NOVA IMS Information Management School

MGI

Mestrado em Gestão de Informação

Master Program in Information Management

Implementação de um Modelo de *Geometric Semantic Genetic Programming* para Aplicação Naval

Pedro Eduardo Quadrado da Fonseca

Trabalho de Projeto apresentado como requisito parcial para obtenção do grau de Mestre em Gestão de Informação

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

IMPLEMENTAÇÃO DE UM MODELO DE *GEOMETRIC SEMANTIC GENETIC PROGRAMMING* PARA APLICAÇÃO NAVAL

por

Pedro Eduardo Quadrado da Fonseca

Trabalho de Projeto apresentado como requisito parcial para a obtenção do grau de Mestre em Gestão de Informação, Especialização em Sistemas e Tecnologias de Informação

Orientador: Dr. Leonardo Vanneschi

11 2015

“Building the future and keeping the past alive are one and the same thing.”

AGRADECIMENTOS

Agradeço antes de mais ao meu professor e orientador Dr. Leonardo Vanneschi, sem o qual este trabalho nunca teria sido possível e graças ao qual descobri a fundo o mundo da Inteligência Artificial. Agradeço todas as aulas dadas, todas as sessões de esclarecimento, todos os conhecimentos passados, não só durante a tese mas também durante todas as aulas anteriores à mesma. Agradeço todo o apoio e interesse que teve neste trabalho que me confiou, e acima de tudo agradeço toda a amizade que sempre me mostrou.

Agradeço do mesmo modo ao Dr. Mauro Castelli, pelo acompanhamento que teve do projeto e pelo apoio a mim dado a todos os passos deste caminho, pela ajuda prestada quando deparado com dificuldades e por todo o companheirismo, dentro e fora do projeto. Agradeço também à Dr. Sara Silva, por toda a ajuda dada durante o trabalho, pelo interesse e apoio que igualmente demonstrou.

Agradeço aos meus pais, à minha avó, à minha tia e à minha irmã por me motivarem e apoiarem durante todo este processo, pelo incentivo e interesse que mostraram durante todos os momentos, por todas as horas que me ouviram a falar sobre algoritmos, pelo apoio que me deram quando o código funcionava e pela paciência que tiveram para me suportar quando não funcionava.

À Rute Correia, obrigado por estar a meu lado e me incentivar quando a motivação mais faltava. À Inês Macedo, obrigado pela paciência e apoio durante todo o processo.

Finalmente, a todos os que alguma vez me inspiraram e motivaram a querer fazer mais e melhor, a aspirar a mais e dar o todo de mim mesmo, presentes ou ausentes, o meu muito e sentido obrigado.

RESUMO

Recaí sob a responsabilidade da Marinha Portuguesa a gestão da Zona Económica Exclusiva de Portugal, assegurando a sua segurança da mesma face a atividades criminosas. Para auxiliar a tarefa, é utilizado o sistema Oversee, utilizado para monitorizar a posição de todas as embarcações presentes na área afeta, permitindo a rápida intervenção da Marinha Portuguesa quando e onde necessário. No entanto, o sistema necessita de transmissões periódicas constantes originadas nas embarcações para operar corretamente – casos as transmissões sejam interrompidas, deliberada ou acidentalmente, o sistema deixa de conseguir localizar embarcações, dificultando a intervenção da Marinha.

A fim de colmatar esta falha, é proposto adicionar ao sistema Oversee a capacidade de prever as posições futuras de uma embarcação com base no seu trajeto até à cessação das transmissões. Tendo em conta os grandes volumes de dados gerados pelo sistema (históricos de posições), a área de Inteligência Artificial apresenta uma possível solução para este problema. Atendendo às necessidades de resposta rápida do problema abordado, o algoritmo de Geometric Semantic Genetic Programming baseado em referências de Vanneschi et al. apresenta-se como uma possível solução, tendo já produzido bons resultados em problemas semelhantes.

O presente trabalho de tese pretende integrar o algoritmo de Geometric Semantic Genetic Programming desenvolvido com o sistema Oversee, a fim de lhe conceder capacidades preditivas. Adicionalmente, será realizado um processo de análise de desempenho a fim de determinar qual a ideal parametrização do algoritmo. Pretende-se com esta tese fornecer à Marinha Portuguesa uma ferramenta capaz de auxiliar o controlo da Zona Económica Exclusiva Portuguesa, permitindo a correta intervenção da Marinha em casos onde o atual sistema não conseguiria determinar a correta posição da embarcação em questão.

PALAVRAS-CHAVE

Programação Genética; Geometric Semantic Genetic Programming; Segurança Marítima; Previsão.

ABSTRACT

It falls under the Portuguese Navy's responsibility to manage the Portuguese Economic Exclusive Zone, ensuring its safety from criminal activities and accidents. To support this task, the Oversee system is used to monitor the position of all ships present in the area, allowing for the Navy's timely intervention where it's needed. However, the system requires periodic transmissions from the ships to operate correctly – if the transmissions stop, deliberately or by accident, the system can no longer track the ships, difficulting the Navy's intervention.

To face this limitation, it's proposed to add to the Oversee system the capability to predict future positions of a ship based on its path up to the interruption of transmissions. Considering the huge amounts of data generated by the system (position logs), a possible solution is found in Artificial Intelligence. Attending to the problem's needs of a fast response, the reference based Geometric Semantic Genetic Programming algorithm developed by Vanneschi at al. presents itself as a possible solution, already having produced good results in similar problems.

This project aims to integrate the Geometric Semantic Genetic Programming algorithm with the Oversee system, in order to grant it predictive capabilities. Furthermore, a performance analysis will be done in order to determine the ideal configuration of the algorithm. This project intends to supply the Portuguese Navy with a tool capable of aiding the management of the Portuguese Economic Exclusive Zone's management, allowing for the correct intervention of the Navy when the current system would not be able to track the intended ship.

KEYWORDS

Genetic Programming; Geometric Semantic Genetic Programming; Maritime Safety; Prediction.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

CE	Computação Evolutiva
AG	Algoritmo(s) Genético(s)
PG	Programação Genética
GSGP	*Geometric Semantic Genetic Programming*
GS-GP	Nova Implementação de GSGP
ZEE	Zona Económica Exclusiva
IC	Inteligência Computacional
AIS	*Automatic Identification System*
LS	*Linear Scaling*
RMSE	*Root Mean Square Error*

1. INTRODUÇÃO

1.1. *IDENTIFICAÇÃO DO PROBLEMA*

Atualmente, os mares são importantes vias de transporte de bens e pessoas, sendo um valioso recurso económico quando bem explorado, para além de um importante património natural. Muitos indivíduos, empresas e países dependem dos mesmos para a sua subsistência. No entanto, a exploração dos mares enfrenta vários problemas cada vez mais relevantes, apesar dos esforços existentes para garantir um Sistema de Transporte Marítimo (MTS) seguro.

Presentemente, o MTS é anualmente responsável por 140 mortes e €1,5 biliões de perdas anuais apenas na Europa, sendo 4 vezes mais perigoso que o Sistema de Transporte Aéreo (Trucco, 2008). Os acidentes mais frequentes (e com maior taxa de sinistralidade) são encalhamentos (32%), embates com objetos submersos (24%) e colisões (16%) (Trucco, 2008). O fator mais determinante para estes acidentes continua a ser o erro humano. Num estudo publicado pelo *Transportation Safety Board of Canada* ((TSB), 1998) é possível ver que 74% dos acidentes marítimos tem como origem o erro humano, contrastando com apenas 20% onde a origem se trata de um error técnico. Agravando o problema estão problemas como a pirataria, o contrabando, etc., atividades que causam prejuízos económicos e sociais nas zonas onde ocorrem, piorando o quadro de figuras acima descrito.

Outra face do problema lida não com o controlo do navio (descrito no parágrafo acima), mas sim com a monitorização de acidentes e incidentes por parte das autoridades competentes. Embora as figuras acima referidas possam ser combatidas com medidas preventivas, a resposta eficaz e eficiente às mesmas situações permitem também mitigar a gravidade das mesmas. A correta identificação de uma situação onde a intervenção é necessária (seja um acidente ou uma infração da lei) e a mobilização dos meios indicados para a localização correta são fatores que podem contribuir grandemente para a atenuação dos riscos existentes no MTS. No entanto, dada a enorme quantidade de informação que a identificação e resposta requerem, o risco de erro humano também está presente nesta vertente.

É neste contexto que a implementação de ferramentas de Inteligência Computacional (IC) podem auxiliar grandemente a tarefa de tornar a navegação mais segura – quer do lado dos navegadores, onde já existem várias propostas de soluções capazes de auxiliar diretamente a navegação (Bukhari, 2013;Yang, 2007;Zhao, 2014); quer do lado dos controladores, onde existem projetos que tornam a deteção de anomalias mais fácil, etc. (Kazemi, 2013;Laxhammer, 2008;Riveiro, 2008). Ferramentas de IC podem ter uma influência tão grande na segurança do MTS que IC é uma das partes centrais de um ambiente de *e-navigation* ((IMO), 2015). O Plano Estratégico de Implementação (SIP) de *e-navigation* da Organização Marítima Internacional (IMO) prevê desenvolver entre 2015 e 2019 um ambiente harmonizado que permita o desenvolvimento de produtos e serviços universais que permitam aumentar a segurança de navegação e reduzir a ocorrência de erros.

O presente trabalho de tese insere-se no âmbito de melhorar a monitorização de navios. Por base tem um algoritmo de *Geometric Semantic Genetic Programming* (GS-GP) desenvolvido por Vanneschi et al. (Vanneschi, 2015), que visa prever a posição futura de navios com base no histórico existente de dados provenientes do *Automatic Identification System* (AIS). Embora o AIS permita monitorizar a

posição atual de um navio, bem como uma previsão de poucos minutos no futuro, a presente tese pretende permitir que essa previsão seja feita para várias horas no futuro. Se bem sucedido, este algoritmo permitirá responder a situações onde a posição de um navio não é conhecida devido a uma desativação do seu AIS (quer acidental ou intencionalmente) – casos onde a atuação da Marinha deve ser o mais rápida possível, pelo que saber a posição onde atuar é fulcral ao sucesso da operação.

