4

Sistema de Evolução de Designs Baseada em Qualidade de Software Orientado a Objetos

4.1
Arquitetura do Sistema Proposto

O Sistema de Evolução de Designs baseada em Qualidade de Software Orientado a Objetos procura sintetizar modelagens de software seguindo métricas de qualidade especificadas no trabalho de Bansiya (2002), já descritas no capítulo 2 desta dissertação.

A arquitetura do processo de síntese proposto pode ser vista na Figura 13 abaixo.

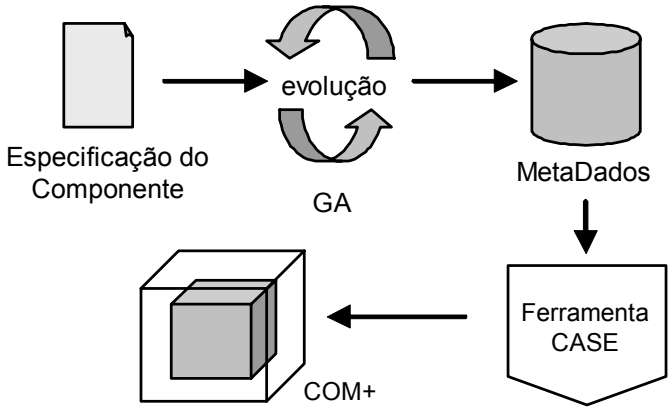

Figura 13 – Descrição do processo de síntese da modelagem

Pode-se entender melhor a figura acima através da descrição de suas etapas:

- Especificação do Componente

Nesta etapa é necessário um conhecimento do problema no qual a modelagem a ser sintetizada irá tratar. Por exemplo, se é desejado sintetizar uma modelagem de um sistema que represente um modelo de Rede Neural (Haykin, 1999) será necessário a consulta a um especialista no assunto.

O especialista deve fazer um levantamento de todas as características relevantes do modelo, assim como as relações entre elas, representando assim uma Especificação Inicial do Componente a ser sintetizado. Essa especificação contém uma coleção de métodos e atributos iniciais que caracterizam o comportamento do

componente, além de possíveis restrições e relações que indicam ligações entre métodos e atributos.

A Especificação do Componente pode ser dividida em três tabelas, como visto nas Tabelas 2, 3 e 4.

Código do Atributo	Nome do Atributo
Número inteiro	Rótulo para o Atributo para que este seja reconhecido na solução final mais facilmente

Tabela 2 – Especificação de Atributos

Código do Método	Nome do Método
Número inteiro	Rótulo para o Método para que este seja reconhecido na solução final mais facilmente

Tabela 3 – Especificação de Métodos

Código da Relação	Código do Método	Código do Atributo	Tipo
Número inteiro	Número inteiro	Número inteiro	1/0

Tabela 4 – Especificação de Relações

As Tabelas 2 e 3 simplesmente representam as listas de Atributos e Métodos, respectivamente. Cada Atributo e Método contém um código, que é utilizado na Tabela 4 para definir as relações entre os mesmos.

A Tabela 4, como foi dito, representa uma lista de relações entre Atributos/Métodos e Métodos/Métodos. Os Atributos e Métodos são referenciados pelos seus respectivos códigos e o tipo define se o campo "Código do Atributo" representa um Atributo ou um Método. Se o tipo for "0" a relação é entre Atributo/Método, mas se o tipo for "1", a relação é entre Método/Método.

As Tabelas 5, 6 e 7 abaixo mostram um exemplo simples de Especificação.

Código do Atributo	Nome do Atributo
1	Atributo_1
2	Atributo_2

Tabela 5 – Exemplo de Especificação de Atributos

Código do Método	Nome do Método
1	Método_1
2	Método_2
3	Método_3

Tabela 6 – Exemplo de Especificação de Métodos

Código da Relação	Código do Método	Código do Atributo	Tipo
1	1	2	0
2	2	1	0
3	2	2	0
4	3	2	1

Tabela 7 – Exemplo de Especificação de Relações

Através da leitura da Tabela 7, pode-se concluir as seguintes relações:

- Método_1 usa o Atributo_2
- Método_2 usa o Atributo_1 e o Atributo_2
- Método_3 usa o Método_2

- Evolução

Nesta fase é onde ocorre a síntese propriamente dita. Através do uso de um Algoritmo Genético (AG), a modelagem é evoluída.

Para realizar o processo evolucionário, o AG utiliza as informações da Especificação do Componente e gera como resultado final uma configuração de *metadados* da modelagem sintetizada. A estratégia então é gerar configurações aleatórias de *metadados* contendo as Classes criadas, com os Métodos e Atributos distribuídos pelas mesmas. Em seguida, avalia-se a modelagem correspondente através de Métricas de Qualidade.

O AG consiste na parte principal do modelo e é descrito em detalhes na próxima seção.

- Metadados

A utilização de *metadados* é comum entre *softwares* que automatizam a modelagem e geração de código. Deste modo, decidiu-se focar a evolução na *modelagem* do componente e não diretamente no seu código, sendo a modelagem

expressa por *metadados*. Com isso, neste trabalho, uma modelagem será representada por estruturas de *metadados* que descrevem as classes e relações que a compõem.

