horária apresentada na Tabela 6.9.

Tabela 6.9: Carga horária dos professores que comporão o código genético para a aplicação da operação de recombinação.

Professor	**Carga Horária**
A	1
B	2
C	1
D	2
E	1
F	2
G	1

Cromossomo-pai 1

A	B	F	D	E	período 1
F	B	C	D	G	período 2

Cromossomo-pai 2

G	B	D	C	F	período 1
D	A	F	E	B	período 2

Figura 6.11: Cromossomos que serão submetidos ao operador genético de recombinação.

Na aplicação do operador genético de recombinação, o primeiro período do cromossomo-pai 1 é passado integralmente para o descendente gerado.

A	B	F	D	E	período 1
					período 2

Figura 6.12: Formação do descendente gerado pelo *crossover*.

O passo seguinte é tentar repassar o gene *D* do cromossomo-pai 2 para o segundo período do descendente. Como a carga horária deste gene é 2 (conforme Tabela 6.9) e ainda não está presente no segundo período do descendente, essa operação será executada.

A	B	F	D	E	período 1
D					período 2

Figura 6.13: Passagem do gene *D* do cromossomo-pai 2
para o segundo período do descendente.

A próxima etapa é tentar repassar o gene *A* (seguinte ao *D*) para o descendente. Como a carga horária deste gene é 1 (conforme Tabela 6.9) e por já estar presente no primeiro período do descendente, a sua passagem não será permitida. Dessa maneira, a próxima tentativa é referente ao gene seguinte (gene *F*). Como a carga horária do gene F é 2 (conforme Tabela 6.9) e ainda não está presente no segundo período do descendente, a sua passagem será efetuada.

A	B	F	D	E	período 1
D	F				período 2

Figura 6.14: Passagem do gene *F* do cromossomo-pai 2
para o segundo período do descendente.

Seguindo os mesmos passos descritos acima, os genes do cromossomo-pai 2 serão repassados para o descendente até que ele esteja com toda a sua carga genética completa. Como resultado final, o cromossomo filho terá a seguinte configuração:

A	B	F	D	E	período 1
D	F	B	G	C	período 2

Figura 6.15: Configuração final do descendente após aplicação do operador genético de recombinação.

A configuração apresentada para o descendente gerado a partir dos cromossomos-pai 1 e 2, tem como característica principal ser um indivíduo factível e estar na forma genética compacta. Vale lembrar que o algoritmo está trabalhando em uma região de factibilidade do espaço (conforme ilustrado na Figura 6.9).

Feita a aplicação desse operador genético de recombinação (*Order Crossover*) será realizado o procedimento de expansão do código para a geração completa do cromossomo (esta geração está sujeita às regras já mencionadas). Realizada a expansão, o mesmo tipo de busca local será efetuada (conforme já mencionado), tentando-se obter um máximo local.

6.8.2 Mutação Gênica

Para todos os indivíduos da população, vai existir uma taxa de mutação p_m para que um cromossomo sofra uma Mutação Inversiva. Os mesmos cuidados devem ser tomados para os cromossomos mutantes que forem gerados, portanto o procedimento de reparação é novamente empregado. Entretanto, para que a mutação seja válida, outros dois processos de validação serão necessários.

Dado um cromossomo, um gene (professor) será sorteado. A seguir, é necessário encontrar o período no qual este professor se localiza no cromossomo selecionado. Como próximo passo, deve-se encontrar (se é que existe) um período que não contenha o professor sorteado e, portanto, possa recebê-lo. Um terceiro procedimento é verificar qual gene do segundo período poderá sofrer a inversão com o professor sorteado. Em outras palavras, o primeiro período não poderá receber um gene que já pertença a ele (isso infactibilizaria o cromossomo).

A Figura 6.16 apresenta um exemplo de cromossomo que não poderá receber a mutação em relação ao gene *G*. Note que esse gene já está presente nos dois períodos. Dessa maneira a sua inversão com qualquer outro elemento faria com que um mesmo

período tivesse dois genes iguais. Esse procedimento tornaria o descendente gerado infactível.

G	F	C	D	E	período 1
G	C	D	E	B	período 2

Figura 6.16: O gene que representa o professor *G* não poderá submeter-se ao operador genético de mutação, uma vez que já está presente nos dois períodos.

O próximo exemplo assinala dois genes que podem ser invertidos. Note que antes da mutação o gene *B* não está presente no primeiro período, somente ocorrendo este fato após a aplicação do operador. O mesmo comentário é válido para o gene *F* em relação ao segundo período. A mutação exemplificada é válida, uma vez que mesmo após essa operação o cromossomo permanecerá factível.

G	F	C	D	E	período 1
G	C	D	E	B	período 2

Figura 6.17: Exemplo de genes que podem ser submetidos à mutação.

A Figura 6.18 apresenta a nova configuração após a realização da inversão.

G	B	C	D	E	período 1
G	C	D	E	F	período 2

Figura 6.18: Nova configuração do cromossomo após a mutação.

Encerrado o processo de mutação, o cromossomo mutante passará pela busca local já descrita, para tentar alcançar um valor de *fitness* que seja um ótimo local.