Em Portugal, a administração do território marítimo nacional recai sobre a Marinha Portuguesa. Tendo em conta a dimensão da ZEE (Zona Económica Exclusiva), o processo requer o controlo e proteção de uma área de 3.877.408 quilómetros quadrados, incluindo patrulha, proteção civil e combate a atividades ilegais – atividades que requerem saber posições exatas de embarcações na zona abrangida, pelo que o AIS é utilizado. A monitorização da ZEE é realizada com o sistema Oversee, da Critical Software (ver http://www.oversee-solutions.com/ para a página do projeto). Visto que o sistema depende do AIS para determinar as posições dos navios, está sujeito aos problemas antes levantados sobre a desativação do AIS.

É o âmbito desta tese realizar a integração do algoritmo de GS-GP com o sistema Oversee, realizando também a configuração do mesmo de modo a permitir a sua correta e ótima operação. Realizada a integração, o sistema Oversee terá acesso a uma nova ferramenta de CI que permite a previsão das posições de embarcações para períodos de tempo de várias horas. O projeto é realizado em parceria com a Critical Software S.A., o Instituto Superior de Estatística e Gestão de Informação (ISEGI/UNL), o Instituto de Engenharia de Sistemas e Computadores, Investigação e Desenvolvimento em Lisboa (INESC ID/INESC/IST/UTL) e a Universidade de Coimbra (UC).

1.2. *OBJETIVO DO PROJETO*

1.2.1. Definição do Objetivo

A presente tese tem como objetivo principal implementar o algoritmo de GSGP desenvolvido por Castelli, Vanneschi e Silva, criando as condições para o mesmo conseguir interagir com as bases de dados da Critical Software e com o sistema Oversee. Após a realização da implementação, a tese tem também por objetivo o levantamento de possíveis problemas provenientes da utilização de dados reais, propondo e testando possíveis soluções, bem como a avaliação das soluções obtidas face aos critérios de avaliação definidos.

1.2.2. Objetivos Específicos

Para atingir o objetivo principal, o presente trabalho focar-se-á em atingir os seguintes objetivos específicos de modo sequencial:

1. Criação de um *software* capaz de preparar dados para serem analisados pelo algoritmo de GSGP, bem como ler e reformatar os resultados obtidos;

2. Integrar o *software* criado de modo a interagir com as bases de dados da Critical Software, permitindo a leitura, processamento e registo de dados no sistema Oversee;

3. Análise dos resultados produzidos pelo algoritmo de GSGP num ambiente *live*, identificando possíveis problemas não detetados;

4. Proposta de soluções a problemas encontrados, passando por testes e recomendações futuras;

5. Execução e avaliação de várias configurações de parâmetros cruciais do algoritmo de PG, a fim de identificar as que permitem melhor equilíbrio entre qualidade de solução e tempo de desempenho.

1.3. *RELEVÂNCIA DO PROJETO*

O presente trabalho de tese visa criar condições para que o algoritmo de GSGP desenvolvido possa ser testado num ambiente *live*, funcionando como uma prova de conceito adicional para fundamentar a integração completa no sistema de maneira a que a funcionalidade seja disponibilizada à Marinha Portuguesa – ao permitir que os resultados, até agora obtidos apenas em condições laboratoriais, sejam vistos no contexto onde serão finalmente utilizados, o sistema Oversee. Visa também identificar quais os possíveis problemas que derivam da transição do algoritmo do contexto laboratorial para o contexto real, identificando bases para desenvolvimentos futuros.

De um modo mais global, este estudo visa também estudar a nova configuração de GS-GP proposta em [4], aplicando-a numa área à qual ainda não foi usada anteriormente. Dado que a mesma é recente, é relevante estudar a sua aplicação em áreas diversas, definindo um grau geral de desempenho para vários problemas tipo (regressões). Contribui também para uma base teórico-prática de estudos que poderá vir a suportar a teoria em si, caso os resultados se mostrem positivos, levando no futuro a uma aplicação mais difundida pela comunidade científica – podendo contribuir para a área de PG como um todo, visto [4] apresentar resultados altamente favoráveis.

1.4. *DESCRIÇÃO DO DOCUMENTO*

Serve o presente capítulo como uma breve descrição da estrutura do documento apresentado. Contando com o presente capítulo de introdução, este documento é composto por 9 capítulos.

O Capítulo 2 contem uma introdução à Computação Evolutiva, com especial ênfase em Algoritmos Genéticos. Serve o Capítulo 2 como contextualização ao Capítulo 3, que diz respeito a apenas uma técnica de Computação Evolutiva, a Programação Genética, utilizada nesta tese – o Capítulo 3 apresenta um breve resumo dos desenvolvimentos realizados na área, com especial foco na Programação Genética com operadores semânticos, que levou ao modelo de Programação Genética utilizado nesta tese. Servindo os Capítulos 2 e 3 como contextualização às técnicas usadas no presente trabalho de tese, o Capítulo 4 apresenta o problema estudado e o contexto onde o mesmo se insere.

O Capítulo 5 apresenta a metodologia adotada para realizar o presente trabalho de tese. O Capítulos 6 apresenta um resumo do processo de implementação, sendo o Capítulo 7 a apresentação dos resultados obtidos em paralelo com a implementação (justificando as parametrizações realizadas no Capítulo 6).

O Capítulo 8 apresenta as considerações finais referentes à presente tese, sendo o Capítulo 9 referente à bibliografia utilizada.

2. INTRODUÇÃO À COMPUTAÇÃO EVOLUTIVA

A Computação Evolutiva (CE) é um subcampo de Inteligência Artificial que tem por base as teorias de evolução de Darwin. Segundo as mesmas, um organismo evolui com base em cinco elementos: reprodução (um organismo provém de progenitores); adaptabilidade (o organismo mais bem adaptado ao seu ambiente tem as melhores hipóteses de sobreviver e de se reproduzir); hereditariedade (os progenitores passam algumas das suas caraterísticas aos filhos); variação (os filhos são distintos dos progenitores, possivelmente mais bem adaptados); e competição (apenas os organismos mais aptos sobrevivem, tendo os menos aptos menores probabilidades de reprodução) (Darwin, 1859). A CE faz o paralelo com esta teoria ao usar conjuntos de soluções que são iteradas a fim de evoluírem de modo automático para melhores soluções ao problema proposto, sendo o processo mais detalhado no capítulo 2.1.

Historicamente, são identificados quatro principais algoritmos de CE: Estratégias de Evolução (Schwefel, 1975); Programação Evolutiva (Fogel, 1962); Algoritmos Genéticos (Holland, 1975); e Programação Genética (Koza, 1992). A presente tese foi realizado com recurso apenas a Programação Genética (PG), pelo que será o algoritmo mais focado. No entanto, a fim de melhor enquadrar a escolha de PG, o presente capítulo realizará uma breve introdução a Algoritmos Genéticos (AG), dada a sua semelhança com PG (e sendo as limitações de AGs a base do desenvolvimento da PG).

2.1. *ALGORITMOS GENÉTICOS*

AGs são um dos primeiros tipos de algoritmos evolutivos desenvolvidos, servindo de base a muitos dos algoritmos que os seguiram, como os de Koza (Koza, 1992) e de Moraglio (Moraglio, 2012), especialmente relevantes para o presente trabalho de tese.

Em AG, um indivíduo é representado como uma sequência de caracteres, pertencentes a um conjunto (geralmente discreto) de um alfabeto preciso, selecionado de consoante as necessidades do problema em questão. Cada um dos indivíduos representa uma solução candidata ao problema, sendo que alguns indivíduos serão melhores respostas ao problema (Holland, 1975). O output de um individuo é um número real que quantifica a capacidade que o mesmo individuo tem de resolver o problema apresentado, sendo denominado por *fitness* do indivíduo – este output é calculado através da função de fitness definida (Holland, 1975). A termo de exemplo, assumindo como individuo a sequência binária (1 0 1 1 0 0), pode ser proposta uma função de fitness $F(x, y) = y + x$, onde x corresponde ao valor binário dos 3 primeiros caracteres e y ao valor dos restantes. O conjunto de soluções composto por todos os indivíduos é denominado de população, sendo que todos os seus elementos possuem o mesmo tamanho e apenas os elementos do alfabeto definido (neste caso, binário), mas são criados aleatoriamente.

Com base nas classificações de *fitness* de cada individuo, estes são selecionados para uma população intermédia, tendo os indivíduos de maior *fitness* mais probabilidades de serem selecionados. Este processo, denominado seleção, é repetido até a população intermédia ter o mesmo número de indivíduos que a população original. Terminada a seleção, a população intermédia será submetida a reprodução, cruzamento ou mutação – escolhendo ou criando os indivíduos da nova população que irá substituir a população original (Holland, 1975).

Reprodução consiste simplesmente em copiar o individuo original para a nova população, deixando-o inalterado (Holland, 1975). Cruzamento consiste na escolha de dois indivíduos da população intermédia para serem combinados – ou seja, tendo por base os indivíduos originais (progenitores ou pais), serão criados dois novos indivíduos (filhos) diferentes um do outro e dos progenitores, sendo os filhos introduzidos na nova população e os pais descartados (Holland, 1975) (é de notar que o cruzamento implica um mínimo de dois indivíduos, mas podem ser considerados mais, sendo apenas necessário que é gerado um número equivalente de filhos). O método mais comum de cruzamento é o *one point crossover,* no qual é definido um ponto de corte nos dois pais e os filhos são a recombinação dos mesmos (a termo de exemplo, considerando (11111) e (00000), o cruzamento com corte após o segundo caracter resultaria em (11000) e (00111)). Mutação consiste na alteração de um único individuo, alterando um ou mais caracteres da sua sequência (Holland, 1975). O método mais comum de mutação é o *point mutation*, onde é escolhido um caracter aleatório do individuo e substituído por outro caracter aleatório válido.

