Universidade de Coimbra
Faculdade de Ciências e Tecnologia
Departamento de Engenharia Informática

Qualidade e Confiabilidade de Software
Assignment 2 – Dynamic Software Testing

Docentes
Henrique Santos do Carmo Madeira
João Tiago Márcia do Nascimento Fernandes
Raul André Brajczewski Barbosa

Equipa

Carlos João Lopes Mendes, nº2017257272, cjmendes@student.dei.uc.pt
Filipe Miguel Fonseca dos Santos, nº2017271196, fmsantos@student.dei.uc.pt
Ricardo David da Silva Briceño, nº2020173503, briceno@student.dei.uc.pt

Índice

1 – Introdução

Este plano de testes tem como objetivo testar o código submetido como proposta, pelo nosso grupo, para o projeto 2 da cadeira de Qualidade e Confiabilidade de Software. Este código, escrito em python, é da criação de um estudante externo ao grupo, que o criou para resolver um problema relacionado com a cadeira de **Computação Evolucionária**. O problema que este algoritmo visa resolver é o **encontro do número mínimo de subconjuntos Si**, que quando reunidos formam um universo **U**, que contém os primeiros **N** números naturais. Por exemplo:

U = {1, 2, 3, 4, 5}

S1 = {1, 3, 5}

S2 = {2, 4}

S3 = {3, 4}

Solução = S1, S2

Em termos de abordagens, utilizámos tanto "**White Box Testing**" como "**Black Box Testing**". Especificamente, utilizámos técnicas de "**Control Flow**" e de "**Data Flow**", relacionadas com a primeira abordagem, e de "**Class Partitioning**" e "**Boundary Value Analysis**" para a segunda.

2 - Riscos relacionados com o Software

Em termos de **aspetos críticos** a serem testados, o foco está muito direcionado (1) à **funcionalidade do programa**, ou seja, a garantir que consegue realmente executar a tarefa que é suposto, (2) ao seu **control flow** e (3) ao seu **data flow**.

Em relação a **potenciais riscos**, podemos notar a existência de **estruturas mais complexas no código** a ser testado. Estas podem tornar mais difícil a criação de testes corretos, que validam realmente o algoritmo. Por exemplo, podem dificultar a criação dos grafos relativos às diversas técnicas de teste, devido à sua maior complexidade. Para este problema devemos prestar especial atenção a essas estruturas, de forma a prevenir a criação de testes incorretos.

```
while i<len(pheno):
    result = [x for x in result if x not in pheno[i]]
    i+=1
```

Figure 1: Exemplo de uma estrutura de código mais complexa

Como **não temos acesso a uma documentação completa** do algoritmo desenvolvido, mas apenas a pequenos comentários no código e ao enunciado que explica o problema, podemos dizer que há um risco de interpretar incorretamente o propósito de algum módulo. De forma a mitigar este risco temos de associar corretamente o que está especificado no problema às funções usadas no código, como também à relação entre estas.

Sendo este um **algoritmo fora da nossa área de especialização** (Sistemas Inteligentes), podem haver problemas de falta de conhecimento, o que pode tornar mais difícil a compreensão de certos passos tomados pelo algoritmo. Em geral, isto pode ser solucionado pelo uso de recursos com informação relacionada com computação evolucionária.

Em relação à parte de **Black Box Testing**, como é uma técnica que é especialmente eficiente quando utilizada por alguém com experiência, podemos dizer que há um **risco de não termos experiência suficiente com a técnica** para tirar partido completo desta. De forma a mitigar este problema, podemos contactar pessoas mais especializadas na área (como os nossos professores de QCS), com o objetivo de tentar esclarecer quaisquer dúvidas existentes.

Em caso destes problemas levarem a **testes imperfeitos**, isto pode ter a consequência da **não deteção de bugs**. O que por si pode levar à validação imprópria do software, que consequentemente leva à descoberta tardia de falhas, que se tornam mais dispendiosas de

resolver com o avanço da fase de desenvolvimento do software.

3 - Itens e Funcionalidades a serem testadas

Em termos do **control flow** do programa a ser testado, temos como objetivo testar as funções (units) ***fitness***, ***muta_bin*** e ***gera_pop***, que devido às chamadas que fazem, levam naturalmente ao teste das funções ***geno2pheno***, ***muta_bin_gene*** e ***gera_indiv***, respetivamente. Estas foram escolhidas pois não só contém funcionalidades chave para o algoritmo evolucionário (calcular a fitness de um indivíduo; alterar os dados de um indivíduo, através de uma mutação; gerar novos indivíduos para a população), como também são as funções mais complexas do programa em termos de VG (McCabe's cyclomatic complexity), como demonstrado nos nossos cálculos para cada função. Sendo que estes foram feitos com base no número de predicados dos respetivos control flow graphs. Em termos do end user, estas funções não têm um impacto óbvio, pois na verdade são apenas uma parte de um todo, sendo que o programa total em si é que garante a funcionalidade esperada.

Função	fitness	muta_bin	gera_pop
V(G)	8	3	3

Tabela 1: V(G) das funções escolhidas para Control Flow Testing (incluindo chamadas de funções internas)

Em relação ao **data flow** decidimos escolher a função de ***fitness***, uma vez que é não só essencial ao funcionamento do algoritmo, como também é a função com o maior número de variáveis, sendo a mais complexa em termos deste tipo de testes. É de notar que, neste caso, testamos apenas a função de ***fitness*** em si e não a sua chamada da função ***geno2pheno***. Esta decisão teve em conta o facto da função de ***fitness*** por si já ser complexa o suficiente (16 variáveis/parâmetros).

Já em relação ao **Black Box Testing**, em que fazemos **Class Partitioning** e **Boundary Value Analysis**, a nossa decisão foi de testar a funcionalidade de todo o programa. Sendo que, não só analisamos os inputs dados ao programa através dos ficheiros de input escolhidos (tamanho do universe e dos sets), como também as "variáveis globais" do programa (número de gerações, tamanho da população, probabilidade de mutação, probabilidade de crossover, tamanho do torneio e elitismo). **Na perspetiva de um utilizador final, estes testes cobrem a**

funcionalidade do sistema em si. Em concreto, avaliam se o algoritmo consegue realmente calcular o número mínimo de sets que unidos formam o universo.

4 - Itens e Funcionalidades a não serem testadas

Em relação ao Control Flow Testing, decidimos não testar as funções não mencionadas no capítulo anterior. Em geral, considerámos estas funções menos importantes para testes pois ou contêm funcionalidades menos relevantes (por exemplo, mostrar o resultado ou ler o ficheiro) ou a sua complexidade em termos de V(G) é demasiado baixa. Na primeira categoria temos as funções ***displayResult*** e ***readFile***, enquanto na segunda temos as funções ***tournament*** (selecionar os elementos que serão os "parents" da geração seguinte), ***one_point_cross*** (realiza o crossover), ***elitism*** (controla os indivíduos que continuam a existir na geração seguinte) e ***best_pop*** (retorna o melhor indivíduo). A exceção aqui é a função ***sea***, que trata de chamar todas as outras, sendo bastante complexa por si só. Em geral, devido à complexidade, e ao facto de para testar completamente esta função teríamos de também testar todas as outras, decidimos não a considerar.

Função	sea	displayResult	tournament	readFile	one_point_cross	elitism	best_pop
V(G)	6 *	2 *	2	2	2	1	1

Tabela 2: V(G) das funções não consideradas para Control Flow Testing (* sem contar com a chamada de funções)

Relativamente ao Data Flow Testing, **apenas testámos a função de *fitness***, não considerando as outras. Em geral, como as suas complexidades relativas a este tipo de teste são mais baixas (poucos atributos) e como temos limites em termos de tempo, optámos por apenas testar a mais complexa.