Metadados são definidos como *dados a respeito de dados*, ou seja, neste caso representam informações estruturadas (dados) sobre a modelagem (dados). Uma visão mais clara de como os *metadados* são utilizados no modelo pode ser obtida nos exemplos de estudo de casos, no capítulo 5.

- Ferramenta CASE

Como já foi descrito no capítulo 2, as ferramentas CASE (Computer-Aided Software Engineering) procuram automatizar as fases do processo de projeto de software orientado a objetos. A partir disso, uma ferramenta CASE é capaz de receber os *metadados* gerados pelo AG e fornecer diversos tratamentos em cima da modelagem sintetizada, utilizando a linguagem UML, podendo-se inclusive gerar o código referente a modelagem UML.

Nesta fase, um especialista pode, provavelmente, realizar modificações na modelagem de modo que esta fique mais de acordo com seus objetivos.

- Componente

A última fase do processo representa simplesmente o componente codificado gerado pela ferramenta CASE. No exemplo da Figura 11, o componente foi gerado e padronizado pelo padrão COM+ (Component Object Model) da Microsoft (Iseminger, 2000).

4.2
Síntese de Design por Algoritmos Genéticos Co-Evolucionários

A parte mais importante do modelo descrito nesta dissertação é a modelagem do Algoritmo Genético (AG). O AG é responsável pela otimização e síntese da solução desejada.

O processo evolucionário realizado pelo AG possui alguns componentes que devem ser desenvolvidos cuidadosamente para o seu correto funcionamento. Os componentes são: Representação, Operadores Genéticos, Decodificação e

Avaliação. Esses componentes foram descritos no capítulo 3 e sua modelagem neste trabalho será descrita nesta seção.

4.2.1
Representação das Espécies

Um dos aspectos cruciais da solução de problemas por Algoritmos Genéticos é a escolha da *representação* do cromossomo que será evoluído e a partir do qual será extraída a solução.

Como o modelo utilizado neste trabalho requer o tratamento de vários aspectos de programação orientada a objetos, como herança, dependências, e a definição de onde alocar cada método e atributo, decidiu-se utilizar técnicas avançadas de Algoritmos Genéticos, como o modelo Co-Evolucionário, permitindo a decomposição do problema. Esse modelo Co-Evolucionário tem um aspecto importante que deve ser levado em conta e que está relacionado com a evolução de subcomponentes interdependentes. Se for possível decompor o problema em subcomponentes independentes, cada um deles pode evoluir sem haver necessidade de se preocupar com os outros; esta característica foi buscada na modelagem pesquisada nesta dissertação.

Assim, a solução proposta neste trabalho foi modelada através de um Algoritmo Genético Co-Evolutivo com quatro espécies, a saber:

- Classes
- Dependências
- Métodos
- Atributos

Cada espécie representa uma parte do problema a ser evoluída e deve possuir uma representação independente. Uma visão geral da modelagem do Algoritmo Genético Co-Evolutivo pode ser vista na Figura 14. As representações de cada espécie estão descritas a seguir.

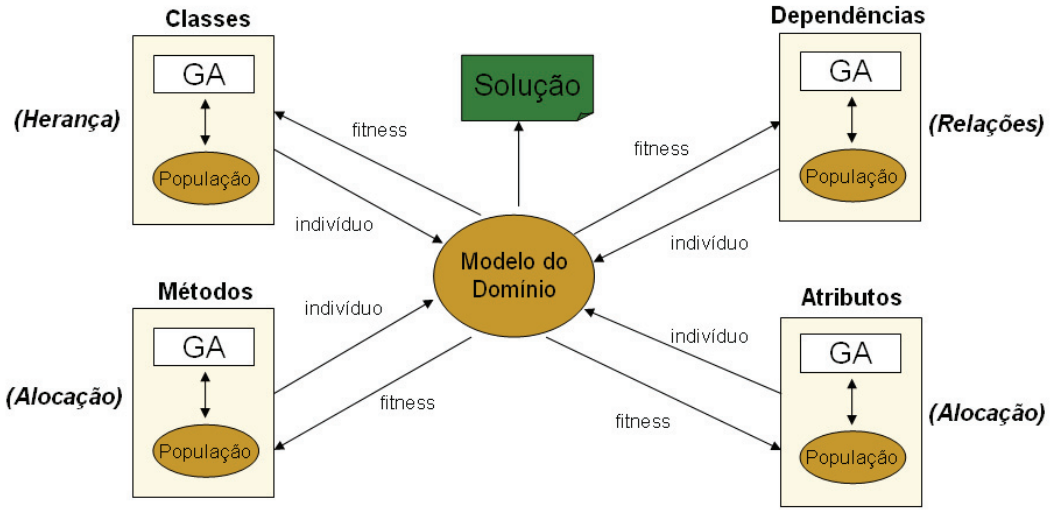

Figura 14 – Visão geral do Modelo Genético com as quatro espécies

Espécie Classes

A espécie *Classes* trata do problema de definir "*quem herda de quem*" e o número de classes do componente. Um diagrama de classes, onde as heranças são mostradas, pode ser visto como vários grafos desconectados e sem ciclos, ou seja, várias árvores, como mostrado no exemplo simplificado da Figura 15. No caso, as setas representam a relação de herança.