6.9 Seleção Parcialmente Elitista

Este algoritmo evolutivo caracteriza-se por ser parcialmente elitista, de modo que a melhor quinta parte da população é reutilizada na próxima geração e o restante é complementado pela produção de novos elementos (certamente esses novos elementos deverão seguir as leis de formação apresentadas e passarão pelo mesmo processo de busca por um ótimo local).

6.10 Cálculo da Função de *Fitness*

Durante todo esse processo de evolução o *fitness* de cada cromossomo é calculado, mostrando o quanto ele é adaptado às condições de contorno do ambiente, portanto sendo um valor característico de cada elemento da população.

Dado um cromossomo factível qualquer, o seu *fitness* será o somatório do grau de satisfação de todos os genes (professores) que o compõem, de modo que quanto maior for este valor, melhor adaptado e com maiores condições de sobrevivência e de reprodução será este elemento. A equação que representa o cálculo do *fitness* é apresentada a seguir:

$$fitness = \sum_{i=1}^{n} GS_i$$

sendo:

n= número de genes do cromossomo;

GS_i= grau de satisfação de cada gene do cromossomo.

Novamente, traçando um paralelo com o processo evolutivo natural, sabe-se que os indivíduos mais adaptados (com maior valor de *fitness*) são melhores candidatos para a sobrevivência e liderança da população, o que nos leva para a segunda conclusão de Darwin.

As próximas tabelas apresentam o cálculo do *fitness* de candidatos factíveis à solução. O grau de satisfação é um valor fornecido pelo professor sendo função da sua preferência pelo dia da semana e pelo período de aula requerido.

A Figura 6.19 representa a evolução do *fitness* ao longo das gerações para a grade horária apresentada na Tabela 6.11.

Tabela 6.10: Cálculo do *fitness* de um candidato factível à solução com cinco salas por período.

							Somatória do Grau de Satisfação
2ª feira	Período 1	19	17	15	14	10	
	Grau de Satisfação	5	5	5	5	5	25
	Período 2	19	17	15	14	18	
	Grau de Satisfação	5	5	5	5	5	25
3ª feira	Período 1	13	3	12	6	2	
	Grau de Satisfação	5	5	5	5	5	25
	Período 2	13	3	12	20	18	
	Grau de Satisfação	5	5	5	5	5	25
4ª feira	Período 1	17	11	7	9	5	
	Grau de Satisfação	5	5	5	5	5	25
	Período 2	17	11	7	20	1	
	Grau de Satisfação	5	5	5	5	5	25
5ª feira	Período 1	20	19	8	5	4	
	Grau de Satisfação	5	5	5	5	5	25
	Período 2	20	19	8	18	16	
	Grau de Satisfação	5	5	5	5	5	25
6ª feira	Período 1	18	15	7	5	16	
	Grau de Satisfação	5	5	5	5	5	25
	Período 2	18	15	7	5	4	
	Grau de Satisfação	5	5	5	3	5	23

Somatória total $\Rightarrow$ 248

Analisando-se a tabela anterior, verifica-se que vários professores estão alocados duas vezes em seqüência para o mesmo dia da semana e com satisfação máxima. Esse resultado obtido está de acordo com o grau de preferência dos professores em relação a cada um dos períodos, conforme indicado na Tabela 6.3. Outra característica que garante a factibilidade do candidato à solução é a obediência à carga horária semanal de cada professor, como indica a Tabela 6.2.

Tabela 6.11: Cálculo do *fitness* de um candidato factível
à solução com onze salas por período.

													Somatória do Grau de Satisfação
2ª feira	Período 1	39	23	40	10	30	29	34	32	17	15	9	
	Grau de Satisfação	5	5	5	5	5	5	5	5	5	5	5	55
	Período 2	36	18	30	32	14	9	29	15	34	17	19	
	Grau de Satisfação	5	5	5	5	5	5	5	5	5	5	5	55
3ª feira	Período 1	20	18	27	5	11	34	40	4	3	2	21	
	Grau de Satisfação	5	5	5	5	5	5	5	5	5	5	5	55
	Período 2	4	2	12	22	21	11	13	34	15	18	33	
	Grau de Satisfação	5	5	5	5	5	5	5	5	5	5	1	51
4ª feira	Período 1	9	5	31	34	26	38	7	40	11	20	17	
	Grau de Satisfação	5	5	5	5	5	5	5	5	5	5	5	55
	Período 2	26	20	40	29	7	1	31	11	17	22	34	
	Grau de Satisfação	5	5	5	5	5	5	5	5	5	5	5	55
5ª feira	Período 1	38	9	39	35	5	19	8	18	1	26	37	
	Grau de Satisfação	5	5	5	5	5	5	5	5	5	5	0	50
	Período 2	13	20	8	4	39	18	1	19	26	29	11	
	Grau de Satisfação	4	5	5	5	5	5	5	5	5	5	5	54
6ª feira	Período 1	7	28	5	18	40	12	15	16	30	36	35	
	Grau de Satisfação	5	5	5	5	5	5	5	5	5	5	5	55
	Período 2	30	40	35	28	12	7	16	3	29	36	26	
	Grau de Satisfação	5	5	5	5	5	5	5	5	5	5	5	50
Sábado	Período 1	39	7	5	33	37	36	25	9	40	4	24	
	Grau de Satisfação	5	3	5	5	5	5	5	1	0	5	5	44
	Período 2	21	24	39	23	7	14	12	19	35	38	6	
	Grau de Satisfação	5	5	5	5	4	0	5	0	0	0	0	29

Somatória total $\Rightarrow$ 608

Os comentários realizados para a Tabela 6.10, também são válidos para o caso da Tabela 6.11. Acrescenta-se que, notadamente no segundo período do Sábado, há professores alocados que não tiveram as suas preferências atendidas (grau de satisfação zero). Essa peculiaridade não infactibiliza esse candidato à solução. Em outras palavras, a função-objetivo que pretende maximizar o grau de preferência dos professores não pôde ser plenamente atendida. Justifica-se esse fato, como sendo um efeito sintomático do processo de expansão do código.