5 – Abordagem dos Testes

5.1 White Box Testing – Control Flow

Para estes testes a nossa **coverage** inclui as funções ***fitness***, ***muta_bin*** e ***gera_pop***, que por sua vez, como expandimos as suas chamadas de funções, nos dão também a coverage das funções ***geno2pheno***, ***muta_bin_gene*** e ***gera_indiv***, respetivamente.

A estratégia para estes testes passou por seguir a seguinte sequência de passos, que podem todos ser verificados nos **Anexos 1** (com exceção dos CFGs relativos às funções não escolhidas, que podem ser observados dentro da pasta "Control Flow", na "Graphs Drawn but not used"):

1) Desenhar o Control Flow Graph (CFG) de todas as funções do código
2) Avaliar os CFGs de todas essas funções em termos de V(G)
3) Exclusão das funções consideradas irrelevantes, quer devido ao baixo V(G), pouca importância à funcionalidade do código ou complexidade excessiva (*sea*)
4) Encontrar todos os caminhos independentes para as funções selecionadas, de forma a cobri-las na sua totalidade
5) Verificar que caminhos são executáveis e exclusão dos outros
6) Definição dos Test Cases, de forma a ter um test case por caminho
7) Adição de test cases adicionais, com intenção de explorar mais os loops presentes nos caminhos

Para estes testes considerámos uma **seed fixa, 500**. Sendo as execuções destas funções determinísticas (ao considerar a seed constante), a métrica para avaliar se um test case passou é simplesmente a comparação do outcome obtido pela função com o outcome esperado. O **oráculo** utilizado foram os **testes manuais efetuados pela nossa equipa**, que analisou os grafos obtidos e calculou o resultado esperado para cada função.

5.2 White Box Testing – Data Flow

Neste caso, a **coverage** contém apenas a função ***fitness***, não expandindo a chamada da função *geno2pheno*.

Para estes testes considerámos o critério de testes **ADUP**, já que é o mais genérico e poderoso entre os critérios disponíveis. Já para estes testes utilizámos a seguinte sequência de

passos:

1) Comparação das funções em termos de variáveis usadas
2) Escolha da função mais relevante (em termos de número de variáveis usadas e de relevância para o programa em si)
3) Desenhar o Data Flow Graph da função escolhida
4) Encontrar os DU paths para cada variável/parâmetro
5) Criação dos caminhos completos, de forma a cobrir todos os DU paths de todas as variáveis
6) Criação de um test case para cada caminho completo

Todos estes passos podem ser verificados nos **Anexos 2**.

Mais uma vez, utilizámos uma **seed fixa, 500**, sendo que neste caso acaba por não ter impacto, devido ao facto da **função** ser **determinística**. Utilizamos a comparação entre o outcome dado pelo algoritmo com o **calculado manualmente** pela nossa equipa como métrica para avaliar se um test case passou (neste caso esse output é o fitnessValue).

5.3 Black Box Testing – Class Partitioning e Boundary Value Analysis

Em termos de **coverage** para o **Class Partitioning** considerámos as seguintes classes, 18 das quais também se referem aos testes de **Boundary Value Analysis** feitos para os inputs válidos, enquanto as outras 21 tratam os inputs inválidos da partição de classes.

Input	Classes de Equivalência Válidas			Classes de Equivalência Inválidas	
	ID	Definição da Classe	Boundaries Escolhidos	ID	Definição da Classe
Número de sets, inteiros >= 0	VC1	[0, 5[	{-1, 0, 4}	IC1	]-∞, 0[
	VC2	[5, 10[	{5, 9}	IC2	Strings, valores não numéricos
	VC3	[10, +∞[	{10}	IC3	Null
Tamanho do Universe, inteiros >= 0	VC4	[0, 10[	{-1, 0, 9}	IC4	]-∞, 0[
	VC5	[10, 50[	{10, 49}	IC5	Strings, valores não numéricos
	VC6	[50, +∞[	{50}	IC6	Null
Número de Gerações, inteiros >= 0	VC7	[0, 100[	{-1, 0, 99}	IC7	]-∞, 0[
	VC8	[100, +∞[	{100}	IC8	Strings, valores não numéricos
Tamanho da População, inteiros >= 2, > Tamanho do Torneio	VC9	[2, 10[	{1, 2, 9}	IC9	]-∞, 2[
	VC10	[10, +∞[	{10}	IC10	Strings, valores não numéricos
Probabilidade de Mutação, floats, [0, 1]	VC11	[0, 0.01[	{-0.01, 0, 0.009}	IC11	]-∞, 0[
	VC12	[0.01, 1]	{0.01, 1, 1.01}	IC12	]1, +∞[
				IC13	Strings, valores não numéricos
Probabilidade de Crossover, floats, [0, 1]	VC13	[0, 0.9[	{-0.01, 0, 0.89}	IC14	]-∞, 0[
	VC14	[0.9, 1]	{0.9, 1, 1.01}	IC15	]1, +∞[
				IC16	Strings, valores não numéricos
Tamanho da Torneio, inteiros >= 1	VC15	[1, 3[	{0, 1, 2}	IC17	]-∞, 1[
	VC16	[3, +∞[	{3}	IC18	Strings, valores não numéricos
Elitismo, floats, [0, 1]	VC17	[0, 0.02[	{-0.01, 0, 0.019}	IC19	]-∞, 0[
	VC18	[0.02, 1]	{0.02, 1, 1.01}	IC20	]1, +∞[
				IC21	Strings, valores não numéricos

Tabela 3: Classes de Equivalência consideradas

Para estes testes seguimos os seguintes passos:

1) Criação de classes distintas para cada input do programa
2) Separação em classes de equivalência válidas e inválidas
3) Escolha dos boundaries a usar nos testes de Boundary Value Analysis para as classes de equivalência válidas
4) Definição de um test case para cada classe inválida (equivalência fraca)
5) Criação de um test case para cada Boundary selecionado para as classes válidas (equivalência média, mais próxima de equivalência fraca)

Os test cases podem ser observados nos **Anexos 3**.

Para os testes das **classes inválidas** optámos por utilizar apenas **uma seed, 500**. Optámos por utilizar 1 seed por 2 razões. A primeira razão é que o **programa não tem um foco em verificar os inputs.** A segunda razão é que **se o programa devolve um output que não seja erro, tendo inputs inválidos, isto demonstra imediatamente que o teste falhou**.

Em relação às **classes válidas**, em que realizamos **Boundary Value Analysis**, decidimos tratar os boundaries que correspondem a inputs inválidos tal como tratamos os inputs das classes inválidas. Já para os boundaries que devem corresponder a inputs válidos consideramos **10 seeds** (7575, 158, 4209, 4610, 8967, 5668, 938, 6559, 4849, 6811). Como a **natureza do programa é não determinística**, optámos por testar com várias seeds, de forma a validar de forma mais correta se o programa cumpre com as funcionalidades básicas.

Para computar o **output esperado**, que neste caso é o **número mínimo de sets** que conseguem reunir para formar o universo, **calculámos manualmente** a(s) **combinação/combinações de sets que levariam à solução desejada**, tendo em atenção os sets presentes nos vários ficheiros de input. Ao avaliar a solução obtida, temos em conta não só o número de sets apresentado, como **também se a união de sets apresentados é**, de facto, uma **solução válida** (forma o universo). Decidimos considerar o número de sets como outcome esperado, pois devido à **natureza não determinística do programa**, este pode apresentar **múltiplas soluções diferentes**, mas corretas, **consoante a seed** usada. Tendo isto em conta, o número de sets é um outcome esperado mais justo, pois é um valor comum a todas as soluções válidas, o que permite eliminar imediatamente outputs com valores diferentes.

6 – Critério de Pass/Fail

Em relação ao **Control Flow Testing**, consideramos que os testes para uma função (unit) foram **completados com sucesso se todos os test cases relativos à função obtiveram os resultados esperados**.