Esta espécie é representada pelo modelo baseado em ordem, muito utilizado e estudado para problemas similares ao do Caixeiro Viajante. Portanto, o cromossomo consiste em uma lista ordenada, onde cada gene representa uma classe, e a posição do gene indica a prioridade da classe; quanto mais à esquerda, maior a prioridade. Um exemplo de cromossomo da espécie *Classes* é mostrado na Tabela 8. Essa lista ordenada indica a ordem em que a classe será escolhida para a montagem das árvores do diagrama de classes.

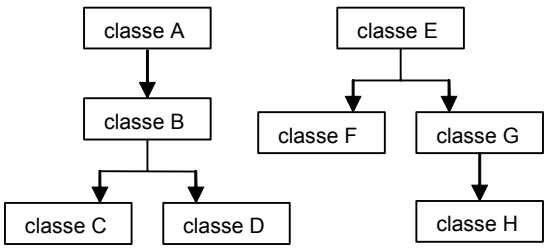

Figura 15 – Diagrama de classes simplificado

Classe	B	C	F	A	G	H	D	E
Prioridade	1	2	3	4	5	6	7	8

Tabela 8 – Exemplo de cromossomo com máximo de 8 classes

Como já foi dito, o número de classes da modelagem deve poder ser variado, por isso, a quantidade de classes deve ser um parâmetro também a ser evoluído. Para resolver esse problema, foi utilizado um cromossomo de tamanho variável abordado por Zebulum (1999) em sua pesquisa aplicada à evolução de circuitos eletrônicos. Nesta abordagem, cada elemento da lista pode se encontrar ativo ou inativo, e para isso é utilizada uma *máscara de ativação.* Quando um elemento é desativado ele passa a não ser considerado na solução. Essa máscara, assim como o cromossomo, possui um tamanho máximo definido por um parâmetro *MaxClasses*. Um exemplo de máscara de ativação pode ser vista na Tabela 9.

Classe	A	B	C	D	E	F	G	H
Máscara	1	0	0	1	1	1	0	0

Tabela 9 – Exemplo de máscara de ativação ativando 4 classes

Deve-se notar que a máscara ativa ou desativa as classes no cromossomo, e não as posições em si. A ordem ocupada pelas classes na lista que representa a máscara de ativação é a ordem alfabética, e isso não muda durante a evolução, permitindo a ativação/desativação das classes e não das posições. Por exemplo, na máscara de ativação da Tabela 9, as classes A, D, E e F estão ativas, embora no cromossomo da Tabela 8 essas classes ocupem as posições 4, 7, 8 e 3, respectivamente.

Essas duas listas permitem a evolução da ordem em que as classes serão inseridas no diagrama e o número e quais classes estão ativas. Além disso, é necessário representar as relações de herança entre as classes. A relação de herança, como foi detalhado no capítulo 2, é uma relação entre duas classes e, segundo a engenharia de *software,* são representadas no *Diagrama de Classes* UML. Na Figura 15, por exemplo, a classe B *herda* da classe A. Para representar as heranças de todo o diagrama, outra máscara binária foi utilizada.

Pode-se considerar a visão simplificada mostrada na Figura 15 como sendo uma coleção de grafos direcionados e representá-la através de uma matriz de incidência. Essa matriz de incidência seria uma nova máscara e um exemplo dela pode ser visto na Tabela 10.

	A	B	C	D	E	F	G	H
A	0	0	0	0	0	0	0	0
B	1	0	0	0	0	0	0	0
C	0	1	0	0	0	0	0	0
D	0	1	0	0	0	0	0	0
E	0	0	0	0	0	0	0	0
F	0	0	0	0	1	0	0	0
G	0	0	0	0	1	0	0	0
H	0	0	0	0	0	0	1	0

Tabela 10 – Máscara representando uma matriz de incidência correspondente ao diagrama de classes da Figura 15

O indivíduo deste espécime possui, portanto, três partes: a lista de classes, uma máscara de ativação e uma máscara de heranças. Cada parte destas sofre a atuação de operadores genéticos específicos separadamente. Os operadores genéticos que atuam na lista de classes são operadores baseados em ordem; já os operadores que atuam nas máscaras são operadores binários. Um detalhamento destes operadores será feito mais adiante neste capítulo.

A leitura deste cromossomo e a montagem do grafo são feitas por um decodificador, utilizando a máscara como matriz de incidência e respeitando a ordem definida pela lista ordenada do cromossomo. Este decodificador utiliza as quatro espécies para a montagem da solução. Seu algoritmo será mostrado adiante.

As máscaras exibidas nas Tabelas 9 e 10 podem ser vistas como extensões do cromossomo da espécie *Classes*, ou seja, cada cromossomo desta espécie possui uma máscara de ativação e uma máscara para a matriz de incidência. Essas duas máscaras poderiam ser consideradas como duas novas espécies, mas isso aumentaria ainda mais o custo computacional da evolução e, por esse motivo, elas foram mantidas acopladas ao cromossomo da espécie *Classes*.

Espécie Dependências

A relação de dependência no diagrama de classes indica que, se a classe A *depende* da classe B, de alguma forma a classe A usa a classe B e, provavelmente, se ocorrerem modificações na classe B, a classe A deverá sofrer adaptações. A representação da espécie *Dependências* indica as dependências entre as classes, portanto contém uma relação entre classes de N:N. Desse modo, foi utilizada uma matriz binária, de dimensão *MaxClasses* x *MaxClasses*, onde cada gene corresponde a uma classe, e é representado por outra lista de classes. Cada alelo possui valor 1 se uma classe depende da outra, ou 0 caso contrário. O número de classes ativas é indicado pela máscara de ativação da espécie *Classes*.