As operações acima são repetidas um determinado número de iterações, ou até que seja atingida uma condição de fim (ex: atingir fitness 0 numa regressão). É de notar que, embora os valores da população não sejam fixos (as soluções mudam durante a execução do algoritmo), o universo definido é sempre o mesmo – ou seja, fatores como o tamanho da população, o tamanho de cada individuo, a função de *fitness*, número máximo de gerações, etc. são constantes (Holland, 1975).

2.2. *LIMITAÇÕES DE ALGORITMOS GENÉTICOS*

AGs já foram historicamente usados com sucesso numa diversidade de problemas, incluindo desenho de circuitos integrados, organização de redes, desenho de redes neuronais, etc. sendo a sua aplicação mais comum a otimização de funções com múltiplos parâmetros (Forrest, 1996). Embora os AGs tenham produzido boas soluções a estes problemas, os mesmos são também ilustrativos das suas limitações.

Nos problemas acima referidos, é possível dizer que o problema é bem conhecido – desse modo, é possível antecipar o formato da solução. Isto representa uma das caraterísticas dos problemas de AG: visto que todas as soluções têm o mesmo tamanho e seguem o mesmo formato, é necessário saber à partida qual o formato a aplicar. Por outras palavras, embora não se conheça a solução, conhece-se o seu formato, o que leva a que AGs só possam ser aplicadas a um certo tipo de problema (Forrest, 1993).

Na prática, grande parte dos problemas do quotidiano é demasiado complexa para permitir que se saiba antecipadamente qual a forma de possíveis soluções (que ou quantos elementos devem ser considerados, como os mesmos devem ser organizados, como se relacionam entre si, etc.). Muitos deles requerem respostas com estruturas hierárquicas, estruturas condicionais que recorrem a operadores e relações booleanos (AND, NOT, =, >, etc.) ou estruturas de repetição (Koza, 1992). A representação destes tipos de estruturas em indivíduos com comprimento fixo compostos unicamente por caracteres é muito pouco transparente, sendo por vezes quase impossível para problemas mais complexos (Koza, 1992). Do mesmo modo, AGs não possuem variabilidade dinâmica, pois o tamanho dos indivíduos é fixado no início da execução e nunca é alterado, o que limita os estados possíveis das soluções, bem como a aprendizagem do algoritmo como um todo (visto limitar as respostas que o algoritmo pode produzir devido ao tamanho estático das mesmas) (Koza, 1992).

3. PROGRAMAÇÃO GENÉTICA

Como visto no capítulo 2.2, os AGs dificilmente representam indivíduos que atuem como programas de computador (capazes de executar decisões, repetições e hierarquias), que constituem a maneira mais simples e natural de representar problemas do quotidiano. Do mesmo modo, o tamanho constante e necessidade de saber à partida quais os componentes das soluções limitam o tipo de problemas a que AGs são aplicados, bem como a possível aprendizagem nos restantes (Koza, 1992). É neste contexto que é desenvolvida a Programação Genética (PG).

Proposta por Koza em 1992 (Koza, 1992), a PG é uma extensão de AG que lida com as limitações existentes ao alterar a representação dos indivíduos da população, substituindo as sequências de caracteres por programas de computador, representados através de árvores de sintaxe. O resto do algoritmo é quase idêntico a AG tradicional, sendo certos procedimentos adaptados à nova representação de indivíduos. De modo geral, o algoritmo opera da mesma forma: é aleatoriamente criada uma nova população p com um tamanho h; é realizada uma seleção dos melhores indivíduos de p para uma população intermédia p'; os indivíduos em p' são submetidos a operadores de reprodução, cruzamento e mutação, sendo p substituída pelos novos indivíduos criados; e o processo descrito é iterado até ser encontrada uma solução satisfatória ou ser atingido um determinado número de iterações.

3.1. *REPRESENTAÇÃO DE INDIVÍDUOS*

A principal diferença entre AG e PG é a representação de indivíduos. Em PG, as soluções são programas de computador (face às sequências de caracteres de AG). Existem várias formas de representação de programas: cartesiana (Miller, 1999), linear (Banzhaf, 1993), etc. A representação utilizada nesta tese são árvores de sintaxe (Koza, 1992), pelo que será a representação apresentada. Na Figura 3.1, é possível ver a representação do programa *x*(x+y)*.

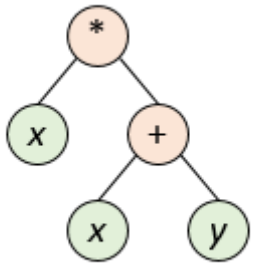


Figura 3.1 – Programa de computador representado como árvore de sintaxe

Como ilustrado pela figura 3.1, existem dois tipos de componentes numa árvore de sintaxe: os nós (na figura, os círculos vermelhos) e as folhas ou terminais (na figura, os círculos verdes). Todos os nós possíveis para os individuos de uma população estão contidios no conjunto de funções (F), e todos os valores terminais estão contidos no conjunto de terminais (T), sendo as árvores de sintaxe construídas através da utilização dos dois. Deste modo, todos os programas que podem ser contruídos com as funções em F e com os terminais em T constituem o espaço de procura de

soluções para determinado problema – ou seja, ao definir T e F, é limitado o modo como os programas podem evoluir (Poli, 2008).

O conjunto de terminais T = {α1, α2, …, αn} define os argumentos utilizados pelo programa (Koza, 1992). T pode incluir constantes (pré-definidas ou geradas aleatoriamente durante a execução), funções de aridade zero (funções que sem argumentos, como *rand*()), ou inputs externos do programa (variáveis) (Koza, 1992). Na Figura 3.1, podemos definir T = {*x*, *y*}.

O conjunto de funções F = {f_1, f_2, …, f_n} define quais as funções que o programa pode executar sobre os terminais (Koza, 1992). Estas podem incluir funções matemáticas e aritméticas (+, *, *sin*, *cos*, etc.), booleanas (AND, OR, NOT), condicionais (IF, THEN, ELSE), de repetição (FOR, WHILE, etc.), etc., podendo também ser utilizadas funções desenvolvidas especificamente para o problema em questão. Para garantir que as funções em F funcionam como pretendido, os mesmos devem possuir uma propriedade denominado como *closure* (Koza, 1992), que pode ser subdividida em duas subpropriedades (Poli, 2008):

- Consistência de tipo – graças ao cruzamento, as árvores podem ser criadas aleatoriamente. Isto pode levar a combinações inválidas (exemplo, a soma de valores booleanos). Para evitar estas situações, todas as funções em F devem aceitar e devolver argumentos do mesmo tipo. Dadas as limitações implícitas a esta regra, é possível implementar conversores que tratem dados como o tipo esperado (por exemplo, ler todos os números positivos como *true* e negativos como *false* permite que funções booleanas aceitem argumentos numéricos), o que, em troca de maio flexibilidade evolutiva, pode influenciar parcialmente a avaliação da solução.

- Segurança de avaliação – dada a natureza aleatória da criação de programas, é possível que sejam implementadas operações que falhem quando executadas, como divisões por zero. Para evitar falhas durante a execução, a maioria dos métodos é implementada de modo protegido, que retornam valores pré-determinados em caso de erro (no caso das divisões por zero, é utilizada a divisão protegida, representada por %, que retorna 1 para divisões por zero). Alternativamente, é possível identificar indivíduos com operações inválidas e penalizar o seu *fitness*, reduzindo a probabilidade dos mesmos se reproduzirem.

Adicionalmente, tanto F como T devem ser conjuntos *suficientes* (Koza, 1992). Um conjunto considera-se suficiente quanto os seus elementos permitem representar a solução ao problema em questão. No entanto, apenas é possível garantir suficiência quando é possível prever qual o resultado do problema será (Poli, 2008). Por exemplo, considerando T = {*x*, *y*}, o conjunto de funções F = {AND, OR, NOT} é suficiente, pois permite representar todas funções booleanas a que *x* e *y* podem ser sujeitos. O conjunto F = {+, -, *, /} não é considerado suficiente, pois não permite representar conceitos como uma função exponencial (pode representar aproximações, que no contexto do problema poderão ser boas o suficiente).

3.2. *INICIALIZAÇÃO DA POPULAÇÃO*

O primeiro passo do algoritmo é criar os indivíduos que constituem a população, que irão constituir a base da evolução. Tipicamente aleatória, podem ser distinguidos 3 métodos como os mais comuns de inicialização: *grow*; *full*; *ramped half-and-half (Koza, 1992)*. Qualquer árvore criada terá uma dada

profundidade, representada por *d*. É de notar que a raiz de uma árvore tem profundidade zero (tomando isso como exemplo, a árvore representada na Figura 3.1 terá uma profundidade máxima de 2).

- *Grow* (ou livre)
    - Para a raiz da árvore (o primeiro nó) é escolhido aleatoriamente (com igual probabilidade) um elemento do conjunto F;
    - Sendo a aridade da função escolhida *n*, são escolhidos aleatoriamente *n* elementos do conjunto F e do conjunto T como nós da função acima;
    - O passo 2 é repetido para todos os nós criados que sejam uma função (caso um nó seja proveniente do conjunto T, esse ramo não irá continuar a ser expandido). Este processo é repetido até ser atingida a profundidade *d-1*, ou até todos os ramos terminarem em elementos terminais;
    - Caso seja atingida a profundidade *d-1*, a iteração seguinte para profundidade *d* irá apenas escolher aleatoriamente do conjunto T, garantindo que nenhum ramo termina numa função.

  O método *grow* produz uma população irregular, visto que os indivíduos não terão necessariamente a mesma profundidade máxima (nunca ultrapassando d).