Para o **Data Flow Testing**, sendo aplicado a uma unit (***fitness***), consideramos que os testes tiveram sucesso se **todos os test cases tiveram resultados correspondentes aos esperados**.

Em relação ao **Black Box Testing de todo o programa**, em que usamos Class Partitioning, em relação às **classes de inputs inválidos**, avaliamos os **testes como sucesso caso estes apresentem erros**, já que os testes são realizados com valores incorretos. Se o programa conseguir correr com um destes testes, isso é considerado um defeito, pois o algoritmo está a aceitar valores que não devia. Neste caso, os defeitos são considerados **minor**, pois o programa claramente não tem um grande foco em validar inputs, mas sim em fornecer a funcionalidade esperada.

Já para as **classes de Inputs válidos** do Black Box Testing, em que realizamos **Boundary Value Analysis**, devido à **natureza não determinística do algoritmo** (pois é evolucionário), decidimos adaptar um critério menos restrito do que os outros, em que desde que o output **use o número mínimo de sets para formar o universo** (e consiga realmente fazê-lo) então o output está correto (pois podem haver várias combinações de sets que o permitam). Devido a essa natureza executámos cada teste com **10 seeds diferentes**, de forma a reduzir o impacto da aleatoriedade dos resultados obtidos e a tirar conclusões mais completas. Consideramos que um test case teve realmente sucesso se pelo menos **8 em 10 seeds obtiveram um resultado correto** (em termos de número de sets, podendo haver várias combinações válidas). Se tiver **menos do que 8** é considerado um **defeito grave**, pois o algoritmo está a ser incapaz de alcançar os resultados pretendidos, não fornecendo a funcionalidade esperada.

7 – Deliverables do Teste

Como parte deste relatório será entregue o **relatório (test plan)** em si. Serão também entregues diversos itens presentes no relatório, como todos os **test cases desenvolvidos**, **processo para chegar a estes e seus resultados**. É de notar que esta informação, em tabelas e imagens, se encontra nos **capítulos 5, 10 e anexos**. No capítulo 10 também se encontra o **Test Completion Report**, em que se encontram os defeitos encontrados.

Para além disso, entregamos também um Power Point em anexo (referido nos Anexos 2), "**fitness_du_paths.pptx**". Este documento engloba todo o **processo do Data Flow Testing**, com exceção do desenho do Data Flow Graph, que se encontra como anexo nos Anexos 2.

Também entregamos o **código desenvolvido para cada fase do teste**, presente em ficheiros **HTML**.

8 – Necessidades Ambientais

Em termos de software para a execução dos testes, usámos o **Jupyter Notebook** para criar e correr todos os **scripts** necessários à **automatização dos test cases**.

9 – Pessoal e Responsabilidades

Os testes foram realizados pelo nosso grupo de trabalho, sendo que todos os elementos participaram em todas as diferentes etapas.

10 – Test Completion Report

10.1 Testes para o Control Flow

Como se pode observar nas tabelas, todos os testes obtiveram os resultados esperados. Sendo assim, podemos dizer que, relativamente ao Control Flow Testing, não foram encontrados defeitos. Ou seja, podemos afirmar que os nossos testes validam o correto control flow das funções ***gera_pop***, ***muta_bin*** e ***fitness*** (e da ***gera_indiv***, ***muta_bin_gene*** e ***geno2pheno***, devido a terem sido expandidas nas 3 funções previamente mencionadas.)

	Inputs				
Test Case	size_pop	size_cromo	Outcome Esperado (returnValue)	Outcome	Correct
P1	0	0	[]	[]	Yes
P2	1	0	[([], 0)]	[([], 0)]	Yes
P2_b	5	0	[([], 0), ([], 0), ([], 0), ([], 0), ([], 0)]	[([], 0), ([], 0), ([], 0), ([], 0), ([], 0)]	Yes
P3	1	1	[([1], 0)]	[([1], 0)]	Yes
P3_b	1	5	[([1, 1, 1, 1, 0], 0)]	[([1, 1, 1, 1, 0], 0)]	Yes

Tabela 4: Resultados para a função gera_pop

	Inputs				
Test Case	indiv	prob_muta	Outcome Esperado (cromo)	Outcome	Correct
P1	[]	0.01	[]	[]	Yes
P2	[1]	0.01	[1]	[1]	Yes
P2_b	[1, 1, 1, 1, 1]	0.01	[1, 1, 1, 1, 1]	[1, 1, 1, 1, 1]	Yes
P3	[1]	0.9	[0]	[0]	Yes

Tabela 5: Resultados para a função muta_bin

	Inputs					
Test Case	universe	sets	solution	Outcome Esperado (fitnessValue)	Outcome	Correct
P1	[]	[]	[]	0	0.0	Yes
P3	[]	[[], []]	[1, 1]	0.4	0.4	Yes
P3_b	[]	[[], [], [], [], [], []]	[1, 1, 1, 1, 1, 1]	1.2	1.2000000000000002	Yes
P4	[1]	[[1]]	[1]	0.2	0.2	Yes
P4_b	[1, 2, 3, 4, 5]	[[1, 2, 3, 4, 5]]	[1]	0.2	0.2	Yes
P5	[1, 2]	[[2], [1]]	[1, 0]	1.2	1.2	Yes
P6	[]	[]	[0]	0	0.0	Yes

Tabela 6: Resultados para a função fitness

10.2 Testes para o Data Flow

Como o output de todos os testes foi o esperado, podemos dizer que, relativamente à função de fitness e apenas a essa, não foram detetados defeitos relacionados com o data flow.

Porém, ao encontrar os DU paths, foi detetada uma anomalia do tipo 3 (definição de uma variável que depois não chega a ser usada), relativamente à variável "i".

	Inputs					
Test Case	universe	sets	solution	Outcome Esperado (fitnessValue)	Outcome	Correct
P1	[]	[]	[]	0	0.0	Yes
P2	[1]	[[], [1]]	[1, 0]	1.2	1.2	Yes
P3	[]	[[], []]	[1, 1]	0.4	0.4	Yes
P4	[1]	[[], [], [], [1]]	[1, 1, 1, 0]	1.6	1.6	Yes
P5	[1, 2]	[[], [1, 2]]	[1, 0]	2.2	2.2	Yes

Tabela 7: Resultados para a função fitness

```
i=0

setSize = len(pheno)

fitnessValue = alfa*missing + beta*overlap + gama*setSize
return fitnessValue
```

Figure 2: Anomalia do tipo 3 (relativa à variável "i")

10.3 Testes de Black Box

Podemos verificar que **10/21 dos test cases não deram erro, como era esperado**. Isto pode dever-se à **falta de validação do input** dado por parte do programador. Estes defeitos são considerados **minor**, pois, como já referido, este programa não se foca na validação do input.

Podemos notar, por exemplo, que **a função aceita números negativos** (até mesmo probabilidades) sem qualquer problema. Porém, estas são situações em que claramente deveria indicar algum erro, já que não podem haver probabilidades negativas, nem tamanhos de universos ou de sets negativos, já que **esses valores não fazem sentido**.