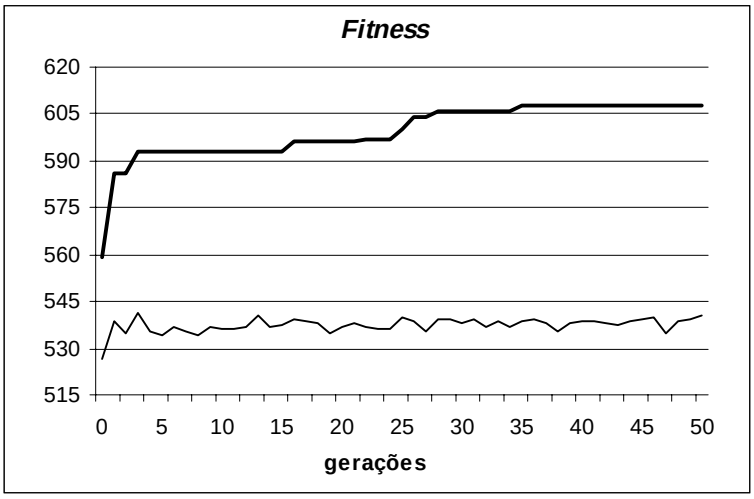


Figura 6.19: Comparação entre o *fitness* médio (linha fina)
e o melhor *fitness* (linha grossa) ao longo das gerações.

A Figura 6.19 mostra a presença de uma distância entre o *fitness* médio e o produzido pelo melhor indivíduo ao longo das gerações, demonstrando, portanto, a existência de um nível significativo de diversidade na população.

A Tabela 6.12 ilustra o tempo de execução para a codificação genética expandida e compacta para um total de 100 gerações e tamanho de população igual a 100. O programa computacional foi desenvolvido no ambiente de programação *Delphi* e os tempos obtidos foram para o processamento em um microcomputador *Pentium* III 650MHz com 128Mbytes de memória *RAM*.

Tabela 6.12: Tempo de execução do programa computacional desenvolvido.

Codificação Genética	**Salas por Período**	***Fitness* alcançado**	**Tempo de Execução**
Expandida	5	236	40s
Compacta	5	239	1min 05s
Expandida	11	601	10min 35s
Compacta	11	600	16min 50s

A diferença entre o *fitness* obtido para a representação genética compacta e expandida não é significativa para as duas configurações testadas. Em relação ao tempo de execução do programa computacional desenvolvido, ela é claramente maior para o caso da codificação compacta. Tal fato é justificado pela presença do algoritmo de expansão de código.

A Tabela 6.13 apresenta testes realizados com a codificação genética compacta, utilizando uma variação no tamanho da população e o número de gerações fixado em 100. Os valores de *fitness* apresentados representam uma média dos dez testes realizados para cada um dos casos.

Tabela 6.13: Resultados obtidos com a variação do tamanho da população.

Salas por Período	**Número de indivíduos da população**	**Média do *fitness* Alcançado**
5	10	230,8
	30	234,5
	70	231,5
	100	233,2
	1000	238,6
11	10	584,9
	30	589,8
	70	593,6
	100	595,9
	1000	599,8

Como esperado, trabalhando-se com um número maior de indivíduos na população a cada geração, ao final das 100 gerações há uma tendência de se obter soluções de melhor qualidade, embora o custo computacional seja muito superior, podendo não ser compensado pelo incremento de qualidade da solução.

CAPÍTULO 7

CONCLUSÃO

A aplicação conjunta de técnicas de computação evolutiva, busca local e otimização baseada em restrições tem por objetivo produzir uma solução de compromisso para problemas de otimização caracterizados por apresentarem um grande número de variáveis, múltiplos objetivos e múltiplas restrições. Os dois casos tratados neste trabalho correspondem a problemas reais de escalonamento: geração de turnos completos em torneios e definição de grade horária de professores em instituições de ensino.

Nesta dissertação, a estrutura de apresentação de conceitos e resultados foi definida com o propósito de focalizar inicialmente cada uma das três técnicas envolvidas (capítulos 2, 3 e 4), seguida da apresentação da proposta de aplicação conjunta (ao final do capítulo 4) e dos resultados de aplicação (capítulos 5 e 6). Os dois anexos abordam aspectos importantes deste trabalho: a necessidade do uso de um gerador de números pseudo-aleatórios com a propriedade de repetitividade e, especificamente para o caso de alocação de carga didática, a formalização de uma função-objetivo real com base em dados obtidos a partir de uma realimentação dos professores por intermédio de um questionário a ser usado para mapear as preferências e disponibilidades do corpo docente em cada período letivo.