- *Full* (ou completa)
    - Todos os nós desde profundidade zero (raiz) até profundidade *d-1* são escolhidos aleatoriamente do conjunto F, garantindo que a árvore é composta apenas por funções até profundidade *d-1*;
    - Quando for atingida a profundidade *d*, os nós seguintes são escolhidos apenas do conjunto T.

  A inicialização *full* gera uma população muito semelhante entre si, visto todos os indivíduos possuírem a mesma profundidade máxima.

- *Ramped half-and-half* (mista em escada)
    - A população é dividida em d partes iguais, nas quais a profundidade máxima é definida, sequencialmente, como 1, …, d-1, d (por exemplo, sendo d=3, serão criados três segmentos onde a população terá como profundidade máxima, respetivamente, 1, 2 e 3);
    - Cada subsegmento é dividido ao meio, sendo metade dos indivíduos criado através de inicialização *full* e os restantes através de inicialização *grow*.

  A inicialização *ramped half-and-half* foi proposta por Koza (Koza, 1992), que observou que as populações criadas pelos outros métodos criavam populações compostas por indivíduos muito semelhantes entre si. Dada a importância da diversidade da população para a

aprendizagem do algoritmo, este método é utilizado para criar populações com alta diversidade.

3.3. *FITNESS*

Na natureza, o *fitness* de um individuo é a sua capacidade de sobreviver e de se reproduzir (Darwin, 1859). Do mesmo modo, em PG o *fitness* de um individuo indica a qualidade da solução que ele representa e é um indicador da sua probabilidade de ser selecionado durante a fase de seleção (Koza, 1992). O *fitness* de um individuo é calculada através da função de fitness.

Visto que em PG os indivíduos são programas, o seu nível de *fitness* provém dos resultados que os indivíduos produzem ao serem executados para cada caso de fitness existente. Um caso de *fitness* é um conjunto de variáveis independentes (os inputs do programa) e de uma variável dependente ou target (que representa o valor esperado após a execução). Geralmente, quanto mais casos onde o programa consiga chegar (ou aproximar-se) do valor esperado, melhor o *fitness* do individuo (Koza, 1992). Existem no entanto várias maneiras de calcular o grau de *fitness* de um individuo.

O *fitness* bruto (ou *raw*) é uma medida expressada na terminologia natural do programa (Koza, 1992). Como tal, o *fitness* ideal pode ser o maior valor possível ou o menor valor possível. Cada individuo possui um grau de *fitness* que é representativo do seu desempenho em todos os casos de fitness, calculado através da função de fitness definida.

Frequentemente, o objetivo de uma evolução é conseguir que o programa resultante consiga, para o maior número de casos de *fitness*, estar o mais perto possível da solução prevista, significando que pode então ser usado com novos inputs com a garantia de que o resultado produzido será provavelmente correto.

Exemplificando, considerem-se a expressão expressa na Figura 3.1 (*x*(x+y)*). O objetivo do programa é, para dado *input*, produzir um output correto, sendo que para evoluir o modelo são usados os seguintes casos de fitness onde o output é conhecido (assumindo *x* e *y* como inputs, *t* como o target):

x	y	t
1	2	3
2	2	7
1	0	5

Tabela 3.1 – Casos de Fitness

Dado que é pretendido que o programa se aproxime o máximo possível do valor target para cada caso, estamos perante um problema de regressão. Executando o programa, são obtidos os seguintes resultados:

Output 1	Output 2	Output 3
3	8	1

Tabela 3.2 – Resultados do programa aos casos de *fitness*

No entanto, é necessário dar um grau de *fitness* ao programa como um todo, não caso a caso. Tendo em conta que o problema é uma regressão, um bom indicador do *fitness* do individuo seria o erro que o mesmo possui para todos os casos. Desse modo, uma boa função de *fitness* para o problema, dada a relevância para o presente trabalho de tese, será o *Root Mean Square Error* (RMSE):

$$RMSE = \sqrt{\frac{1}{nº\ casos\ de\ fitness} \sum_{i=1}^{nº\ casos\ de\ fitness} (output_i - target_i)^2}$$

Voltando ao exemplo, se considerarmos o RMSE como função de fitness, então o programa *x*(x+y)* tem um fitness de (a):

Output	**Target**	**(output – target)²**
3	3	0
8	7	1
1	5	16
	MSE	17/3≈5.7
	RMSE	√5.7≈2.4

Tabela 3.3 – Calculo do *fitness* (RMSE) do programa

Podemos então considerar o valor de RMSE como o *fitness* bruto do programa. Tendo em conta tratar-se de uma regressão, é necessário definir que quanto menor for, melhor será o *fitness* do individuo.

Fitness padronizado (*standardized fitness*) converte o fitness bruto de maneira a que um valor menor é sempre melhor (não tendo de ser definido consoante o problema) (Koza, 1992). Isto faz com que o melhor valor de *fitness* padronizado seja zero. No caso de um problema onde o melhor valor de *fitness* possível é sempre o menor valor (minimização), então o fitness padronizado é igual ao fitness bruto. No caso de um problema onde o melhor valor de *fitness* é i maior valor possível (maximização), então o *fitness* padronizado é obtido pela subtração do maior valor possível de *fitness* pelo valor de *fitness* bruto – deste modo, mantem-se que o melhor valor será zero.

A vantagem deste método é padronizar todos os problemas de modo a que procura-se consistentemente o menor valor de *fitness* possível. Existem métodos adicionais para refinar o cálculo do *fitness* de um individuo a partir do *fitness* padronizado, tal como o *fitness* ajustado ou o *fitness* normalizado, mas dada a sua baixa importância teórica ou prática para a presente tese, o leitor interessado é remetido para (Koza, 1992).

3.4. *SELEÇÃO*

Através do *fitness* de um individuo, é possível distinguir quais os melhores indivíduos de uma população. Fazendo o paralelismo com as teorias de evolução de Darwin, quanto melhor adaptado um individuo estiver ao meio (em PG, quanto melhor responder ao problema), maior a sua probabilidade de sobreviver e de se reproduzir, enquanto indivíduos menos adaptados têm menor probabilidade de se propagarem para populações vindouras. Em PG, isto traduz-se na fase de seleção, onde os indivíduos da população original *p* são selecionados com base no seu grau de *fitness* a fim de determinar quais formaram a população intermédia *p'*, que serão utilizados para

cruzamento ou mutação. Efetivamente, este processo reduz a diversidade da população, idealmente aumentando a qualidade geral da mesma (Dianati, 2002). No entanto, é necessário notar que nunca nenhum individuo terá uma probabilidade de seleção de zero, sendo sempre possível (embora improvável) que um mau individuo seja selecionado.

Para realizar o processo de seleção, é necessário definir qual o operador de seleção a aplicar. Existe um grande número dos mesmos, podendo até o mesmo ser criado especificamente para o problema em questão, mas geralmente são considerados como os três mais comuns: seleção proporcional ao *fitness* (*fitness proportionate selection*, também conhecida como seleção em roleta); seleção por ranking ou seriação; e seleção por torneio.

A seleção proporcional ao *fitness* atribui ao individuo a probabilidade de ser selecionado como *Pi = (fitness de i / total de fitness da população)*. Ou seja, a probabilidade de um individuo ser selecionado baseia-se em como a sua *fitness* individual se compara face ao total da população. Naturalmente, este método de seleção reflete fortemente diferenças de *fitness* (um individuo excecionalmente bom terá uma probabilidade de seleção muito maior do que o próximo individuo). Uma representação comum deste método é representar o total de *fitness* como um círculo, que é depois seccionado em n segmentos (onde n é o tamanho da população), sendo o tamanho de cada segmento proporcional ao *fitness* do individuo que representa (novamente, quanto maior o individuo, maior o tamanho do segmento). Para selecionar um individuo, é, analogamente a uma roleta, "largada uma bola na roleta a girar" e o segmento onde a mesma parar indica o individuo selecionado.

A seleção por seriação ordena todos os indivíduos com base no seu *fitness*, atribuindo-lhes probabilidades de serem selecionados utilizando uma função, geralmente linear ou exponencial. Tal como na seleção proporcional ao *fitness*, quanto melhor o individuo, maior a probabilidade de o mesmo ser selecionado. No entanto, contrariamente ao operador de seleção anterior, a seleção por seriação não é sensível às diferenças de *fitness*, sendo a medida usada apenas para ordenar a população (não influenciando a sua probabilidade de seleção diretamente) – ou seja, o melhor individuo de uma população terá a mesma probabilidade de ser escolhido, quer seja excecionalmente melhor que toda a população ou apenas marginalmente melhor que o segundo melhor individuo.

A seleção por torneio utiliza subgrupos para selecionar elementos, pelo não compara todos os indivíduos simultaneamente – para além de evitar situações onde elementos de *fitness* muito elevada podem dominar o resto da população, visto não ser feita uma comparação global entre todos os indivíduos, o facto de trabalhar com subgrupos torna a execução deste operador ligeiramente mais rápida (os restantes operadores obrigam a execução de todos os indivíduos para determinar o seu grau de *fitness*) (Koza, 1992). É definida uma contante *k*, conhecida como tamanho do torneio, que determina quantos indivíduos são selecionados para um torneio. Cada torneio seleciona apenas um individuo, pelo que terão de ser corridos *n* torneios, onde *n* é o tamanho da população original. Para executar um torneio, são selecionados aleatoriamente k indivíduos da população original e dessa seleção o individuo com melhor *fitness* é selecionado para a população intermédia. Após um torneio, qualquer individuo utilizado pode voltar a ser selecionado para torneios seguintes, incluindo o que foi colocado na população intermédio. É de notar que k controla a pressão seletiva da seleção (ou quão bom um individuo deve ser para ser selecionado): um valor de k elevado traduz-se numa pressão seletiva alta, visto os torneios utilizarem mais indivíduos, sendo

menos aleatórios; um valor reduzido de k leva a uma pressão de seleção baixa, dependendo mais da seleção aleatória inicial.