10.3.1 Classes Inválidas

	Teste / Input	Número de Sets	Tamanho do Universe	Número de Gerações	Tamanho da População	Probabilidade de Mutação	Probabilidade de Crossover	Tamanho do Torneio	Elitismo	Nome do Ficheiro	Outcome Esperado	Outcome Obtido (Seed 500)
Número de Sets	**IC1**	**-1**	10	100	10	0.01	0.9	3	0.02	inv1.txt	**Erro**	**Sets Vazio**
	IC2	**word**	10	100	10	0.01	0.9	3	0.02	inv2.txt	**Erro**	**Sets Vazio**
	IC3	**Null**	10	100	10	0.01	0.9	3	0.02	inv3.txt	Erro	IndexError: list index out of range
Tamanho do Universe	**IC4**	5	**-1**	100	10	0.01	0.9	3	0.02	inv4.txt	**Erro**	**Sets Vazio**
	IC5	5	**word**	100	10	0.01	0.9	3	0.02	inv5.txt	Erro	invalid literal for int() with base 10: 'word'
	IC6	5	**NULL**	100	10	0.01	0.9	3	0.02	inv6.txt	Erro	IndexError: list index out of range
Número de Gerações	**IC7**	5	10	**-1**	10	0.01	0.9	3	0.02	inv7.txt	**Erro**	**3 Sets**
	IC8	5	10	**word**	10	0.01	0.9	3	0.02	inv7.txt	Erro	TypeError: 'str' object cannot be interpreted as an integer
Tamanho da População	**IC9**	5	10	100	**1**	0.01	0.9	3	0.02	inv7.txt	Erro	ValueError: Sample larger than population or is negative
	IC10	5	10	100	**word**	0.01	0.9	3	0.02	inv7.txt	Erro	TypeError: 'str' object cannot be interpreted as an integer
Probabilidade de Mutação	**IC11**	5	10	100	10	**-0,01**	0.9	3	0.02	inv7.txt	**Erro**	**3 Sets**
	IC12	5	10	100	10	**2**	0.9	3	0.02	inv7.txt	**Erro**	**3 Sets**
	IC13	5	10	100	10	**word**	0.9	3	0.02	inv7.txt	Erro	TypeError: '<' not supported between instances of 'float' and 'str'
Probabilidade de Crossover	**IC14**	5	10	100	10	0.01	**-0,01**	3	0.02	inv7.txt	**Erro**	**2 Sets**
	IC15	5	10	100	10	0.01	**2**	3	0.02	inv7.txt	**Erro**	**2 Sets**
	IC16	5	10	100	10	0.01	**word**	3	0.02	inv7.txt	Erro	TypeError: '<' not supported between instances of 'float' and 'str'
Tamanho do Torneio	**IC17**	5	10	100	10	0.01	0.9	**0**	0.02	inv7.txt	Erro	IndexError: list index out of range
	IC18	5	10	100	10	0.01	0.9	**word**	0.02	inv7.txt	Erro	TypeError: '<=' not supported between instances of 'int' and 'str'
Elitismo	**IC19**	5	10	100	10	0.01	0.9	3	**-0,01**	inv7.txt	**Erro**	**2 Sets**
	IC20	5	10	100	10	0.01	0.9	3	**2**	inv7.txt	**Erro**	**3 Sets**
	IC21	5	10	100	10	0.01	0.9	3	**word**	inv7.txt	Erro	ValueError: invalid literal for int() with base 10: 'wordwordwordwordwordwordwordwordwordwordword'

Tabela 8: Testes com as classes inválidas (NOTA: recomenda-se o uso de zoom a 200%)

10.3.2 Classes Válidas

Facilmente observamos que **apenas 6/42 dos testes tiveram resultados satisfatórios**. Estes resultados são ainda mais preocupantes do que aparentam. Desses 6, metade são simples testes com inputs inválidos nos boundaries, que retornam o erro esperado. 2 dos outros são casos em que retorna imediatamente um lista de sets vazia, por ter ou o número de sets ou o tamanho do universe igual a 0. O único caso em que realmente tem o outcome esperado é o VC6(50), no qual só não obteve o resultado esperado para uma seed (o que é o suficiente para o nosso critério de acertar pelo menos 8/10).

Todos os outros testes tiveram ou tudo errado ou, o mais usual, acertaram em 2-3 seeds e falharam todas as outras. Isto mostra que não só o algoritmo evolucionário pode estar mal otimizado, como também não está a controlar corretamente situações em que chega a outcomes que são impossíveis (pois apesar de darem o número de sets correto, a sua reunião não forma o universo) ou que estão completamente errados (pois dão um número diferente de sets do que o esperado). Como estes resultados estão abaixo do critério definido (na verdade são muito inferiores ao esperado), podemos dizer que **estes defeitos mencionados são graves**. Isto pois demonstram uma clara falta da funcionalidade esperada.

	Teste / Input	Número de Sets	Tamanho do Universe	Número de Gerações	Tamanho da População	Probabilidade de Mutação	Probabilidade de Crossover	Tamanho do Torneio	Elitismo	Nome do Ficheiro	Outcome Esperado (número mínimo de sets)
Número de Sets	**VC1 (-1)**	**-1**	10	100	10	0.01	0.9	3	0.02	1.txt	**Erro**
	VC1 (0)	**0**	10	100	10	0.01	0.9	3	0.019	2.txt	Sets vazio
	VC1 (4)	**4**	10	100	10	0.01	0.9	2	0.019	3.txt	**3 sets**
	VC2 (5)	**5**	10	100	10	0.01	1	2	0.019	4.txt	**2 sets**
	VC2 (9)	**9**	10	100	10	0.009	1	2	0.019	5.txt	**2 sets**
	VC3 (10)	**10**	10	100	9	0.009	1	2	0.019	6.txt	**2 sets**
Tamanho do Universe	**VC4 (-1)**	5	**-1**	100	10	0.01	0.9	3	0.02	7.txt	Erro
	VC4 (0)	5	**0**	100	10	0.01	0.9	3	0.019	8.txt	Sets Vazio
	VC4 (9)	5	**9**	100	10	0.01	0.9	2	0.019	9.txt	**2 sets**
	VC5 (10)	10	**10**	100	10	0.01	1	2	0.019	10.txt	**2 sets**
	VC5 (49)	10	**49**	100	10	0.009	1	2	0.019	11.txt	**3 Sets**
	VC6 (50)	10	**50**	100	9	0.009	1	2	0.019	12.txt	3 sets
Número de Gerações	**VC7 (-1)**	10	10	**-1**	10	0.01	0.9	3	0.02	10.txt	**Erro**
	VC7 (0)	10	10	**0**	10	0.01	0.9	3	0.019	10.txt	**2 sets**
	VC7 (99)	10	10	**99**	10	0.01	0.9	2	0.019	10.txt	**2 sets**
	VC8 (100)	10	10	**100**	10	0.01	1	2	0.019	10.txt	**2 sets**
Tamanho da População	**VC9 (1)**	10	10	99	**1**	0.01	0.9	3	0.02	10.txt	Erro
	VC9 (2)	10	10	99	**2**	0.01	0.9	1	0.019	10.txt	**2 sets**
	VC9 (9)	10	10	99	**9**	0.01	0.9	3	0.019	10.txt	**2 sets**
	VC10 (10)	10	10	99	**10**	0.01	1	3	0.019	10.txt	**2 sets**
Probabilidade de Mutação	**VC11 (-0.01)**	10	10	99	9	**-0,01**	0.9	3	0.02	10.txt	**Erro**
	VC11 (0)	10	10	99	9	**0**	0.9	3	0.019	10.txt	**2 sets**
	VC11 (0.009)	10	10	99	9	**0.009**	0.9	2	0.019	10.txt	**2 sets**
	VC12 (0.01)	10	10	99	9	**0.01**	1	2	0.019	10.txt	**2 sets**
	VC12 (1)	10	10	99	9	**1**	1	2	0.019	10.txt	**2 sets**
	VC12 (1.01)	10	10	99	9	**1.01**	1	2	0.019	10.txt	**Erro**
Probabilidade de Crossover	**VC13 (-0.01)**	10	10	99	9	0.009	**-0,01**	3	0.02	10.txt	**Erro**
	VC13 (0)	10	10	99	9	0.009	**0**	3	0.019	10.txt	**2 sets**
	VC13 (0.89)	10	10	99	9	0.009	**0.89**	2	0.019	10.txt	**2 sets**
	VC14 (0.9)	10	10	99	9	0.009	**0.9**	2	0.019	10.txt	**2 sets**
	VC14 (1)	10	10	99	9	0.009	**1**	2	0.019	10.txt	**2 sets**
	VC14 (1.01)	10	10	99	9	0.009	**1.01**	2	0.019	10.txt	**Erro**
Tamanho do Torneio	**VC15 (0)**	10	10	99	9	0.009	0.9	**0**	0.019	10.txt	Erro
	VC15 (1)	10	10	99	9	0.009	0.9	**1**	0.019	10.txt	**2 sets**
	VC15 (2)	10	10	99	9	0.009	0.9	**2**	0.019	10.txt	**2 sets**
	VC16 (3)	10	10	99	9	0.009	0.9	**3**	0.019	10.txt	**2 sets**
Elitismo	**VC17 (-0.01)**	10	10	99	9	0.009	0.9	3	**-0,01**	10.txt	**Erro**
	VC17 (0)	10	10	99	9	0.009	0.9	3	**0**	10.txt	**2 sets**
	VC17 (0.019)	10	10	99	9	0.009	0.9	3	**0.019**	10.txt	**2 sets**
	VC18 (0.02)	10	10	99	9	0.009	0.9	3	**0.02**	10.txt	**2 sets**
	VC18 (1)	10	10	99	9	0.009	0.9	3	**1**	10.txt	**2 sets**
	VC18 (1.01)	10	10	99	9	0.009	0.9	3	**1.01**	10.txt	**Erro**