Como principais contribuições deste trabalho, é possível mencionar:

- a própria iniciativa de aplicação conjunta de técnicas de computação evolutiva, busca local e otimização baseada em restrições, com a distribuição adequada de “papéis” entre cada técnica, procurando explorar o melhor que cada uma poderia oferecer ao longo da busca pela solução ótima;
- a formalização da proposta em termos de conceitos bem definidos no campo da computação evolutiva, como decodificadores, algoritmos de reparação, algoritmos meméticos e efeito Baldwin;
- elaboração e implementação computacional de um algoritmo de expansão de código para o problema da geração de turnos completos em torneios;

- elaboração e implementação computacional de um algoritmo de expansão de código para o problema da definição de grade horária de professores em instituições de ensino. Este algoritmo apresenta uma inovação importante: como a expansão de código pode ser caracterizada como uma caminhada em uma árvore de decisão, a existência de objetivos a serem otimizados foi levada em conta de modo a polarizar o processo de tomada de decisão, sempre no sentido de caminhar por ramos da árvore que indicassem um aumento na qualidade da solução.
- comparação de desempenho entre codificação expandida e codificação compacta.

Os resultados obtidos ao aplicar a técnica proposta aos dois problemas de escalonamento mencionados acima permitem concluir que é possível se beneficiar com o uso de uma representação compacta em conjunto com um algoritmo de expansão de código, quando comparado com o emprego direto de codificação expandida. As razões levantadas para explicar o sucesso desta abordagem são:

- o processo evolutivo somente tem lugar no espaço compacto, onde não há infactibilidade ou onde a factibilidade pode ser facilmente conquistada. Com isso, a existência de restrições praticamente não afeta a taxa de evolução no espaço compacto.
- o algoritmo de expansão de código, responsável pelo atendimento das restrições, é um método de otimização baseado em restrições, de modo que a representação expandida resultante será sempre factível;
- as duas possíveis limitações do algoritmo de expansão de código, ou seja, a impossibilidade de garantir que qualquer representação expandida factível tenha uma representação compacta associada e a existência de ótimos locais como resultado do processo de expansão, são amenizadas respectivamente pela presença de busca local e pela evolução de uma população de representações compactas.

A abordagem proposta, como qualquer outra estratégia de busca implementada em computador, também apresenta suas desvantagens:

- necessidade de definição de uma codificação compacta para cada problema de aplicação, mais especificamente de um espaço de busca compacto, sendo que sua dimensão mostrou-se decisiva na eficiência do processo evolutivo;

- necessidade de implementação de um algoritmo de expansão de código para cada problema de aplicação, levando em conta todas as particularidades do problema;
- custo computacional vinculado ao processo de busca. O algoritmo pode ter reduzida sua capacidade de busca no caso de problemas de tamanho muito elevado.

Para o caso de geração automática de tabelas em torneios, além de superar a abordagem baseada no código expandido, a comparação de desempenho junto a soluções propostas por especialistas, adotadas para problemas do mundo real, permite concluir que a adoção de uma codificação compacta, seguida da expansão de código com reparação e busca local, é muito eficaz na procura de soluções de alta qualidade, superando de forma evidente soluções que vem sendo adotadas na prática.

Como perspectivas futuras para este trabalho são sugeridos os seguintes tópicos:

- verificar a elaboração de novos operadores genéticos que possam adaptar-se melhor a cada contexto de aplicação;
- verificar a possibilidade de incorporação de punições na função de *fitness* para o caso do processo de expansão, juntamente com a busca local e os algoritmos de reparação, requerer recursos computacionais excessivos (por exemplo, número elevado de ocorrências de *backtracking*);
- permitir que um mesmo processo de expansão de código possa operar com diferentes geradores de números aleatórios, todos apresentando a propriedade de repetitividade;
- no caso da aplicação da estratégia de solução proposta neste trabalho a problemas que admitem o emprego de mais de um algoritmo de otimização baseada em restrições, criar condições para que o algoritmo de expansão de código possa optar por um ou outro dentre os algoritmos disponíveis para expansão de código;
- verificar a possibilidade de aplicação do conceito de sistemas auto-adaptativos, conforme discutido no capítulo 2;
- verificar a possibilidade de incorporação das técnicas utilizadas nos sistemas co-evolutivos (capítulo 2), procurando criar funções de *fitness* distintas para as restrições globais e locais;
- extensão da estratégia de solução proposta a outros problemas de escalonamento e possivelmente a outros problemas de otimização com restrições.

Os algoritmos evolutivos em sua composição tradicional encontram dificuldades no tratamento de problemas de otimização com restrições. Também é fato, que técnicas de otimização baseada em restrições quando aplicadas isoladamente podem encontrar sérias dificuldades em virtude da existência de ótimos locais de baixa qualidade. Dessa maneira, este trabalho, por intermédio da aplicação conjunta de ambas as metodologias, mostra que resultados promissores podem ser obtidos, desde que haja um equilíbrio no papel a ser desempenhado por essas técnicas.