3.5. *OPERADORES GENÉTICOS*

Em PG os operadores genéticos são aplicados aos indivíduos escolhidos pelo operador de seleção aplicado. A aplicação de operadores genéticos terá sempre de gerar um número de indivíduos igual ao numero de indivíduos da população original, garantindo que o tamanho da população se mantem constante durante a execução do algoritmo.

O operado genético utilizado para criar novos indivíduos é escolhido probabilisticamente, sendo que a aplicação de operadores é mutuamente exclusiva – um individuo pode ser criado através de mutação ou por *crossover*, mas nunca pelos dois simultaneamente. A escolha do operador a aplicar é feita com base na taxa de *crossover* e na taxa de mutação. Geralmente, a taxa de *crossover* é muito elevada e a taxa de mutação muito reduzida. Caso a soma das duas taxas resulte num valor *p* que seja menor que 100%, é aplicado um operador de reprodução com probabilidade 1 – *p* (Poli, 2008).

3.5.1. *Crossover*

O *crossover* consiste na escolha de (geralmente) dois indivíduos (pais), usando-os para criar novos indivíduos (filhos) através da sua recombinação – ou seja, os filhos são compostos por partes de ambos os pais (Koza, 1992).

O método mais comum de *crossover*, definido como *standard crossover* (Koza, 1992), utiliza os operadores de seleção para escolher dois indivíduos como pais, produzindo dois filhos. Seguidamente, para cada um dos pais é definido um ponto de *crossover*, escolhido com probabilidade uniforme de entre todos os pontos da árvores (nós e terminais). O ponto de *crossover* é a raiz da subárvore que será utilizada no *crossover*. O primeiro filho do *crossover* da árvore A com a árvore B consiste em remover a subárvore definida de A, substituindo-a pela subárvore de B. O segundo filho resulta da operação simétrica para B. Considere-se a figura 3.2 como exemplo.

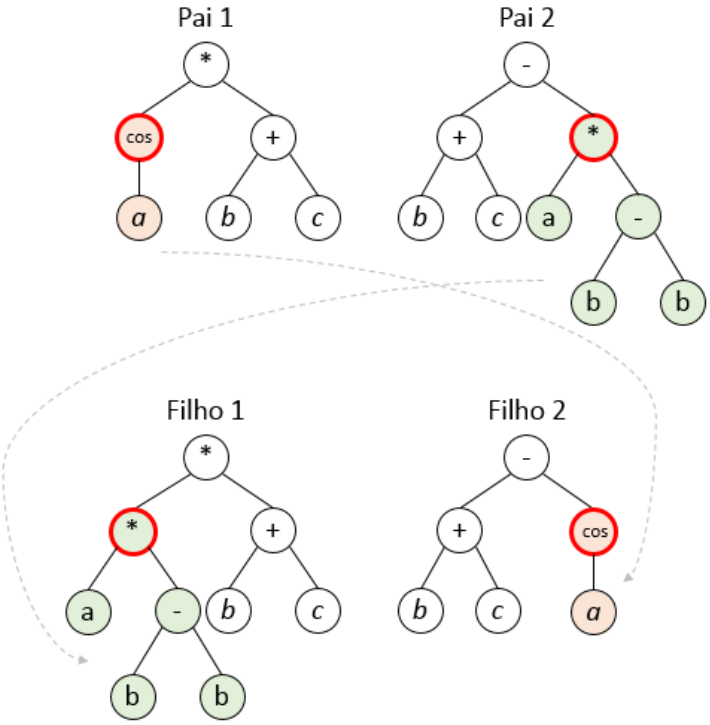


Figura 3.2 – Exemplo de *standard crossover*. Pontos de *crossover* realçados a vermelho e subárvores de 1 e 2 a laranja e verde, respetivamente.

Existem vários outros métodos de *crossover*, variando em aspetos como a probabilidade de seleção de prontos de *crossover* (por exemplo, reduzindo a probabilidade da raiz da árvore ser selecionada), o número de pais, filhos ou pontos de *crossover*, etc. Por exemplo, o método de *crossover* de Poli (Poli, 2008) utiliza dois pais com um ponto de *crossover* cada, mas gera apenas um filho. A seleção do método de *crossover* irá influenciar como a diversidade da população irá evoluir.

3.5.2. Mutação

A mutação introduz mudanças aleatórias na população, sendo utilizada para reintroduzir diversidade na população, evitando a convergência prematura (Koza, 1992). Ao contrário do *crossover*, é uma operação assexuada, pelo que requer apenas um individuo.

A operação tradicional de mutação consiste na escolha aleatória de um ponto de mutação, escolhido com probabilidade uniforme de todos os elementos do individuo (nós e terminais). O ponto de mutação, bem como a subárvore de que o mesmo possa ser raiz, é removido do individuo e no seu lugar é colocada uma nova árvore gerada aleatoriamente (Koza, 1992). A profundidade desta nova árvore é limitada por um parâmetro que controla a sua profundidade máxima. Considere-se a figura 3.3 como exemplo.

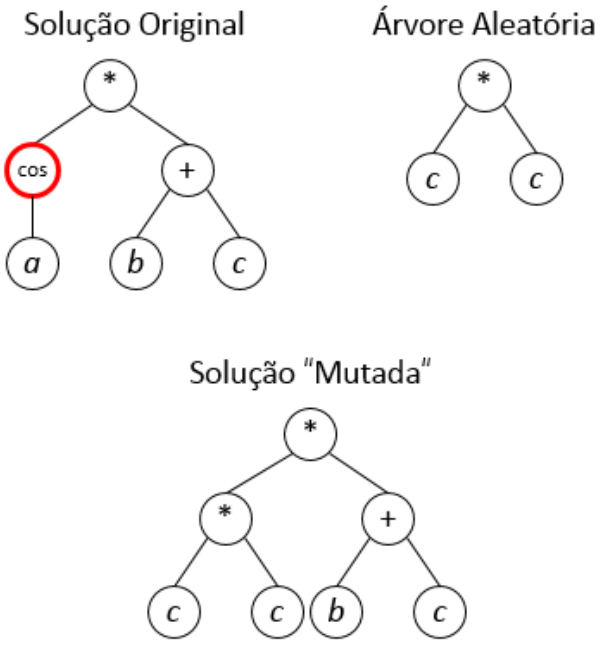


Figura 3.3 – Exemplo de mutação. Ponto de mutação realçado a vermelho.

Tal como com o *crossover*, existem vários operadores de mutação diferentes. Outro de uso também comum é definido como *point mutation*, onde é selecionado um ponto de mutação (novamente com probabilidade uniforme de todos os elementos da árvore). Este ponto é substituído por outro equivalente – uma função será substituída por outra função do conjunto de funções com a mesma aridade, e um valor terminal será substituído por um valor do conjunto de terminais do mesmo tipo. Caso o ponto selecionado seja a raiz de uma subárvore, a mesma é mantida intacta (Poli, 2008).

3.5.3. Reprodução

Em PG, a reprodução consiste simplesmente na cópia de um individuo da população intermédia para a nova população, sem aplicar operadores de *crossover* ou de mutação. Essencialmente, o individuo persiste inalterado para a geração seguinte.

A reprodução pode ser aplicada quando as taxas de mutação e de *crossover* não totalizem uma probabilidade de 100%. No caso de não ser selecionado nenhum dos operadores, é aplicada a reprodução.

Um caso notável de reprodução é o elitismo. Quando se aplica elitismo a uma execução de PG, o melhor (ou n melhores) individuo de uma população é automaticamente colocado na geração seguinte, inalterado. Isto pretende garantir que o melhor individuo não é perdido entre gerações (fenómeno que pode acontecer dado o elemento aleatório do algoritmo – por muito baixa a probabilidade de o melhor individuo não ser selecionada, é sempre teoricamente possível que o mesmo ocorra).

3.6. *Execução do Algoritmo*

Para qualquer execução de PG, é necessário definir as seguintes configurações (os pontos abaixo refletem uma execução padrão segundo Koza (Koza, 1992)):

- Conjunto de terminais (constantes, funções de aridade zero, variáveis de input);
- Conjunto de funções;
- Definir a função de *fitness* utilizada para avaliar os indivíduos;
- Definir o critério de paragem (nº máximo de gerações, atingir um individuo com determinado nível de *fitness*, etc.);
- Tamanho da população;
- Método de inicialização da população;
    - Caso desejado, definir o tamanho máximo/mínimo das soluções criadas;
- Operador de seleção;
    - No caso de ser utilizada seleção por torneio, definir o tamanho de torneio;
- Operador de *crossover*;
- Operador de mutação;
    - No caso de ser utilizada mutação *standard* (ou uma variação que gere árvores aleatoriamente), definir, se desejado, o tamanho máximo da árvore aleatória;
- Taxa de *crossover* e taxa de mutação;
- Definir se o conceito de elitismo é executado (e adicionalmente, a quantos indivíduos pode ser aplicado);
- Se desejado, indicar a profundidade máxima das árvores, limitando o crescimento das soluções a partir de certo ponto.

É de notar que duas execuções com os mesmos parâmetros provavelmente não produzirão os mesmos resultados, embora devam produzir resultados equiparáveis. Isto deve-se à natureza não determinística da PG – todas as execuções possuem um elemento de aleatoriedade, dado que as soluções são criadas aleatoriamente (essencialmente, a população inicial tende a ser sempre diferente); adicionalmente, os elementos aleatórios da seleção, reprodução e mutação introduzem diferenças adicionais de execução para execução. Este facto não implica que as soluções produzidas por execuções diferentes sejam notoriamente melhores ou piores entre si (embora exista a baixa probabilidade de uma execução com uma inicialização particularmente má produza maus resultados), mas sim que serão geralmente diferentes. Este fator leva a que se devam considerar várias execuções de uma configuração antes de se dar um parecer sobre a mesma.