Tabela 9: Testes com as classes válidas – Parte 1 (outcomes esperados a negrito = falhou o teste)

	Teste / Input	Outcome Esperado (número mínimo de sets)	Outcome para inputs inválidos (Seed 500)	Outcome 1 (Seed 7575)	Outcome 2 (Seed 158)	Outcome 3 (Seed 4209)	Outcome 4 (Seed 4610)	Outcome 5 (Seed 8967)	Outcome 6 (Seed 5668)	Outcome 7 (Seed 938)	Outcome 8 (Seed 6559)	Outcome 9 (Seed 4849)	Outcome 10 (Seed 6811)
Número de Sets	VC1 (-1)	Erro	Sets Vazio										
	VC1 (0)	Sets vazio		Sets Vazio	Sets Vazio	Sets Vazio	Sets Vazio	Sets Vazio	Sets Vazio	Sets Vazio	Sets Vazio	Sets Vazio	Sets Vazio
	VC1 (4)	3 sets		2 Sets	2 Sets	3 Sets	3 Sets	2 Sets	2 Sets	2 Sets	2 Sets	2 Sets	2 Sets
	VC2 (5)	2 sets		3 Sets	2 Sets	2 Sets *	2 Sets *	2 Sets	2 Sets	2 Sets *	2 Sets *	2 Sets	3 Sets
	VC2 (9)	2 sets		3 Sets	3 Sets	3 Sets	3 Sets	3 Sets	2 Sets	3 Sets	3 Sets	3 Sets	3 Sets
	VC3 (10)	2 sets		2 Sets	3 Sets	3 Sets	3 Sets	2 Sets	4 Sets	3 Sets	3 Sets	3 Sets	3 Sets
Tamanho do Universe	VC4 (-1)	Erro	IndexError: list index out of range										
	VC4 (0)	Sets Vazio		Sets Vazio	Sets Vazio	Sets Vazio	Sets Vazio	Sets Vazio	Sets Vazio	Sets Vazio	Sets Vazio	Sets Vazio	Sets Vazio
	VC4 (9)	2 sets		3 Sets	2 Sets	2 Sets *	2 Sets *	2 Sets	2 Sets	2 Sets *	2 Sets *	2 Sets *	3 Sets
	VC5 (10)	2 sets		3 Sets	3 Sets	3 Sets	3 Sets	3 Sets	4 Sets	2 Sets	3 Sets	2 Sets	3 Sets
	VC5 (49)	3 Sets		2 Sets	2 Sets	2 Sets	2 Sets	2 Sets	2 Sets	2 Sets	2 Sets	2 Sets	2 Sets
	VC6 (50)	3 sets		3 sets	3 sets	3 sets	3 sets	3 sets	3 sets	3 sets	3 sets	3 sets	2 Sets
Número de Gerações	VC7 (-1)	Erro	5 Sets										
	VC7 (0)	2 sets		4 Sets	3 Sets	4 Sets	4 Sets	4 Sets	3 Sets	4 Sets	2 Sets *	3 Sets	3 Sets
	VC7 (99)	2 sets		3 Sets	3 Sets	3 Sets	3 Sets	4 Sets	2 Sets	3 Sets	3 Sets	2 Sets	3 Sets
	VC8 (100)	2 sets		3 Sets	3 Sets	3 Sets	3 Sets	3 Sets	4 Sets	2 Sets	3 Sets	2 Sets	3 Sets
Tamanho da População	VC9 (1)	Erro	ValueError: Sample larger than population or is negative										
	VC9 (2)	2 sets		4 Sets	5 Sets	5 Sets	5 Sets	4 Sets	4 Sets	5 Sets	7 Sets	4 Sets	3 Sets
	VC9 (9)	2 sets		2 Sets	3 Sets	3 Sets	4 Sets	3 Sets	4 Sets	2 Sets	3 Sets	2 Sets	3 Sets
	VC10 (10)	2 sets		2 Sets	3 Sets	3 Sets	3 Sets	3 Sets	4 Sets	3 Sets	3 Sets	3 Sets	3 Sets
Probabilidade de Mutação	VC11 (-0.01)	Erro	5 Sets										
	VC11 (0)	2 sets		4 Sets	2 Sets *	3 Sets	3 Sets	3 Sets	3 Sets	4 Sets	3 Sets	3 Sets	4 Sets
	VC11 (0.009)	2 sets		2 Sets	3 Sets	3 Sets	3 Sets	3 Sets	4 Sets	3 Sets	3 Sets	3 Sets	3 Sets
	VC12 (0.01)	2 sets		2 Sets	3 Sets	3 Sets	4 Sets	2 Sets	4 Sets	3 Sets	3 Sets	3 Sets	3 Sets
	VC12 (1)	2 sets		5 Sets	6 Sets	5 Sets	7 Sets	5 Sets	5 Sets	5 Sets	6 Sets	4 Sets	6 Sets
	VC12 (1.01)	Erro	5 Sets										
Probabilidade de Crossover	VC13 (-0.01)	Erro	3 Sets										
	VC13 (0)	2 sets		3 Sets	3 Sets	3 Sets	3 Sets	2 Sets	3 Sets	3 Sets	2 Sets	2 Sets	2 Sets
	VC13 (0.89)	2 sets		2 Sets	3 Sets	3 Sets	3 Sets	3 Sets	3 Sets	3 Sets	3 Sets	3 Sets	3 Sets
	VC14 (0.9)	2 sets		2 Sets	3 Sets	3 Sets	3 Sets	3 Sets	4 Sets	3 Sets	3 Sets	3 Sets	3 Sets
	VC14 (1)	2 sets		2 Sets	3 Sets	3 Sets	3 Sets	2 Sets	4 Sets	3 Sets	3 Sets	3 Sets	3 Sets
	VC14 (1.01)	Erro	3 Sets										
Tamanho do Torneio	VC15 (0)	Erro	IndexError: list index out of range										
	VC15 (1)	2 sets		6 Sets	3 Sets	5 Sets	5 Sets	3 Sets	6 Sets	5 Sets	5 Sets	7 Sets	2 Sets *
	VC15 (2)	2 sets		2 Sets	3 Sets	3 Sets	3 Sets	3 Sets	4 Sets	3 Sets	3 Sets	3 Sets	3 Sets
	VC16 (3)	2 sets		2 Sets	3 Sets	3 Sets	4 Sets	3 Sets	3 Sets	2 Sets	3 Sets	2 Sets	3 Sets
Elitismo	VC17 (-0.01)	Erro	3 Sets										
	VC17 (0)	2 sets		2 Sets	3 Sets	3 Sets	4 Sets	3 Sets	3 Sets	2 Sets	3 Sets	2 Sets	3 Sets
	VC17 (0.019)	2 sets		2 Sets	3 Sets	3 Sets	4 Sets	3 Sets	3 Sets	2 Sets	3 Sets	2 Sets	3 Sets
	VC18 (0.02)	2 sets		2 Sets	3 Sets	3 Sets	4 Sets	3 Sets	3 Sets	2 Sets	3 Sets	2 Sets	3 Sets
	VC18 (1)	2 sets		4 Sets	3 Sets	4 Sets	4 Sets	4 Sets	3 Sets	4 Sets	2 Sets *	3 Sets	3 Sets
	VC18 (1.01)	Erro	5 Sets										
			* -> Output incorreto, pois não reconstrói o universo										