ANEXO A

GERADOR DE NÚMEROS ALEATÓRIOS COM DISTRIBUIÇÃO UNIFORME E REPETITIVIDADE

Conforme definido por L'ECUYER (1994), computadores digitais só podem gerar números pseudo-aleatórios, por se tratarem de máquinas totalmente determinísticas. No entanto, desde que um gerador de números pseudo-aleatórios seja aprovado em testes de aleatoriedade, sua aplicação vai levar a comportamentos equivalentes aos produzidos por geradores puramente aleatórios. Sendo assim, não se faz distinção neste trabalho entre aleatoriedade e pseudo-aleatoriedade.

Um gerador de números (pseudo-)aleatórios em computadores digitais tem um estado que evolui em um espaço *S*. Esse espaço é composto por um número finito de estados e a repetitividade é garantida a partir de uma recorrência na forma: $s_n = f(s_{n-1}),\ n \geq 1$, sendo que $s_o \in S$ será denominado a semente e $f : S \rightarrow S$ será a função determinística de transição. No enésimo passo, a função de saída do gerador será dada por $u_n = g(s_n)$ com $g : S \rightarrow [0,1]$ (essa saída poderia ser mais geral, entretanto está sendo assumido o intervalo *[0,1]*). Observe que a seqüência de saída do gerador será um conjunto de valores representado por $\{u_n, n \geq 0\}$. Como o espaço *S* é finito, a seqüência $\{u_n, n \geq 0\}$ deverá ser periódica (possivelmente após um transitório inicial). Em outras palavras, todas as vezes em que a semente s_o for a mesma, a seqüência aleatória gerada será repetida. Em situações em que é necessário aumentar a periodicidade do gerador, ou seja quando a quantidade de números aleatórios a serem gerados é muito grande, será desejável fazer com que esse período seja o mais próximo possível da cardinalidade do espaço *S*.

Geralmente, os modelos fornecidos pelos sistemas computacionais são os geradores lineares apresentando uma relação de recorrência $I_{j+1} = (aI_j + c) \bmod m,\ j = 1,2,\ldots,$ responsável pela geração de uma seqüência $I_1, I_2, I_3, \ldots$ de inteiros entre 0 e $m-1$, sendo m o módulo, a e c inteiros positivos denominados multiplicador e incremento. A recorrência vai

certamente produzir, para algum $j = p \leq m, I_j = I_k \ (k < j)$, ou seja, ela terá um período $p \leq m$. Se o período for $p = m$, todo inteiro entre 0 e m–1 vai ocorrer em alguma das próximas m–1 iterações, fazendo com que a escolha do valor inicial I_0 da recorrência (semente da geração pseudo-aleatória) não influa de forma significativa no resultado estatístico associado a seqüências longas. Os geradores lineares têm a vantagem de serem rápidos, de simples implementação e repetitivos para uma mesma máquina.

ANEXO B

QUESTIONÁRIO PARA OBTENÇÃO DAS PREFERÊNCIAS DOS PROFESSORES PARA FINS DE CRITÉRIO DE DESEMPENHO

Esta pesquisa foi responsável pelo levantamento da disponibilidade dos docentes envolvidos no processo e das suas respectivas preferências. Os dados apresentados são valores reais de uma instituição de ensino superior da cidade de São Paulo.

Curso: Ciências Ano: 2000

Nome: _____

Disciplina	Ano	Número de aulas

	2ª feira	3ª feira	4ª feira	5ª feira	6ª feira	Sábado
Período 1	Grau de preferência ( )	Grau de preferência ( )	Grau de preferência ( )	Grau de preferência ( )	Grau de preferência ( )	Grau de preferência ( )
Período 2	Grau de preferência ( )	Grau de preferência ( )	Grau de preferência ( )	Grau de preferência ( )	Grau de preferência ( )	Grau de preferência ( )

O grau de preferência mostra a sua "satisfação" em relação ao dia e período indicado. Deverá ser preenchido com valores no intervalo de 1 a 5, sendo que grau 5 significa satisfação máxima e grau 1 representa satisfação mínima. O grau de preferência somente deverá ser marcado nos períodos que forem assinalados como disponíveis para a atribuição de aulas (os períodos não assinalados terão grau de preferência zero).

ANEXO C

CÁLCULO PARA A DEFINIÇÃO DO NÚMERO DE CANDIDATOS FACTÍVEIS NO CASO DE TURNOS COMPLETOS EM TORNEIOS

A dedução apresentada a seguir, representa o número de candidatos presentes no espaço de busca da representação expandida, embora não esteja sendo considerado, nos cálculos que seguem, nenhuma restrição referente a mando de jogo. Existe distinção entre jogos fora ou no domínio de cada participante, mas serão também considerados elementos do espaço de busca (turnos) que não apresentam equilíbrio entre jogos fora e no domínio de cada participante.

Considere um torneio com seis participantes: A, B, C, D, E e F. Todos os possíveis jogos existentes entre esses participantes são os apresentados a seguir, sendo que há um agrupamento em pares, já que vai ser possível optar por qualquer um dos elementos do par, sem alterar o jogo e, ainda, somente um dos elementos do par estará presente em um dado turno. Por exemplo, dado que se decidiu pela inclusão do jogo entre os participantes A e B em uma determinada rodada do torneio, a escolha do mando de jogo representa um evento independente que será considerado após a ordenação de todos os jogos do turno.