Um exemplo da representação descrita é mostrado na Tabela 11 e a representação gráfica das dependências pode ser vista na Figura 16. Na Figura 16, a direção da seta indica a direção onde a dependência ocorre, por exemplo, a Classe 6 *depende* da Classe 5, e assim por diante.

	C_1	C_2	C_3	C_4	C_5	C_6
C_1	0	1	0	0	0	0
C_2	1	0	0	0	0	0
C_3	0	0	0	1	1	0
C_4	0	1	0	0	0	0
C_5	0	0	0	0	0	0
C_6	1	0	0	0	1	0

Tabela 11 – Exemplo de cromossomo da espécie Dependências

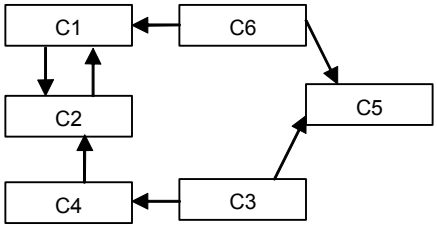

Figura 16 – Representação gráfica das dependências do cromossomo da Tabela 11

Espécie Métodos

A espécie *Métodos* deve indicar *onde* será colocado cada método, ou seja, sua distribuição pelas classes do sistema.

O cromossomo possui uma representação similar à espécie *Classes*, sendo constituído por uma lista de métodos que serão permutados formando diversas configurações. Além dos métodos, essa lista possui marcadores que têm a função de representar a divisão de classes. A função dos marcadores ficará mais clara no detalhamento do decodificador. O número de marcadores presentes na lista é *MaxClasses – 1.* Naturalmente, cada marcador possui um código único que não pode ser repetido, assim como cada método.

Essa lista contém os métodos da *especificação inicial* e a ordem na qual essa lista aparece no gene indica a *prioridade* em que se deve tentar utilizar o método. Um exemplo de um cromossomo da espécie *Métodos* está representado na Tabela 12. As posições contendo um **M** representam os marcadores.

M1	M4	**M**	M5	**M**	M3	M6	M2	**M**	**M**	M7	**M**

Tabela 12 – Exemplo de cromossomo da espécie Métodos, com MaxClasses = 6

Espécie Atributos

A espécie *Atributos* deve indicar *onde* será colocado cada atributo e é bem similar a espécie *Métodos*.

A representação do cromossomo é idêntica à representação dos métodos; novamente tem-se uma lista de prioridade de atributos com marcadores que indicam as classes. Os atributos que compõem a lista pertencem à *especificação inicial,* e a ordem na qual essa lista aparece no gene indica a *prioridade* em que se deve tentar alocar o atributo na classe. A Tabela 13 exemplifica este cromossomo.

A3	**M**	A7	A1	A5	**M**	A6	**M**	A2	**M**	A4	**M**

Tabela 13 – Exemplo de cromossomo da espécie Atributos, com MaxClasses = 6

4.2.2 Operadores Genéticos

Cada espécie possui operadores genéticos específicos para a sua representação. Abaixo, são apresentados os operadores genéticos utilizados para cada espécie.

A primeira espécie a ser considerada é a espécie responsável pelas classes do sistema. Como já foi dito, a espécie *Classes* possui um cromossomo baseado em ordem, similar à representação utilizada para o Problema do Caixeiro Viajante (Travelling Salesman Problem – TSP) (Fagerholt & Christiansen, 2000) (Moon et al, 2002), onde o algoritmo genético se encarrega de gerar novas permutações de uma lista ordenada, que no caso do Problema do Caixeiro Viajante representa um caminho de cidades a ser percorrido na ordem descrita.

A espécie *Classes* se utiliza da mesma representação, com a diferença de que cada cidade do caminho é substituída por uma Classe a ser criada.

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

O operador PMX constrói descendentes através da seleção de uma subseqüência de um caminho de um pai e mantendo a ordem e a posição de quantas cidades for possível do outro pai. Esta subseqüência é escolhida selecionando-se dois pontos de corte aleatoriamente. Por exemplo, sejam os dois cromossomos abaixo (os pontos de corte estão indicados pelo sinal |):

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

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

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

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

$$1 \leftrightarrow 4;8 \leftrightarrow 5;7 \leftrightarrow 6;6 \leftrightarrow 7 \tag{3}$$

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

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

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

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

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

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

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

Este operador explora o fato de que a ordem das cidades é que é importante e não a sua posição. Isto pode ser observado nos caminhos 9-3-4-5-2-

1-8-7 e 4-5-2-1-8-7-6-9-3 que, apesar de serem diferentes com relação a posição, seguem a mesma ordem e portanto representam o mesmo caminho.

Finalmente, o operador CX constrói descendentes de modo que cada cidade e a sua posição venham de um dos pais. Isto funciona conforme descrito abaixo. Sejam os dois cromossomos abaixo:

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

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

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

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

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

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

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

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

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

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

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

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

O cromossomo da espécie *Classes* possui dois tipos de máscaras que também sofrem a atuação de operadores genéticos específicos de mutação (não foi utilizado operadores de *crossover* para as máscaras). As máscaras são representações binárias, em uma e duas dimensões, respectivamente, como mostrado nas Tabelas 10 e 11 acima. Portanto os operadores genéticos são mais simples e intuitivos. São eles: ADD, SUB, FLIPBITS.