Após configurada, a execução de PG segue geralmente o seguinte algoritmo:

1. Inicializar a população original *p* com *n* indivíduos
2. Enquanto o critério de paragem não for atingido, iterar:
    1. Calcular o *fitness* de todos os indivíduos de *p* (este passo é ignorado quando o operador de seleção é por torneio e não é aplicado elitismo)
    2. Aplicar o operador de seleção definido para selecionar *i* indivíduos de *p* (podendo haver repetições), colocando-os na população intermédia *p'*
    3. Até serem criado *n* indivíduos, iterar:
        1. Escolher probabilisticamente o operador genético a aplicar (mutação, *crossover* ou, se aplicável, reprodução)
        2. Escolher aleatoriamente, com probabilidade uniforme, o número apropriado de indivíduos para aplicar o operador escolher
        3. Aplicar o operador escolhido, colocando o(s) novo(s) indivíduo(s) numa população *p''*
    4. Descartar a população *p'* e substituir a população *p* por *p''*
3. O individuo com melhor *fitness* da última geração de *p* é a solução apresentada

3.7. *APLICAÇÕES TÍPICAS DE PG*

As aplicações de PG como ferramenta de programação automática, *machine learning* e de resolução automática de problemas estão registadas num enorme diversidade de problemas, não estando o seu uso restrito a apenas algumas áreas do saber. É possível afirmar que a PG é particularmente eficiente em áreas que possuam uma ou mais das seguintes propriedades (Poli, 2008):

- As relações entre as variáveis relevantes ao problema são desconhecidas ou pouco/mal compreendidas;
- A forma e tamanho da solução são desconhecidas, sendo que encontra-las representa grande parte do problema;
- Existem quantidades significantes de dados de teste num formato legível por computador;
- Existem bons simuladores para testar possíveis soluções, mas boas soluções são difíceis de gerar;
- Não é possível produzir soluções analíticas através de análise matemática convencional;
- Uma solução aproximada é aceitável, ou então é a única plausível de ser obtida;
- Pequenos aumentos do desempenho da solução são altamente valorizados.

Naturalmente, vários tipos de problemas apresentam uma ou mais das características acima. Seguidamente serão apresentados alguns tipos onde é considerado que a PG mais tem contribuído, existindo no entanto muito mais, sendo imprático criar uma lista completa. É de notar que será dada uma especial enfâse às regressões (3.7.1), visto serem o tipo de problema mais relevante para a presente tese, sendo o problema trabalhado no mesmo.

3.7.1. Regressões

Muitos problemas aos quais PG poderá ser aplicado são regressões (como no caso desta tese), onde se procura descobrir qual a função que, com base em determinados *inputs*, possa prever o seu resultado. Para tal, os dados fornecidos ao algoritmo são compostos por um conjunto de *n* variáveis

independentes ou inputs, que representam os dados conhecidos para aplicar ao problema (estes dados farão parte do conjunto de terminais como dados de input), e por um conjunto de variáveis dependentes ou *targets*, que representam o valor que os inputs deverão produzir. Um registo de dados é composto por *n inputs* diferentes e (geralmente) por um *target* associado aos mesmos, sendo que o conjunto de dados deverá possuir vários registos.

Naturalmente, este tipo de problemas requerem uma grande quantidade de dados para que o algoritmo possa "aprender" qual a função que os rege e assegurar-se que a mesma é generalizável a dados futuros. Para tal, os dados são subdivididos em conjuntos a serem utilizados pelo algoritmo em diferentes momentos da sua execução:

- Conjunto de treino (*training set*) – por regra, o conjunto que contem mais dados (sendo 70%-80% uma média generalizável). É utilizado durante o treino para determinar o grau de *fitness* dos indivíduos para o processo de seleção – ou seja, durante a evolução os indivíduos são executados para todos os registos, usando as variáveis independentes como dados de input e comparando o resultado gerado ao *target* definido (aplicando a função de *fitness* para classificar o individuo);
- Conjunto de Teste (*testing set*) – aplicado apenas no final da evolução ao melhor individuo gerado, este conjunto de dados é utilizado para confirmar a qualidade da solução. Esta confirmação é importante dado existir o risco de que o individuo possa sofrer de *overfitting*, ou seja, produz resultados bons apenas para os dados de treino (efetivamente, aprendeu apenas a função que rege os dados de treino, não a função generalizável ao problema em questão). Ao executar o individuo com dados de teste, pretende-se verificar a sua capacidade de generalização;
- Conjunto de Validação (*validation set*) – este conjunto destina-se a controlar o *overfitting* durante a aprendizagem. Caso o *fitness* de uma solução aumente para o conjunto de teste, mas fique igual ou pior para o conjunto de validação, é provável que se esteja na presença de *overfitting*. Este conjunto de dados é geralmente utilizado com outro tipo de algoritmos de aprendizagem que não PG, embora a sua aplicação seja possível.

3.7.2. *Human Competitive Results*

Estes problemas focam-se em produzir programas capazes de produzir resultados iguais ou melhores que aqueles que um ser humano conseguiria produzir – fazendo uso da noção de competitividade humana. Competitividade humana é uma noção desenvolvida por Koza (Koza, 1999), uma evolução do tradicional (mas em prática não utilizável) teste de Turing (Turing, 1950), que define critérios que um resultado deve possuir para ser considerado competitivo com resultados humanos (devendo cumprir um ou mais critérios para tal):

1. O resultado é uma invenção patenteada no passado, uma melhoria sobre uma invenção patenteada ou uma nova invenção patenteável;
2. O resultado é igual ou melhor que um resultado aceite como novo aquando a sua publicação numa publicação submetida a revisão;
3. O resultado é melhor ou igual que um resultado guardado numa base de dados ou arquivo mantido por um grupo de especialistas reconhecidos;
4. O resultado é publicável por si só como um novo resultado científico;

5. O resultado é igual ou melhor que o mais recente resultado criado por humanos para um problema cujas soluções humanas têm sido sucessivamente melhores;
6. O resultado é igual ou melhor que um resultado considerado como um grande feito no seu campo aquando a sua descoberta;
7. O resultado soluciona um problema indisputavelmente difícil no seu campo;
8. O resultado é competitivo ou vitorioso numa competição que inclua resultados humanos.

Embora estes critérios sejam aplicados a todas as áreas de Inteligência Artificial, muitas aplicações de PG produze resultados considerados competitivos com humanos, dando como curto exemplo ilustrativo: o desenho de uma antena para uso na missão *Space Technology 5* da NASA (Lohn, 2004); um algoritmo de pesquisa para problemas de xadrez de Mate em n Jogadas (Hauptman, 2007); a síntese de detetores de pontos de interesse para análise de imagens (Trujilo, 2006).

3.7.3. Processamento de Sinais e Imagens

Sinais e imagens, visto serem dois formatos facilmente legíveis por computadores, são plausíveis de serem alvo do uso de PG. Muitos dos problemas que envolvem estes tipos de dados passam pela classificação dos mesmos através da análise dos seus conteúdos e do cruzamento com bases de conhecimento previamente estabelecidas (basicamente, os programas são treinados para reconhecer certos padrões).

Muitas aplicações de PG de processamento de imagens são para fins militares, possibilitando melhores formas de vigilância automatizada – como a deteção de tanques em imagens de infravermelhos (Howard, 2006). Outras aplicações incluem a indexação ou classificação/identificação de objetos em imagens (Theiler, 1999;Zhang, 2006). Relativamente a sinais, podem ser vistas aplicações como a análise de dados de sonar (Martin, 2006) ou algoritmos para identificar características no gelo dos mares polares a partir de dados de satélites SAR (*Synthetic Aperture Radar*) (Daida, 1996).

3.7.4. Híper-Heurísticas

Híper-Heurísticas são "heurísticas usadas para escolher outras heurísticas" (Burke, 2003). Ou seja, uma heurística atua diretamente sobre o espaço de soluções de um problema com o objetivo de encontrar soluções ótimas (o mais parecido possível). Híper-heurísticas são utilizadas para definir que heurísticas se devem utilizar para um determinado problema. Dado que a heurística escolhida afeta grandemente a qualidade da solução, o uso de PG como híper-heurística permite tornar a escolha da heurística a usar mais informada (especialmente importante quando o problema é mal conhecido e a melhor heurística não é conhecida).

3.8. *PROGRAMAÇÃO GENÉTICA COM OPERADORES SEMÂNTICOS*

Uma das grandes limitações da PG tradicional é o facto dos operadores da pesquisa se basearem apenas na sintaxe dos programas evoluídos, ignorando o seu significado (Moraglio, 2012). Isto leva a que a pesquisa seja "cega" – muitas vezes (mais de 75% segundo (McPhee, 2007)) os operadores de *crossover* resultam operações irrelevantes em indivíduos que pouco ou nada melhoram em relação aos pais, ou na pior das hipóteses, resultam em operações destrutivas, que pioram o *fitness* dos filhos em comparação aos pais (McPhee, 2007). O facto de o algoritmo trabalhar com um número muito elevado de indivíduos faz com que a evolução seja sempre tendencialmente boa, mas é um

facto que muito poder computacional é "desperdiçado" em operações que acabam por pouco ou nada afetar a evolução.

O problema acima deriva da falta de sintaxe na evolução dos programas evoluídos, mas acaba por impactar apenas o tempo de execução do algoritmo. No entanto, a falta de sintaxe leva a um novo problema quando se consideram os problemas reais a que PG pode ser submetida: ótimos locais. Muitos dos problemas de real aplicação de PG não possuem uma paisagem de *fitness*[1] unimodal, ou seja, para além da melhor solução possível (denominada por ótimo global), existem áreas onde os indivíduos tendem para soluções boas (denominadas de ótimos locais), embora não tanto quanto o ótimo global. Muitas vezes, devido à inicialização ou aos operadores genéticos, grande parte de uma população encontra-se perto de um ótimo local, sendo que para se aproximar do ótimo global o *fitness* teria de piorar primeiro – ou seja, nesta população centrada num ótimo local, os indivíduos mais próximos do ótimo global são na verdade piores que os restantes. Dado que a probabilidade de os indivíduos de baixo *fitness* se reproduzirem é reduzida, geralmente a população evolui tendendo para o ótimo local. Como se pode ver na figura 3.4, o individuo 2 tende para B, o ótimo glocal, visto que à sua esquerda a *fitness* será pior; no entanto, o individuo 1 tende para o ótimo local A do mesmo modo, mas após lá chegar, dificilmente sairá de lá – qualquer outro individuo na sua vizinhança representa uma redução de *fitness*.