AB	AC	AD	AE	AF
BA	CA	DA	EA	FA
BC	BD	BE	BF	
CB	DB	EB	FB	
CD	CE	CF		
DC	EC	FC		
DE	DF			
ED	FD			
EF				
FE				

Para o primeiro jogo da primeira rodada a ser sorteado e para quaisquer dois participantes, há um número de cinco pares que poderão ser escolhidos. Uma vez que um dado elemento do par seja sorteado, nenhum dos elementos deste par poderá participar novamente do mesmo turno. Isso faz com que os pares que poderão fazer parte do turno tenham o seu número reduzido. Para ficar mais claro, suponha que o primeiro par sorteado tenha sido AB, implicando que os jogos:

AB	AC	AD	AE	AF
BA	CA	DA	EA	FA
BC	BD	BE	BF	
CB	DB	EB	FB	

não poderão estar presentes na mesma rodada, uma vez que os participantes A e B já apareceram. Também não será possível com que os elementos AB e BA apareçam novamente no mesmo turno.

Para o segundo jogo da primeira rodada, estarão disponíveis os seguintes pares:

CD	CE	CF
DC	EC	FC
DE	DF	
ED	FD	
EF		
FE		

Portanto, verifica-se que há três pares para cada um dos participantes que ainda são passíveis de sorteio. Supondo, agora, que CD tenha sido sorteado, implicando que os jogos:

CD	CE	CF
DC	EC	FC
DE	DF	
ED	FD	

não poderão mais estar presentes na mesma rodada. Restam, para o terceiro jogo da primeira rodada, os pares EF e FE, indicando um único par para fechar a rodada.

Da segunda rodada em diante, deve-se levar em consideração os pares que já fazem parte do turno e, portanto, não poderão estar presentes novamente no torneio. Pelo exemplo considerado, os pares AB, BA, CD, DC, EF e FE não poderão fazer parte do novo sorteio. Portanto, para o primeiro jogo da segunda rodada, haverá as seguintes possibilidades:

----	AC	AD	AE	AF
----	CA	DA	EA	FA
BC	BD	BE	BF	
CB	DB	EB	FB	
----	CE	CF		
----	EC	FC		
DE	DF			
ED	FD			

----				

de modo que o número de pares que poderão ser sorteados para quaisquer participantes foi reduzido para quatro.

Aplicando, agora, a mesma metodologia empregada para a primeira rodada, o segundo jogo da segunda rodada terá dois pares possíveis para o sorteio. Da mesma maneira, o terceiro jogo terá um único par possível.

Após a conclusão da escolha dos jogos que comporão o turno, fica faltando escolher o domínio (quais participantes jogarão em seu domínio e quais jogarão fora dele). Para cada rodada existem três jogos, de modo que o número de possibilidades é 2^3.

O esquema a seguir representa o número de candidatos factíveis para o exemplo dado (seis participantes):

Primeira rodada	5	3	1	$\times 2^3$
Segunda rodada	4	2	1	$\times 2^3$
Terceira rodada	3	1	1	$\times 2^3$
Quarta rodada	2	1	1	$\times 2^3$
Quinta rodada	1	1	1	$\times 2^3$

Portanto, o número de candidatos factíveis será dado pela expressão:

$$(5\times3\times1\times8)\times(4\times2\times1\times8)\times(3\times1\times1\times8)\times(2\times1\times1\times8)\times(1\times1\times1\times8) =$$
$$= (5\times4\times3\times2\times1)\times(3\times2\times1\times1\times1)\times(1\times1\times1\times1\times1)\times(2^3\times2^3\times2^3\times2^3\times2^3) =$$
$$= 5!\times3!\times1!\times2^{5\times3} \cong 2{,}36\times10^7.$$

Repetindo a mesma seqüência de passos, agora para oito participantes, o esquema será o seguinte:

Primeira rodada	7	5	3	1	$\times2^4$
Segunda rodada	6	4	2	1	$\times2^4$
Terceira rodada	5	3	1	1	$\times2^4$
Quarta rodada	4	2	1	1	$\times2^4$
Quinta rodada	3	1	1	1	$\times2^4$
Sexta rodada	2	1	1	1	$\times2^4$
Sétima rodada	1	1	1	1	$\times2^4$

Portanto, o número de candidatos factíveis será dado pela expressão:

$$(7\times5\times3\times1\times16)\times(6\times4\times2\times1\times16)\times(5\times3\times1\times1\times16)\times(4\times2\times1\times1\times16)\times(3\times1\times1\times1\times16)\times$$
$$\times(2\times1\times1\times1\times16)\times(1\times1\times1\times1\times16) =$$
$$= (7\times6\times5\times4\times3\times2\times1)\times(5\times4\times3\times2\times1\times1\times1)\times(3\times2\times1\times1\times1\times1\times1)\times(1\times1\times1\times1\times1\times1\times1)\times$$
$$\times(2^4\times2^4\times2^4\times2^4\times2^4\times2^4\times2^4) =$$
$$= 7!\times5!\times3!\times1!\times2^{7\times4} \cong 9{,}7410\times10^{14}.$$

Por intermédio desses cálculos é possível deduzir a seguinte fórmula para *n* participantes:

$$(n-1)!\times(n-3)!\times(n-5)!\times\ldots\times(n-(n-1))!\times2^{(n-1)\times\frac{n}{2}},$$

representando o número de turnos completos possíveis para o caso de *n* participantes, lembrando que está sendo assumido que *n* é par.