Tabela 10: Testes com as classes válidas – Parte 2 (NOTA: recomenda-se o uso de zoom a 200%)

11 – Anexos

11.1 – Anexos 1 – Control Flow (Todo o Processo)

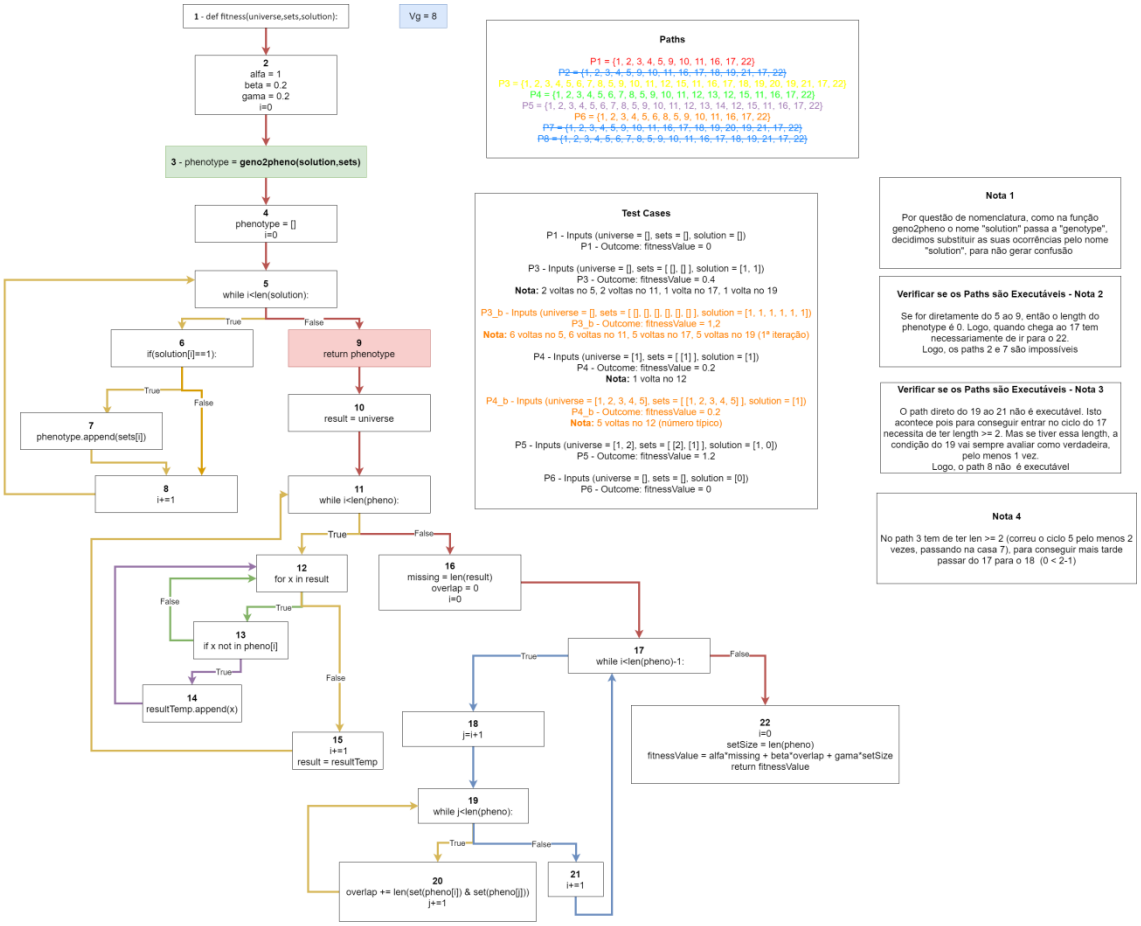

Figure 3: Processo de Control Flow para a função fitness

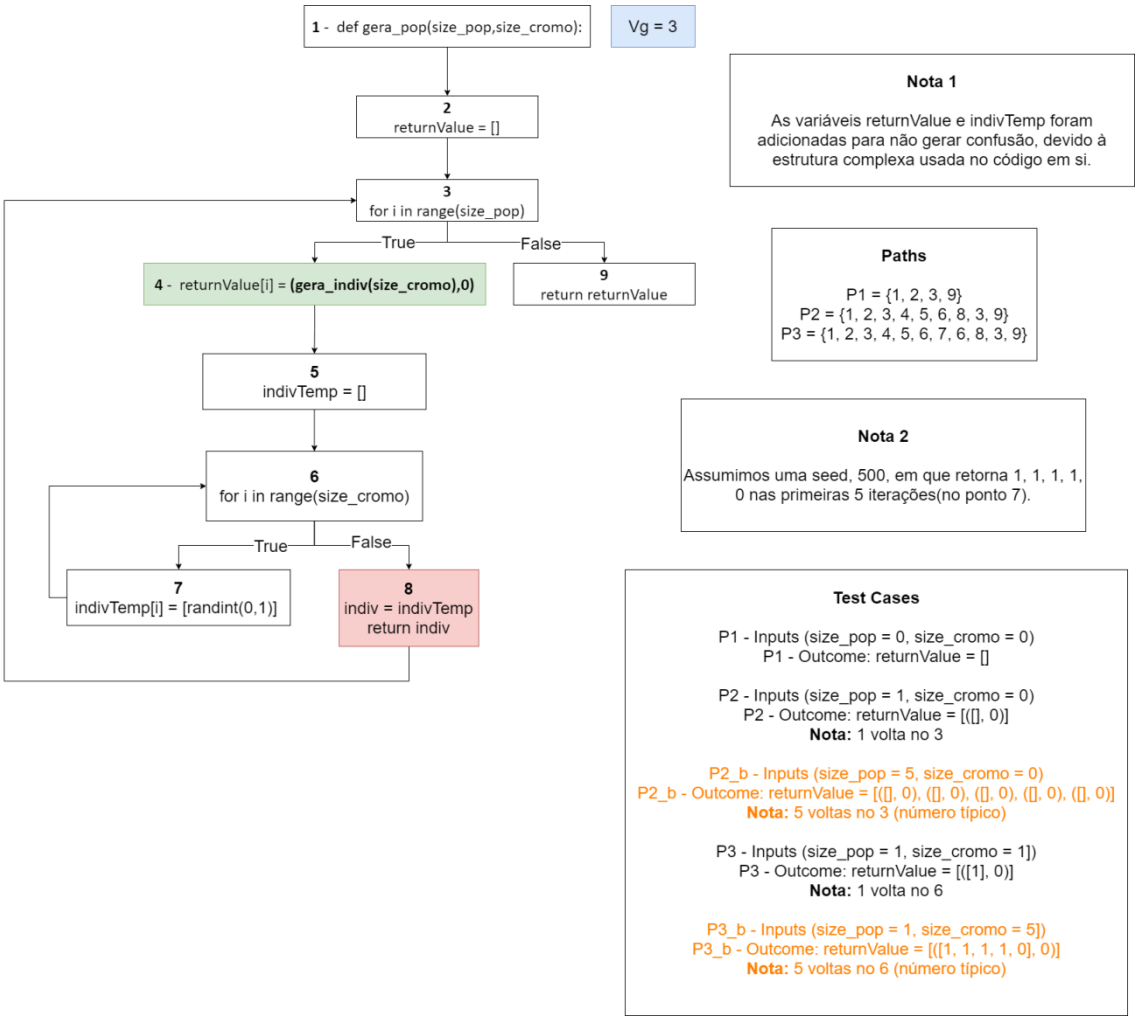

Figure 4: Processo de Control Flow para a função gera_pop

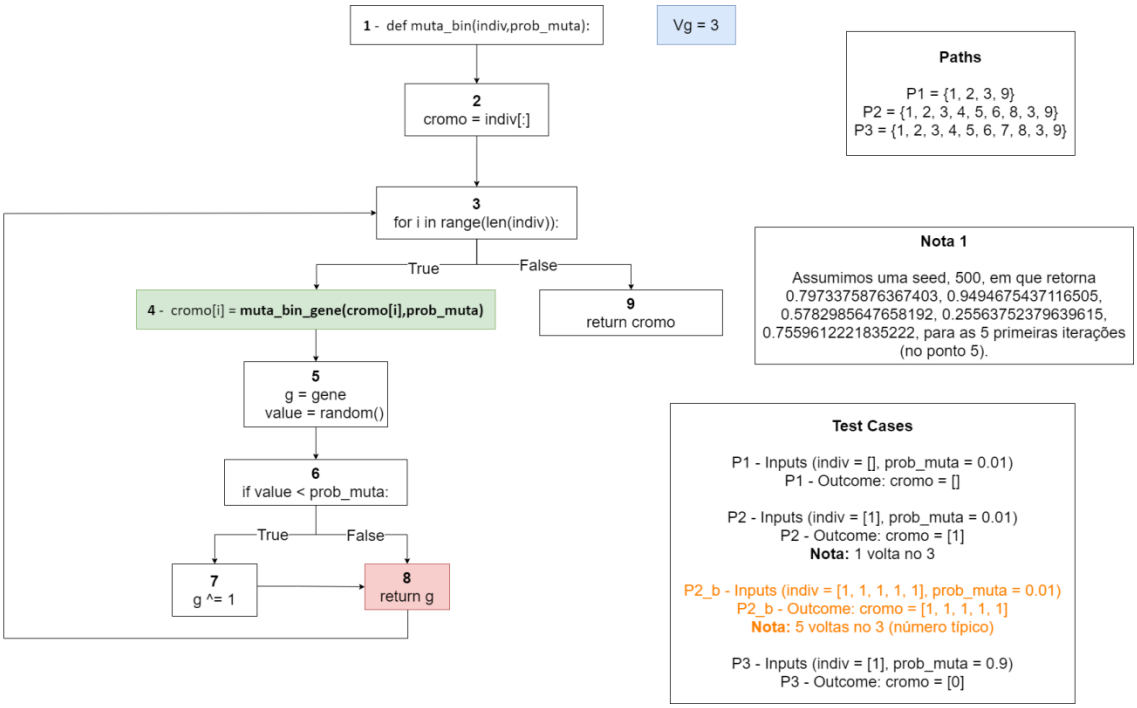