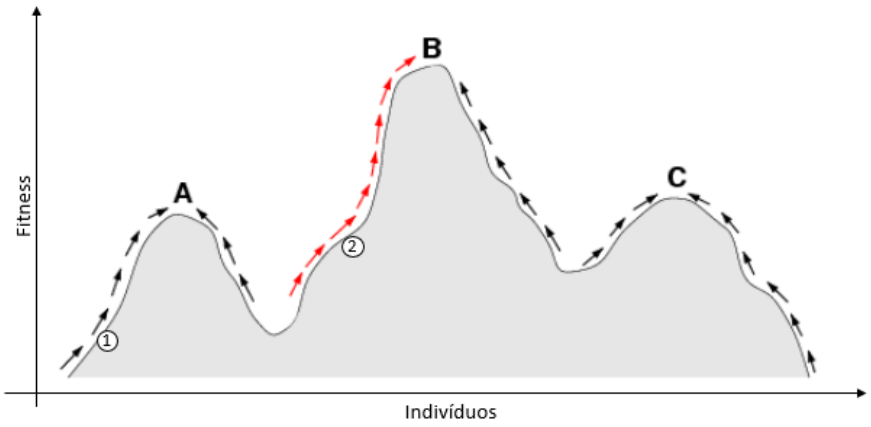


Figura 3.4 – Exemplo de paisagem de *fitness* simples.

Como solução a este problema é considerada o significado dos programas presentes na população para além da sua sintaxe: de modo a garantir que a população inicial é diversa, idealmente propagando-se por todo o espaço de solução (e assim aumentado a probabilidade de existirem indivíduos perto do ótimo global), existem métodos de inicialização que garantem a criação de indivíduos semanticamente distintos (Beadle, 2009;Jackson, 2010); operadores de *crossover* e

[1] Paisagem de *fitness* é a representação gráfica do *fitness* de todos os indivíduos possíveis de um problema, inicialmente utilizado para representar evolução biológica (Wright, 1932). O eixo vertical representa o *fitness* da solução, enquanto o eixo horizontal representa os indivíduos – organizados de um modo consistente com os operadores genéticos usados (ex: considerando os indivíduos {010; 111; 000}, se organizados pelo número de zeros da solução, teríamos um eixo com a organização "000, 010, 111"). Notavelmente, ao alterar os operadores genéticos usados, é alterada a distribuição dos indivíduos no eixo, podendo alterar completamente a forma da paisagem (incluindo a localização ou até a existência de ótimos locais) – mas o problema continuará o mesmo.

mutação que descartam indivíduos gerados sejam semanticamente idênticos aos progenitores, evitando operações genéticas que não gerem melhorias ou diversidade (Beadle, 2008;Beadle, 2009); operadores de *crossover* que analisam o individuo, identificando subárvores que não afetam o *fitness* do individuo e evitando que o ponto de *crossover* seja selecionado nas mesmas, o que evita operações de *crossover* sem impacto real (Blickle, 1994); etc. Todos os métodos apresentados relatam um aumento no desempenho de PG após a sua aplicação, e estudos subsequentes constatam que a maioria das operações semânticas não têm efeitos mais destrutivos do que os operadores tradicionais – antes pelo contrário, em média relatam melhor desempenho e maior probabilidade de gerar soluções mais viáveis (McPhee, 2007).

3.8.1. Geometric Semantic Genetic Programming

Apesar da melhoria de desempenho dos métodos semânticos, todos eles partilham uma característica: são indiretos (Moraglio, 2012). Isto é, as operações semânticas são aplicadas sintaticamente aos pais para gerarem filhos, funcionando num sistema de tentativa e erro onde os filhos são produzidos e verificados a fim de se saber se respeitam os critérios semânticos definidos – caso tal não se verifique, são descartados e novos filhos são gerados, o que faz com que haja um desperdício de capacidade computacional. Por outras palavras, os operadores desenvolvidos não trabalham diretamente no espaço semântico do problema.

Moraglio et al. (Moraglio, 2012) propõe operadores capazes de atuar diretamente no espaço semântico das soluções, garantindo que os mesmos produzem indivíduos semanticamente válidos (eliminando a vertente de tentativa e erro presente nos métodos até então existentes). Particularmente, os operadores de transformação criados perturbam aleatoriamente um único output previsto do individuo criado face aos pais. Ou seja, assumindo um individuo B transformado a partir do individuo A, o individuo B irá produzir todos os valores de output que A produzia, exceto um, perturbado aleatoriamente. Efetivamente, isto quer dizer que o individuo produzido nunca será pior que o pior dos pais (mas não necessariamente melhor que os dois, embora tal possa ocorrer).

O facto de as soluções produzidas representarem sempre uma melhoria face às anteriores faz com que a paisagem de *fitness* do problema seja sempre cónica (ou unimodal) (Moraglio, 2012), muito à semelhança de problemas de AG onde o fitness de uma solução resulta da diferença absoluta entre o resultado da solução e o target definido. Ou seja, o problema deixa de possuir ótimos locais.

Os operadores semânticos criados por Moraglio et al. são aplicados diretamente sobre os filhos, pelo que a seguinte estrutura é aplicada aos operadores genéticos de *crossover* e mutação:

- *Geometric Semantic Crossover* – para duas soluções T_1 e T_2, o *crossover* retorna (T_1 * T*r*) + ((1 – T*r*) * T2), onde T*r* é uma função aleatória cujo resultado varia entre [0, 1] (ver Figura 3.5 para uma representação em árvore do operador);

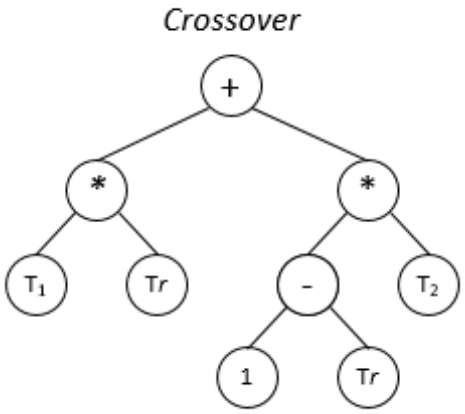


Figura 3.5 – Representação em árvore do operador de *Geometric Semantic Crossover*

- *Geometric Semantic Mutation* – para uma função T, onde *mutation step* é *ms* (a força absoluta da mutação sobre a solução), a mutação retorna T + *ms* * (Tr_1 – Tr_2), onde Tr_1 e Tr_2 são funções aleatórias cujos resultados variam entre [0, 1] (ver Figura 3.6 para uma representação em árvore do operador).

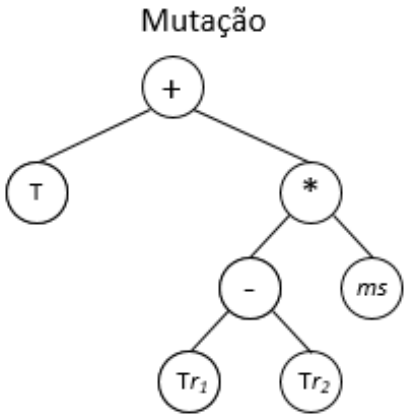


Figura 3.6 – Representação em árvore do operador de *Geometric Semantic Mutation*

É importante reparar que o individuo gerado contem a totalidade dos pais, juntamente com árvores aleatórias. Comparativamente a operadores tradicionais de PG, embora tenham um melhor desempenho, os indivíduos têm um crescimento exponencial, onde os filhos são muito maiores que os pais (Moraglio, 2012). Na pior das hipóteses, o algoritmo pode começar a enfrentar uma situação denominada de *bloat* das soluções, onde o tamanho dos indivíduos aumenta a um ritmo acelerado sem que exista um aumento equivalente do *fitness* dos mesmos (Poli, 2008). Este problema de crescimento exponencial faz com que os operadores sejam inutilizáveis em problemas reais, visto que o *fitness* é calculado a partir da execução dos indivíduos – tendo em conta o tamanho dos indivíduos após poucas gerações e a quantidade de indivíduos (e casos de *fitness* a que cada um é submetido), o processo torna-se excessivamente longo. Moraglio et al. (Moraglio, 2012) afirma que o uso de operadores semânticos necessita de uma fase de simplificação após a criação de indivíduos –

ou seja, substituir os mesmos por indivíduos semanticamente idênticos, mas menos complexos, o que permitiria reduzir o processo de avaliação. No entanto, esta fase adicional tem um custo computacional substancial, sendo que, na melhor das hipóteses, apenas retarda o problema do crescimento exponencial dos indivíduos; na pior das hipóteses, dependendo da linguagem do problema, a simplificação pode ser um processo extremamente difícil de implementar (e computacionalmente intensivo) (Vanneschi, 2013).

3.8.2. Nova Implementação de *Geometric Semantic Genetic Programming*

Vanneschi et al. (Vanneschi, 2013) propõe um novo modelo de implementação de GSGP baseado em referências. O novo modelo utiliza os algoritmos definidos por Moraglio et al. (Moraglio, 2012), mas evita o problema de crescimento exponencial das soluções ao representar as árvores de soluções através referências para outras árvores.