REFERÊNCIAS BIBLIOGRÁFICAS

AARTS, E. & KORST, J. Simulated Annealing and Boltzmann Machines – A Stochastic Approach to Combinatorial Optimization and Neural Computing, John Wiley & Sons, 1989.

AARTS, E. & VERHOEVEN, M. Operations Research, in Bäck, T., Fogel, D. B. & Michalewicz, Z. Handbook of Evolutionary Computation, Oxford University Press, 1997.

AARTS, E. & LENSTRA, J. K. Local Search in Combinatorial Optimization, Wiley, 1997.

AHUJA, R. K., MAGNANTI, T. L. & ORLIN, J. B. Network Flows: Theory, Algorithms and Applications, Prentice Hall, Englenwood Cliffs, NJ, 1993.

AMABIS, J. M. & MARTHO, G. R. Biologia – Populações – Genética, Evolução e Ecologia, Editora Moderna Ltda., vol. 3, 1997.

ATMAR, W. Notes on the Simulation of Evolution, IEEE Transactions on Neural Networks, 5(1): pp.130-147, 1994.

BÄCK, T., HAMMEL, U. & SCHWEFEL, H. -P. Evolutionary Computation: Comments on the History and Current State, IEEE Transactions on Evolutionary Computation, 1(1): pp. 3-17, 1997(a).

BÄCK, T., FOGEL, D. B. & MICHALEWICZ, Z. (eds.) Handbook of Evolutionary Computation, Oxford University Press, 1997(b).

BEASLEY D. Why Evolutionary Computation?, in Bäck, T., Fogel, D. B. & Michalewicz, Z. Handbook of Evolutionary Computation, Oxford University Press, 1997.

CANTÚ-PAZ, E. A Survey of Parallel Genetic Algorithms, Calculateurs Parallèles, Reseaux et Systems Repartis, 10(2): pp. 141-171, 1998.

CONCILIO, R. & VON ZUBEN, F. J. Geração Automática de Tabela de Jogos em Torneios: Uma Abordagem Evolutiva com Busca Local e Representação Genética Compacta, Anais do XIII Congresso Brasileiro de Automática (CBA 2000), pp. 1247-1252, Florianópolis 2000(a).

CONCILIO, R. & VON ZUBEN, F. J. Evolutionary Design of Schedules in Championships with Compact Genetic Codification and Local Search, Workshop on Memetic Algorithms (WOMA) at the 2000 Genetic and Evolutionary Computation Conference (GECCO-2000), pp. 109-113, Las Vegas, Nevada, USA, July, 2000(b).

COSTA, M. F. N. Computação Evolutiva para a Minimização de Perdas Resistivas em Sistemas de Distribuição de Energia Elétrica, Dissertação de Mestrado, Faculdade de Engenharia Elétrica e de Computação, Unicamp, 1999.

DAWKINS, R. The Selfish Gene, Oxford: Oxford University Press, 1976.

DE CASTRO, L. N., IYODA, E. M., VON ZUBEN, F. J. & GUDWIN, R. Feedforward Neural Network Initialization: an Evolutionary Approach, Proceedings of the Vth Brazilian Symposium on Neural Networks, vol. 1, pp. 43-48, 1998.

EIBEN, A. E. & RUTTKAY, Z. Constraint-Handling Techniques, in Bäck, T., Fogel, D. B. & Michalewicz, Z. Handbook of Evolutionary Computation, Oxford University Press, 1997.

EIBEN, A. E., HINTERDING, R. & MICHALEWICZ, Z. Parameter Control in Evolutionary Algorithms, IEEE Transactions on Evolutionary Computation, 3(2): pp. 124-141, 1999.

ESHELMAN, L. J., CARUANA, R. A. & SCHAFFER, J. D. Biases in the Crossover Landscape, em Schaffer, J. (ed.), Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, pp. 10-19, 1989.

FOGEL, L. J., OWENS, A. J. & WALSH, M. J. Artificial Intelligence Through Simulated Evolution, John Wiley, 1966.

FOGEL, D. B. An Introduction to Simulated Evolutionary Computation, IEEE Transactions on Neural Networks, 5(1): pp. 3-14, 1994.

FOGEL, D. B. Why Evolutionary Computation?, in Bäck, T., Fogel, D. B. & Michalewicz, Z. Handbook of Evolutionary Computation, Oxford University Press, 1997.

FOGEL, D. B. Evolutionary Computation – Toward a New Philosophy of Machine Intelligence, 2ª edição, IEEE Press, 1999.

GLOVER, F. & LAGUNA, M. Tabu Search, Kluwer Academic Publishers, 1997.

GOLDBERG, D. E. Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, 1989.

GOLDBERG, D. E. A Meditation on the Application of Genetic Algorithms, IlliGAL Report No. 98003, Department of General Engineering, University of Illinois at Urbana-Champaign, EUA, 1998.

GRUAU, F. Genetic Micro Programming of Neural Networks, in Kinnear Jr., K. (ed.), Advances in Genetic Programming, The MIT Press, pp. 495-518, 1994.