Figure 5: Processo de Control Flow para a função muta_bin

11.2.a. – Anexos 2 – Data Flow (Data Flow Graph)

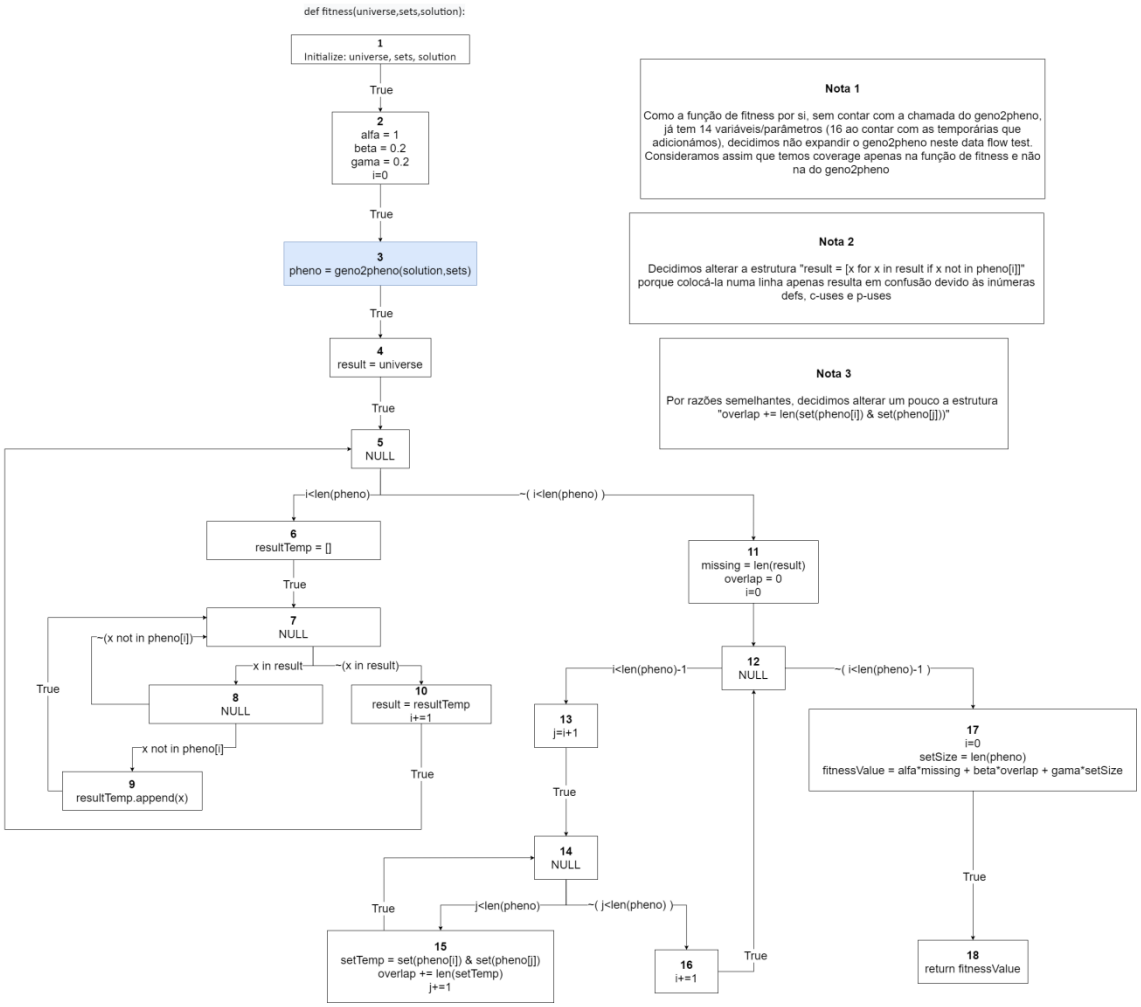

Figure 6: Data Flow Graph para a função fitness

11.2.b. – Anexos 2 – Data Flow (DU paths, caminhos completos, test cases)

fitness_du_paths.pptx

11.3.a. – Anexos 3 – Black Box (Test Cases para classes inválidas)