O operador ADD, fica encarregado de adicionar um valor 1 à máscara. Assim, se a máscara é a máscara de ativação de classes, ele está incluindo uma nova classe no sistema. Pode-se ver em (11) uma máscara antes e depois da aplicação do operador.

$$\begin{aligned}&(1\ 0\ \underline{0}\ 1\ 0\ 0\ 1\ 1\ 0) \quad \text{- Antes}\\ &(1\ 0\ \underline{1}\ 1\ 0\ 0\ 1\ 1\ 0) \quad \text{- Depois}\end{aligned} \tag{11}$$

O operador SUB é o contrário do ADD, ele fica encarregado de adicionar um valor 0 a máscara. Assim, se a máscara é a máscara de ativação de classes, ele está removendo uma classe do sistema. Pode-se ver em (12) uma máscara antes e depois da aplicação do operador.

$$\begin{array}{l}(1\ 0\ \underline{1}\ 1\ 0\ 0\ 1\ 1\ 0) \quad \text{- Antes} \\ (1\ 0\ \underline{0}\ 1\ 0\ 0\ 1\ 1\ 0) \quad \text{- Depois}\end{array} \tag{12}$$

Observa-se que em ambos os operadores uma posição do cromossomo é sorteada e o valor contido nesta posição é transformado em 1 ou 0 dependendo do operador, mas, pode acontecer da posição já possuir o novo valor, e assim, o cromossomo resultante será idêntico ao genitor.

O operador FLIPBITS atua invertendo o valor de um gene sorteado. Esse operador pode acabar inserindo ou removendo uma classe do sistema, caso a máscara seja a máscara de ativação de classes. Com o operador FLIPBITS não ocorre o problema citado acima para os operadores ADD e SUB, pois é garantido que o cromossomo resultante sempre terá um gene com o valor invertido em relação ao genitor.

A espécie *Dependências* também possui a representação de uma matriz binária, portanto nela também é utilizada o operador de mutação FLIPBITS, já explicado acima. Agora, um operador de *crossover* é utilizado, o ONEPOINT *crossover* para duas dimensões. A Figura 17 mostra a atuação do ONEPOINT *crossover*. Como o cromossomo possui duas dimensões, deve-se sortear dois pontos de corte e assim realizar a troca de material genético.

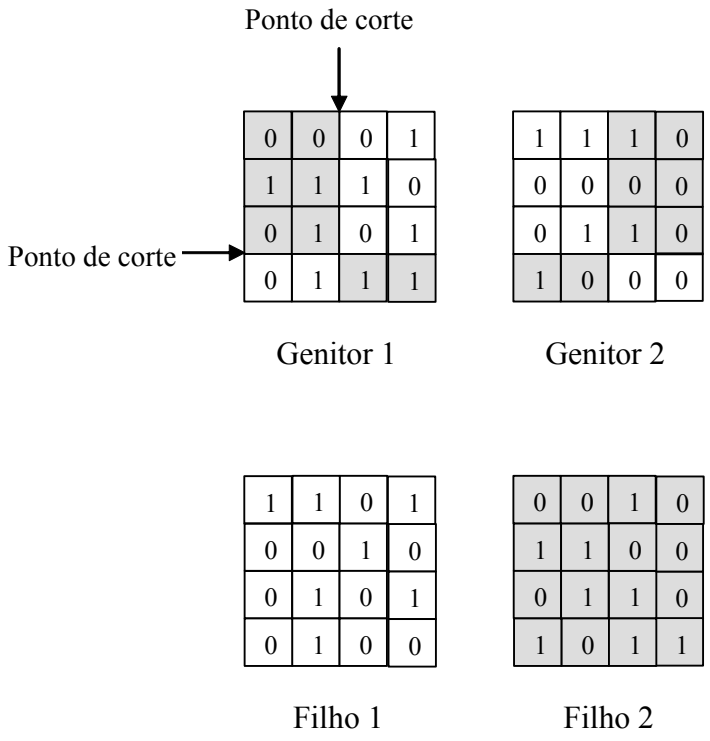

Figura 17 – Exemplo do ONEPOINT crossover em duas dimensões

Os operadores genéticos utilizados para as espécies *Métodos* e *Atributos* são os mesmos, já que as representações dos cromossomos são as mesmas. Os operadores utilizados são os mesmos utilizados na espécie *Classes* que também emprega uma representação baseada em ordem.

Na próxima seção será descrito como é realizada a decodificação dos cromossomos das quatro espécies, e assim, como é formada uma solução para o problema que será avaliada pela função de avaliação.

4.2.3 Decodificação

A decodificação consiste basicamente na construção da solução real do problema a partir do cromossomo. O processo de decodificação constrói a solução para que esta seja avaliada pelo problema.

No modelo desenvolvido nesta dissertação, as quatro espécies possuem representações indiretas da solução, e cada uma representa uma parte da solução. Portanto, o decodificador precisa utilizar as quatro espécies para a construção da solução a ser avaliada.