HARTL, D. L. & CLARK, A. G. Principles of Population Genetics, Sinauer, 1989.

HOLLAND, J. H. Adaptation in Natural and Artificial Systems, 2ª edição, MIT Press, 1992.

IYODA, E. M. Inteligência Computacional no Projeto Automático de Redes Neurais Híbridas e Redes Neurofuzzy Heterogêneas, Dissertação de Mestrado, Faculdade de Engenharia Elétrica e de Computação, Unicamp, 2000.

KOZA, J. R. Genetic Programming: on the Programming of Computers by Means of Natural Selection, MIT Press, 1992.

KOZA, J. R., BENNET III, F. H., ANDRE, D., KEANE, M. A. & DUNLAP, F. Automated Synthesis of Analog Electrical Circuits by Means of Genetic Programming, IEEE Transactions on Evolutionary Computation, 1(2): pp. 109-128, 1997.

L'ECUYER, P. Testing random number generation, Annals of Operations Research, vol. 53, pp. 77-120, 1994.

MERZ, P. & FREISLEBEN, B. A Comparison of Memetic Algoritms, Tabu Search, and Ant Colonies for the Quadratic Assignment Problem, Congress on Evolutionary Computation, vol. 3, pp. 2063-2070, 1999.

MICHALEWICZ, Z. Genetic Algorithms + Data Structures = Evolution Programs, 3ª edição, Springer, 1996.

MICHALEWICZ, Z. & SCHOENAUER, M. Evolutionary Algorithms for Constrained Parameter Optimization Problems, Evolutionary Computation, 4(1): pp. 1-32, 1996.

MICHALEWICZ, Z. Constraint-Handling Techniques, in Bäck, T., Fogel, D. B. & Michalewicz, Z. Handbook of Evolutionary Computation, Oxford University Press, 1997.

MICHALEWICZ, Z. Evolutionary Algorithms for Constrained Optimization Problems, 2000 Genetic and Evolutionary Computation Conference Tutorial Program (GECCO-2000), pp.164-223, Las Vegas, Nevada, USA, July, 2000.

MITCHELL, M. & FORREST, S. Genetic Algorithms and Artificial Life, vol. 1, pp. 267-289, 1995.

MOSCATO, P. On Evolution, Search, Optimizaton, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms, Tech. Rep. Caltech Concurrent Computation Program, Report. 826, California Institute of Technology, Pasadena, California, USA, 1989.

MOSCATO, P. & NORMAN, M. G. A memetic approach for the travelling salesman problem – implementation of a computational ecology for optimisation on message-passing systems, Proceedings of the International Conference on Parallel Computing and Transputer Applications, Amsterdam, IOS Press, 1992.

MOSCATO, P. Memetic Algorithms: A Short Introduction, in Corne, D., Dorigo, M. & Glover, F. (eds.) New Ideas in Optimization, McGraw-Hill, pp. 219-234, 1999.

NEWALL, J. P. Hybrid Methods for Automated Timetabling, PhD Thesis, University of Notthingham, Department of Computer Science, UK, 2000.

PALMER, C. C. An approach to a problem in network design using Genetic Algorithn, PHD Thesis, Politechnic University, New York, NY, 1994.

RADCLIFFE, N. J. & SURRY, D. P. "Formal Memetic Algorithms", Evolutionary Computing: AISB Workshop, Ed: T. Forgaty, Springer-Verlag, 1994.

RECHENBERG, I. Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution, Frommann-Holzboog, 1973.

SCHWEFEL, H. –P. Evolution and Optimum Seeking, John Wiley, 1995.

SHERALI, H. D., BAZARAA, M. S. & SHETTY, C. M. Nonlinear Programming: Theory and Algorithms, 2nd edition, John Wiley & Sons, 1993.

SILVA JR., C. & SASSON, S. Biologia, Editora Saraiva, vol. 3, 1996.

SMITH, A. E. & COIT, D. W. Constraint-Handling Techniques, in Bäck, T., Fogel, D. B. & Michalewicz, Z. Handbook of Evolutionary Computation, Oxford University Press, 1997.

SURRY, P. D., RADCLIFFE, N. J. & BOYD, I. D. A multi-objective approach to constrained optimization of gas supply networks, AISB-95 Workshop on Evolutionary Computing, ed. T. Fogarty (Berlin: Springer), pp. 166-180, 1995.

SYSWERDA, G. Uniform Crossover in Genetic Algorithms, em Schaffer, J. D. (ed.), Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, pp. 2-9, 1989.

TURNEY, P., WHITLEY, D. & ANDERSDON, R. W. Evolution, Learning, and Instinct: 100 Years of the Baldwin Effect, Evolutionary Computation, vol. 4, no. 3, pp. iv-viii, 1996.

WEARE, R. F. Automated Examination Timetabling, PhD Thesis, University of Notthingham, Department of Computer Science, UK, 1995.

WHITLEY, D., GORDON, V. S. & MATHIASK K. Lamarckian Evolution, the Baldwin Effect and Function Optimization, Proceedings of the Third Conference on Parallel Problem Solving from Nature, ed. Yu Davidor, H-P Schwefel and R. Männer (Berlin: Springer), pp. 6-15, October, 1994.

ÍNDICE DE AUTORES