	Teste / Input	Número de Sets	Tamanho do Universe	Número de Gerações	Tamanho da População	Probabilidade de Mutação	Probabilidade de Crossover	Tamanho do Torneio	Elitismo	Nome do Ficheiro	Outcome Esperado
Número de Sets	**IC1**	**-1**	10	100	10	0.01	0.9	3	0.02	inv1.txt	**Erro**
	IC2	**word**	10	100	10	0.01	0.9	3	0.02	inv2.txt	**Erro**
	IC3	**Null**	10	100	10	0.01	0.9	3	0.02	inv3.txt	Erro
Tamanho do Universe	**IC4**	5	**-1**	100	10	0.01	0.9	3	0.02	inv4.txt	**Erro**
	IC5	5	**word**	100	10	0.01	0.9	3	0.02	inv5.txt	Erro
	IC6	5	**NULL**	100	10	0.01	0.9	3	0.02	inv6.txt	Erro
Número de Gerações	**IC7**	5	10	**-1**	10	0.01	0.9	3	0.02	inv7.txt	**Erro**
	IC8	5	10	**word**	10	0.01	0.9	3	0.02	inv7.txt	Erro
Tamanho da População	**IC9**	5	10	100	**1**	0.01	0.9	3	0.02	inv7.txt	Erro
	IC10	5	10	100	**word**	0.01	0.9	3	0.02	inv7.txt	Erro
Probabilidade de Mutação	**IC11**	5	10	100	10	**-0,01**	0.9	3	0.02	inv7.txt	**Erro**
	IC12	5	10	100	10	**2**	0.9	3	0.02	inv7.txt	**Erro**
	IC13	5	10	100	10	**word**	0.9	3	0.02	inv7.txt	Erro
Probabilidade de Crossover	**IC14**	5	10	100	10	0.01	**-0,01**	3	0.02	inv7.txt	**Erro**
	IC15	5	10	100	10	0.01	**2**	3	0.02	inv7.txt	**Erro**
	IC16	5	10	100	10	0.01	**word**	3	0.02	inv7.txt	Erro
Tamanho do Torneio	**IC17**	5	10	100	10	0.01	0.9	**0**	0.02	inv7.txt	Erro
	IC18	5	10	100	10	0.01	0.9	**word**	0.02	inv7.txt	Erro
Elitismo	**IC19**	5	10	100	10	0.01	0.9	3	**-0,01**	inv7.txt	**Erro**
	IC20	5	10	100	10	0.01	0.9	3	**2**	inv7.txt	**Erro**
	IC21	5	10	100	10	0.01	0.9	3	**word**	inv7.txt	Erro

Tabela 11: Test Cases para classes inválidas

11.3.b. – Anexos 3 – Black Box (Test Cases para classes válidas)

	Teste / Input	Número de Sets	Tamanho do Universe	Número de Gerações	Tamanho da População	Probabilidade de Mutação	Probabilidade de Crossover	Tamanho do Torneio	Elitismo	Nome do Ficheiro	Outcome Esperado (número mínimo de sets)
Número de Sets	**VC1 (-1)**	**-1**	10	100	10	0.01	0.9	3	0.02	1.txt	**Erro**
	VC1 (0)	**0**	10	100	10	0.01	0.9	3	0.019	2.txt	Sets vazio
	VC1 (4)	**4**	10	100	10	0.01	0.9	2	0.019	3.txt	**3 sets**
	VC2 (5)	**5**	10	100	10	0.01	1	2	0.019	4.txt	**2 sets**
	VC2 (9)	**9**	10	100	10	0.009	1	2	0.019	5.txt	**2 sets**
	VC3 (10)	**10**	10	100	9	0.009	1	2	0.019	6.txt	**2 sets**
Tamanho do Universe	**VC4 (-1)**	5	**-1**	100	10	0.01	0.9	3	0.02	7.txt	Erro
	VC4 (0)	5	**0**	100	10	0.01	0.9	3	0.019	8.txt	Sets Vazio
	VC4 (9)	5	**9**	100	10	0.01	0.9	2	0.019	9.txt	**2 sets**
	VC5 (10)	10	**10**	100	10	0.01	1	2	0.019	10.txt	**2 sets**
	VC5 (49)	10	**49**	100	10	0.009	1	2	0.019	11.txt	**3 Sets**
	VC6 (50)	10	**50**	100	9	0.009	1	2	0.019	12.txt	3 sets
Número de Gerações	**VC7 (-1)**	10	10	**-1**	10	0.01	0.9	3	0.02	10.txt	**Erro**
	VC7 (0)	10	10	**0**	10	0.01	0.9	3	0.019	10.txt	**2 sets**
	VC7 (99)	10	10	**99**	10	0.01	0.9	2	0.019	10.txt	**2 sets**
	VC8 (100)	10	10	**100**	10	0.01	1	2	0.019	10.txt	**2 sets**
Tamanho da População	**VC9 (1)**	10	10	99	**1**	0.01	0.9	3	0.02	10.txt	Erro
	VC9 (2)	10	10	99	**2**	0.01	0.9	1	0.019	10.txt	**2 sets**
	VC9 (9)	10	10	99	**9**	0.01	0.9	3	0.019	10.txt	**2 sets**
	VC10 (10)	10	10	99	**10**	0.01	1	3	0.019	10.txt	**2 sets**
Probabilidade de Mutação	**VC11 (-0.01)**	10	10	99	9	**-0,01**	0.9	3	0.02	10.txt	**Erro**
	VC11 (0)	10	10	99	9	**0**	0.9	3	0.019	10.txt	**2 sets**
	VC11 (0.009)	10	10	99	9	**0.009**	0.9	2	0.019	10.txt	**2 sets**
	VC12 (0.01)	10	10	99	9	**0.01**	1	2	0.019	10.txt	**2 sets**
	VC12 (1)	10	10	99	9	**1**	1	2	0.019	10.txt	**2 sets**
	VC12 (1.01)	10	10	99	9	**1.01**	1	2	0.019	10.txt	**Erro**
Probabilidade de Crossover	**VC13 (-0.01)**	10	10	99	9	0.009	**-0,01**	3	0.02	10.txt	**Erro**
	VC13 (0)	10	10	99	9	0.009	**0**	3	0.019	10.txt	**2 sets**
	VC13 (0.89)	10	10	99	9	0.009	**0.89**	2	0.019	10.txt	**2 sets**
	VC14 (0.9)	10	10	99	9	0.009	**0.9**	2	0.019	10.txt	**2 sets**
	VC14 (1)	10	10	99	9	0.009	**1**	2	0.019	10.txt	**2 sets**
	VC14 (1.01)	10	10	99	9	0.009	**1.01**	2	0.019	10.txt	**Erro**
Tamanho do Torneio	**VC15 (0)**	10	10	99	9	0.009	0.9	**0**	0.019	10.txt	Erro
	VC15 (1)	10	10	99	9	0.009	0.9	**1**	0.019	10.txt	**2 sets**
	VC15 (2)	10	10	99	9	0.009	0.9	**2**	0.019	10.txt	**2 sets**
	VC16 (3)	10	10	99	9	0.009	0.9	**3**	0.019	10.txt	**2 sets**
Elitismo	**VC17 (-0.01)**	10	10	99	9	0.009	0.9	3	**-0,01**	10.txt	**Erro**
	VC17 (0)	10	10	99	9	0.009	0.9	3	**0**	10.txt	**2 sets**
	VC17 (0.019)	10	10	99	9	0.009	0.9	3	**0.019**	10.txt	**2 sets**
	VC18 (0.02)	10	10	99	9	0.009	0.9	3	**0.02**	10.txt	**2 sets**
	VC18 (1)	10	10	99	9	0.009	0.9	3	**1**	10.txt	**2 sets**
	VC18 (1.01)	10	10	99	9	0.009	0.9	3	**1.01**	10.txt	**Erro**

Tabela 12: Test Cases para classes válidas (Value Boundary Analysis)