O algoritmo de decodificação possui os seguintes passos:

- Adicionar classes ativas na solução

- Montar relações de herança

- Montar relações de dependência

- Alocar métodos

- Alocar atributos

- Estabelecer métodos e atributos da super-classe para a classe

Segue um detalhamento dos passos acima:

- Adicionar classes ativas na solução

. Percorrer o cromossomo da espécie Classes na ordem da esquerda para direita

. Consultar posição da classe na máscara de ativação de classes

. Se a classe estiver ativa

. Adiciona classe na solução

- Montar relações de herança

. Para cada classe i na solução

. Percorrer linha referente à classe na máscara de herança

. Para cada coluna j com valor '1'

. Se não ocorre formação de ciclos

. Classe j é superclasse da classe i

- Montar relações de dependência

. Para cada classe i na solução

. Percorrer linha referente à classe no cromossomo da espécie Dependências

. Para cada coluna j com valor '1'

. Classe j é dependente da classe i

- Alocar métodos

. ClassIndex = 0

. Percorrer o cromossomo da espécie Métodos na ordem da esquerda para direita

. Se o gene for um marcador

. ClassIndex = ClassIndex + 1

. ClassIndex = ClassIndex MOD (# de classes na solução)

. Adiciona método na classe referente a ClassIndex

- Alocar atributos

. ClassIndex = 0

. Percorrer o cromossomo da espécie Atributos na ordem da esquerda para direita

. Se o gene for um marcador

. ClassIndex = ClassIndex + 1

. ClassIndex = ClassIndex MOD (# de classes na solução)

. Adiciona atributo na classe referente a ClassIndex

Esse algoritmo constrói a solução que poderá ser avaliada pelas métricas na função de avaliação. O método de avaliação está descrito na próxima seção.

4.2.4
Avaliação Multi-objetivo

A avaliação da solução é uma das fases principais de um AG. É a avaliação que indica se uma solução é boa ou não, portanto, é ela que direciona o processo evolucionário a fim de chegar a uma solução ótima.

A função de avaliação considera os atributos de qualidade descritos no capítulo 2. O cálculo dos atributos é baseado na pesquisa realizada por Bansiya (2002) no QMOOD. Os atributos de qualidade a serem calculados são: Reutilização, Flexibilidade, Inteligibilidade, Funcionalidade, Extensibilidade e Efetividade. De acordo com a pesquisa de Bansiya esses atributos podem ser inferidos através de propriedades da modelagem. Essas propriedades podem ser vistas na Tabela 14.

Propriedade	Definição
Tamanho da modelagem	A medida do número de classes usadas na modelagem.
Hierarquias	Hierarquias são usadas para representar diferentes conceitos de generalização-especialização numa modelagem. É a contagem do número de classes que possuem descendentes, mas não possuem super-classe na modelagem.
Abstração	A medida do aspecto de generalização-especialização de uma modelagem. Classes que possuem uma ou mais descendentes numa modelagem possuem essa propriedade de abstração.
Encapsulamento	Em modelagens de orientação a objetos essa propriedade se refere a modelar classes que previnem o acesso às declarações dos atributos através da sua declaração como privados, protegendo assim a representação interna dos objetos.
Modularidade	É a medida do desvio da média do número de serviços providos por classes num projeto. A intenção é identificar modelagens que possuem classes com grande desvios da média do número de serviços.
Acoplamento	Define a inter-dependência de um objeto em outros objetos numa modelagem. É a medida do número de outros objetos que precisam ser acessados por um objeto para que este funcione corretamente.
Coesão	Infere a relação entre métodos e atributos numa classe. Forte sobreposição nos parâmetros dos métodos e tipos de atributos é uma indicação de forte coesão.
Composição	Mede as relações “é parte de”, “tem”, “consiste em” e “parte-todo”, que são relações de agregação numa modelagem de orientação a objetos.
Herança	Uma medida da relação “é uma” entre classes. Essa relação está relacionada com o nível de aninhamento de classes numa hierarquia de heranças.
Polimorfismo	A habilidade de substituir um objeto por outro que possue a mesma interface em tempo de execução. É a medida de serviços que são dinamicamente determinados em tempo de execução num objeto.

Comunicação (*Messaging*)	É a contagem do número de métodos públicos que estão disponíveis como serviços para outras classes. Essa é a medida dos serviços que uma classe oferece.
Complexidade	Uma medida do grau de dificuldade no entendimento e compreensão da estrutura interna e externa das classes e as relações entre elas.

Tabela 14 – Definição das Propriedades de uma modelagem

Em sua pesquisa, Bansiya inferiu uma relação entre as propriedades da modelagem descritas na Tabela 14 e os atributos de qualidade já mencionados. Essa relação está detalhada nas equações da Tabela 15 abaixo.

Atributo de qualidade	Equações
Reutilização	-0.25 * Acoplamento + 0.25 * Coesão + 0.5 * Comunicação + 0.5 * Tamanho da modelagem
Flexibilidade	0.25 * Encapsulamento - 0.25 * Acoplamento + 0.5 * Composição + 0.5 * Polimorfismo
Inteligibilidade	-0.33 * Abstração + 0.33 * Encapsulamento - 0.33 * Acoplamento + 0.33 * Coesão - 0.33* Polimorfismo - 0.33 * Complexidade - 0.33 *Tamanho da modelagem
Funcionalidade	0.12 * Coesão + 0.22 * Polimorfismo + 0.22 * Comunicação + 0.22 * Tamanho da modelagem + 0.22 * Hierarquias
Extensibilidade	0.5 * Abstração - 0.5 * Acoplamento + 0.5 * Herança + 0.5 * Polimorfismo
Efetividade	0.2 * Abstração + 0.2 * Encapsulamento + 0.2 * Composição + 0.2 * Herança + 0.2 * Polimorfismo

Tabela 15 – Equações para inferência dos atributos de qualidade

Para o cálculo das propriedades da modelagem nas equações, são utilizadas métricas que podem estimar esses valores numa dada modelagem. Existem diversas métricas que medem de alguma forma o software. Em sua pesquisa, Bansiya estudou um conjunto de mais de 30 métricas, e selecionou as que melhor estimam as principais propriedades de uma modelagem estrutural de software. Essas métricas, estão descritas na Tabela 16.

Sigla	Nome	Descrição
DSC	Tamanho da modelagem em classes (*Design Size in Classes*)	Número de métodos públicos em uma classe
NOH	Número de hierarquias (*Number of Hierarchies*)	Número de hierarquias de classes na modelagem
ANA	Número médio de ancestrais (*Average Number of Ancestors*)	Número médio de classes das quais a classe herda informação
DAM	Métrica de acesso a dados (*Data Access Metric*)	Razão do número de atributos privados (protegidos) pelo número total de atributos declarados na classe. Um valor alto é desejado.
MFM	Medida de modularidade funcional (*Measure of Functional*	Essa métrica computa a modularidade baseada no desvio do número de métodos numa classe do número médio de métodos por classe na modelagem.

	Modularity)	Um valor perto de zero é preferido. Um valor pequeno indica um menor desvio entre classes no número de serviços oferecidos.
DCC	Acoplamento direto da classe (*Direct Class Coupling*)	Contagem do diferente número de classes que a classe está diretamente relacionada. A métrica inclui classes que são diretamente relacionadas por declaração de atributos e passagem de mensagens (parâmetros) em métodos.
CAM	Coesão entre métodos da classe (*Cohesion Among Methods in Class*)	Essa métrica computa a relação entre métodos de uma classe baseada nas listas de parâmetros dos métodos. A métrica é calculada usando o somatório da interseção de parâmetros do método com o conjunto máximo independente de todos os tipos de parâmetros na classe. Um valor próximo de 1 é preferível.
MOA	Medida de agregação (*Measure of Aggregation*)	Essa métrica mede a extensão da relação parte-todo, realizada através do uso de atributos. A métrica é uma contagem do número de dados declarados cujos tipos são classes definidas pelo desenvolvedor.
MFA	Medida de abstração funcional (*Measure of Functional Abstraction*)	Razão entre o número de métodos herdados pela classe e o total de métodos acessíveis por membros na classe.
NOP	Número de métodos polimorfos (*Number of Polymorphic Methods*)	Contagem dos métodos que podem exibir um comportamento polimorfo. Tais métodos em C++ são definidos como *virtual*.
CIS	Tamanho da interface da classe (*Class Interface Size*)	Contagem do número de métodos públicos na classe.
NOM	Número de métodos (*Number of Methods*)	Contagem de todos os métodos definidos na classe.

Tabela 16 – Descrição das métricas

A relação entre essas métricas e as propriedades da modelagem é o que falta para poder-se calcular valores para os atributos de qualidade. Essa relação é intuitiva e está descrita na Tabela 17.

Propriedade da modelagem	Métrica
Tamanho da modelagem	DSC
Hierarquias	NOH
Abstração	ANA
Encapsulamento	DAM
Modularidade	MFM
Acoplamento	DCC
Coesão	CAM
Composição	MOA
Herança	MFA
Polimorfismo	NOP
Comunicação (*Messaging*)	CIS
Complexidade	NOM

Tabela 17 – Relação das métricas com as propriedades da modelagem

Pode-se notar que a propriedade *Modularidade* não aparece nas equações na Tabela 16, portanto, isso exclui a métrica *MFM*. Outra consideração a ser feita é com relação às métricas *CIS* e *NOM*, pois no caso desta pesquisa, onde só estão sendo considerados os métodos públicos das classes, essas duas métricas refletem a mesma medida, portanto somente a métrica *CIS* será utilizada.

Aplicando a relação acima, as equações para estimar os atributos de qualidade ficam da seguinte forma:

Reutilização *= -0.25 *DCC + 0.25 *CAM + 0.5 *CIS + 0.5 *DSC*
Flexibilidade *= 0.25 *DAM - 0.25 *DCC + 0.5 * MOA + 0.5 *NOP*
Inteligibilidade *= -0.33 * ANA + 0.33 * DAM - 0.33 * DCC + 0.33 * CAM - 0.33 * NOP - 0.33 * CIS - 0.33 * DSC*
Funcionalidade *= 0.12 * CAM + 0.22 * NOP + 0.22 * CIS + 0.22 * DSC + 0.22 * NOH*
Extensibilidade *= 0.5 * ANA - 0.5 * DCC + 0.5 * MFA + 0.5 * NOP*
Efetividade *= 0.2 * ANA + 0.2 * DAM + 0.2 * MOA + 0.2 * MFA + 0.2 * NOP*

As métricas são obtidas a partir de componentes básicos da modelagem. Esses componentes básicos são: atributos, métodos e classes. Essa relação provoca certas contradições entre as métricas, ou seja, duas métricas que são baseadas em um mesmo componente da modelagem podem ser opostas, e uma não pode melhorar sem prejudicar a outra. A Tabela 18 mostra as relações entre métricas e os componentes da modelagem.

Métrica	Atributo	Método	Classe
DSC			X
NOH			X
ANA			X
DAM	X	X	X
MFM			X
CAM		X	X
DCC	X	X	X
CIS		X	X
MFA			X
MOA	X		
NOP		X	X
NOM	X	X	X

Tabela 18 – Relação das métricas com os componentes básicos da modelagem

As métricas são calculadas e normalizadas, para serem aplicadas nas equações dos atributos de qualidade. Assim, os valores para cada métrica varia entre 0 e 1.

Cada atributo de qualidade representa um objetivo a ser otimizado pelo AG. Assim, a função de avaliação deve utilizar uma estratégia para tratar múltiplos objetivos.

Algumas penalizações são utilizadas na função de avaliação para eliminar soluções indesejáveis: soluções que não possuam classes, soluções que possuam classes vazias, ou seja, sem conter métodos e atributos, e soluções com dependências inválidas. Dependências inválidas são: dependência de uma classe com ela mesma, ou dependência de uma classe com uma outra classe pertencente a mesma hierarquia. Essas penalizações atuam nas equações do cálculo dos atributos de qualidade, subtraindo o valor da penalização normalizado do valor do atributo. Isso evita que o AG procure soluções em partes do espaço de busca que, pelo conhecimento especialista, não fazem sentido.

No capítulo 3 foram discutidas estratégias de avaliação multi-objetivo, sendo que, nesta pesquisa, foi aplicada a Técnica de Pareto. O processo de avaliação dos indivíduos se divide em três etapas:

1. Cálculo do valor de cada objetivo para todos os indivíduos da população
2. Determinação dos indivíduos pertencentes ao conjunto Pareto-ótimo (não dominados)
3. Diferenciação dos indivíduos pertencentes ao conjunto Pareto-ótimo

A primeira etapa simplesmente consiste do cálculo das métricas e aplicação das equações para determinação dos atributos de qualidade.

A segunda etapa segue um algoritmo descrito por Fonseca (Fonseca & Fleming, 1993) para determinação do conjunto Pareto-ótimo. Nesse algoritmo é realizado um *ranking* entre os indivíduos e, ao final, tem-se o conjunto desejado. O algoritmo utilizado é descrito a seguir:

Percorrer todos os indivíduos aplicando a seguinte fórmula:

$rank(i) = 1 + dom(i)$

$dom(i)$: número de indivíduos que dominam o individuo i.

Nota-se que o conjunto Pareto Ótimo possui *rank = 1*.

Ordenar a população de acordo com o *rank.*

A terceira etapa é necessária a fim de evitar grandes oscilações na escolha do melhor indivíduo em cada geração. Sem essa diferenciação entre os indivíduos do conjunto Pareto-ótimo, todos possuem a mesma chance de serem escolhidos como melhor indivíduo; portanto isso provoca uma grande variação no decorrer das gerações. Para realizar essa diferenciação é criado um novo *ranking* dentro do conjunto Pareto-ótimo de acordo com o seguinte algoritmo:

Para cada indivíduo i pertencente ao conjunto Pareto-ótimo

Calcular o número de indivíduos de toda a população que o indivíduo i domina

Elaborar novo *rank*, favorecendo o individuo com maior dominância na população

Reordenar a população considerando o *rank* entre os indivíduos não dominados

O *fitness* de cada individuo recebe o valor da interpolação a partir do melhor (*rank = 1*) até o pior (*rank = n* $\leq$ *pop_size*) da maneira usual, de acordo com uma função linear.

Calcular a média dos *fitness* dos indivíduos com o mesmo *rank*, de modo que possuam a mesma chance no processo de seleção.

Ao final das três etapas a população estará ordenada e o melhor indivíduo estará na primeira posição e com maiores chances de ser escolhido pelo operador de seleção. Para favorecer ainda mais o conjunto dos não dominados, de modo que eles tenham mais chances de continuar na população, é feita uma interpolação do *fitness* de modo que o conjunto Pareto-ótimo ganhe maiores valores e se distanciem mais do resto da população. Um exemplo de interpolação pode ser visto abaixo:

Seja $P = \{I_1, I_2, I_3, I_4\}$ o conjunto Pareto-ótimo já ordenado segundo o algoritmo descrito na terceira etapa.

Percorrer *P* na ordem, preenchendo o *fitness* usando a interpolação: 2**pop_size, 2*pop_size-2, 2*pop_size-4, 2*pop_size-6.*

Seja I_5 o próximo indivíduo da população, que não pertence a *P*.

Preencher o *fitness* de I_5 com *pop_size-3*, e os seguintes com *pop_size-4*, *pop_size-5, ...*

Com o uso desta estratégia de Pareto não ocorre a mistura de objetivos, e pode-se até priorizar um ou outro objetivo. Durante a evolução pode-se observar cada objetivo sendo evoluído, embora quando se consideram todos os objetivos juntos um acaba influenciando na evolução do outro, pois são calculados através das mesmas métricas e o AG tenta encontrar um equilíbrio entre eles.

No próximo capítulo será apresentado o estudo de casos realizado para o modelo proposto.