Designado por GS-GP (*Geometric Semantic Genetic Programming*), o algoritmo gera apenas as árvores correspondentes aos indivíduos da população original e às árvores aleatórias criadas para a aplicação dos operadores genéticos. Todos os indivíduos gerados subsequentemente são contruídos não explicitamente (ou seja, sem gerar a expressão absoluta do mesmo). Alternativamente, são construídos através do uso de referências de memória que remetem para os indivíduos originais e do operador utilizado para os gerar. Adicionalmente, a semântica dos indivíduos também é gravada em memória (entenda-se, os resultados de um individuo para todos os casos de *fitness* considerados), o que evita que a mesma tenha de ser recalculada para cada avaliação de *fitness,* calculando a semântica dos filhos através da semântica dos pais (Vanneschi, 2014).

Essencialmente, o problema do crescimento exponencial ainda está presente, visto que as soluções continuam a crescer muito rapidamente, mas o problema é minimizado porque os indivíduos, apesar de longos, são representados de um modo muito menor e mais simples de executar. A simplificação faz com que o algoritmo seja passível de ser executado muito mais rapidamente, o que torna os operadores genéticos de Moraglio aplicáveis a problemas reais (Vanneschi, 2013).

O novo algoritmo opera tendo como base as seguintes estruturas de dados:

- Um repositório de indivíduos *P*. Neste são guardados os indivíduos da população inicial e as árvores aleatórias geradas durante a execução, expressos sintaticamente;
- Uma matriz de números reais *S* de tamanho *n* x *k*, onde *n* é o tamanho da população e *k* o número de casos de *fitness* no conjunto de treino. Cada linha guarda o vetor semântico (ou seja, os resultados do individuo para todos os casos de *fitness*) do individuo correspondente da população;
- Uma matriz de números reais S_{rand}, de tamanho *m* x *k*, onde *m* é o número de árvores aleatórias geradas durante a execução e *k* é o número de casos de *fitness* no conjunto de treino. Cada linha guarda o vetor semântico da árvore aleatória correspondente. *m* pode ser definido antes da execução (caso se use um número limitado de árvores aleatórias) ou vão sendo acrescentados valores dinamicamente durante a execução;
- Uma matriz de números reais S_{new}, de tamanho *n* x *k*, onde *n* é o tamanho da população e *k* o número de casos de *fitness* no conjunto de treino. Nesta matriz são guardados os vetores semânticos dos indivíduos gerados pelos operadores genéticos que irão constituir a geração

seguinte. Como tal, S_{new} é copiada para *S* no final de cada geração e os registos guardados são apagados;

- Uma estrutura *M*, onde são guardados os indivíduos gerados por referenciação. É a partir desta estrutura que os indivíduos são reconstruídos quando necessário. Esta estrutura cresce durante a execução, contendo indivíduos no formato (*operador, (antepassado*$_1$*, antepassado*$_2$*, atepassado*$_3$*))*, onde *operador* corresponde ao operador genético utilizado para gerar o individuo e *antepassado* corresponde a uma referência a um individuo já existente: quer para *P*, quer para outros registos de *M*.

Com recurso às estruturas de dados acima, o algoritmo é, estruturalmente, muito semelhante ao algoritmo tradicional de PG:

1. Criar repositório *P*, vazio;
2. Criar matriz *S*, vazia;
3. Criar matriz S_{rand}, vazia;
4. Criar *n* indivíduos aleatórios (população inicial), guardando-os em *P*;
5. Avaliar o *fitness* dos indivíduos em *P*, guardando a semântica em *S*;
6. Criar estrutura *M*, vazia;
7. Até ser atingida a condição de fim, iterar:
    1. Criar nova matriz S_{new}, vazia;
    2. Iterar *n* vezes:
        1. Escolher operador genético a aplicar;
        2. Se escolhido *crossover*:
            1. Gerar árvore aleatória T*r* e guarda-la em *P*;
            2. Avaliar o *fitness* de T*r*, guardando a semântica em S_{rand};
            3. Selecionar dois indivíduos T1 e T2;
            4. Guardar *(crossover, (&T*$_1$*, &T*$_2$*, &Tr))* em *M*; [2]
            5. Avaliar o *fitness* do filho, guardando a semântica em S_{new};
        3. Se escolhida mutação:
            1. Gerar duas árvores aleatórias Tr_1 e Tr_2 e guarda-las em *P*;
            2. Avaliar o *fitness* de Tr_1 e Tr_2, guardando as semânticas em S_{rand};
            3. Selecionar um individuo T;
            4. Guardar *(mutation, (&T, &Tr*$_1$*, &Tr*$_2$*))* em *M*;
            5. Avaliar o *fitness* do filho, guardando a semântica em S_{new};
        4. Se escolhida reprodução (se aplicável):
            1. Copiar o registo original para *M*;
            2. Copiar o registo correspondente de *S* para S_{new};
    3. Substituir *S* por S_{new};
    4. Apagar todos os registos de *P*, *M* e S_{rand} que não sejam referenciados na nova população (ou seja, não são antepassados);
8. O individuo com melhor *fitness* da última geração é a solução apresentada

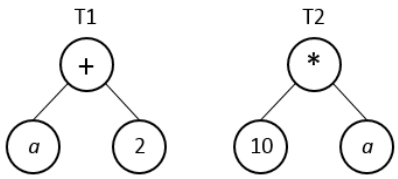
É necessário realçar alguns aspetos do pseudo-código acima. O passo 4.7 contribui para gerir o crescimento dos dados, apagando registos que não sobrevivem ao processo de seleção – visto não contribuírem para o futuro desenvolvimento das soluções, não é necessário continuarem em

memória. É importante também notar que os indivíduos em *M* inicialmente referenciam os indivíduos de *P*, mas com o avançar da evolução passarão a referenciar também indivíduos antigos de *M*. Do mesmo modo, a seleção inicial é realizada sobre os indivíduos em *P*, enquanto as seleções subsequentes serão realizadas das últimas *n* linhas de *M* (que representam a população atual).

Ao contrário do algoritmo de Moraglio, a GS-GP, devido à referenciação, avalia cada individuo num período de tempo constante (exceto os indivíduos da primeira geração) (Vanneschi, 2014). O crescimento da quantidade de dados é outro aspeto importante: embora não seja possível perpetuar indefinidamente a execução do algoritmo pois, mesmo apagando os dados irrelevantes quando aparecem, *P* cresce um máximo de *n* x 2 registos por geração (se todos os indivíduos resultarem de mutação) e *M* um máximo de *n* registos por geração (a nova população). No entanto, é possível assegurar o desempenho em tempo útil por vários milhares de gerações, o que, dado o desempenho médio do algoritmo, é geralmente mais que suficiente para assegurar uma solução satisfatória (se não ótima).

Durante a execução, os indivíduos são representados por referências, formato que não os torna aplicáveis diretamente ao problema proposto. Isto leva a que seja acrescentada uma fase adicional de "reconstrução" ao final do algoritmo, onde se substituem as referências pelos indivíduos originais. Dada a natureza dos problemas, esta fase geralmente será apenas aplicada ao melhor individuo da população, mas também implica que o mesmo será geralmente uma árvore extremamente complexa como ocorria em GSGP (pelo que a simplificação continua a ser recomendada) (Vanneschi, 2014). Alternativamente, caso a forma da solução não seja importante, apenas os resultados produzidos, o individuo poderá ser utilizado em regime de "caixa negra" para um conjunto de *inputs*, o que torna a fase de "reconstrução" desnecessária.

Veja-se o seguinte exemplo ilustrativo de uma curta execução do algoritmo. Considere-se uma população original de dois indivíduos, T1 e T2 (respetivamente, *a*+2 e 10**a*, sendo *a* uma variável). Criados os indivíduos, é criada uma tabela *P* onde é guardada a estrutura dos mesmos, como se pode ver na Figura 3.3.7.



P

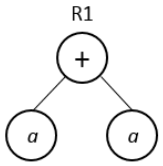
ID	Indivíduo
T1	*a*+2
T2	10**a*

Figura 3.7 – População inicial.

É também criada uma tabela *S*, não representada na Figura 3.7. Assumindo 100 casos de *fitness*, *S* terá 2x100 como dimensões iniciais, sendo que as linhas representam os resultados de um individuo para cada um dos cem casos de *fitness* (a primeira linha de S dizendo respeito ao primeiro individuo da tabela *P* e assim sucessivamente) – cada registo sendo portanto denominado de vetor semântico do respetivo indivíduo. São também criadas as tabelas *M*, S_{rand} e S_{new}, por agora deixadas vazias.

[2] Em pseudo-código, uma referencia é representada acrescentando & antes do individuo (ex: &T corresponde a uma referência ao individuo T).

Após a criação dos indivíduos, tabelas e o cálculo inicial de *fitness*, assuma-se agora que é pretendido realizar um *crossover* do indivíduo T1 e do individuo T2. Para tal é usado o operador *Geometric Semantic Crossover* de Moraglio et al. (Moraglio, 2012), sendo para tal necessária a criação de um indivíduo aleatório R1, neste caso *a*+*a*. Tal como os indivíduos da população inicial, a sua estrutura é guardada em *P*, e o seu vetor semântico é calculado e guardado em S_{rand} (Figura 3.8).



P

ID	Indivíduo
T1	*a*+2
T2	10**a*
R1	*a*+*a*

Figura 3.8 – Indivíduo aleatório R1.

Tradicionalmente, o *crossover* implicaria representar a totalidade do novo individuo. No entanto, dado que o operador de *crossover* resulta sempre numa estrutura semelhante, é possível utilizar referências para representar os indivíduos desejados, embora de um modo mais simplificado (combatendo o problema do crescimento exponencial). Tecnicamente, o individuo nunca é representado na sua totalidade (como uma expressão completa sem referências), mas todos os seus elementos estão guardados nas tabelas existentes. O novo individuo é guardado na tabela *M*, seguindo o formato já descrito. Na Figura 3.3.9, é possível ver uma comparação entre uma aplicação tradicional do operador de *crossover* e uma aplicação que utiliza referências.

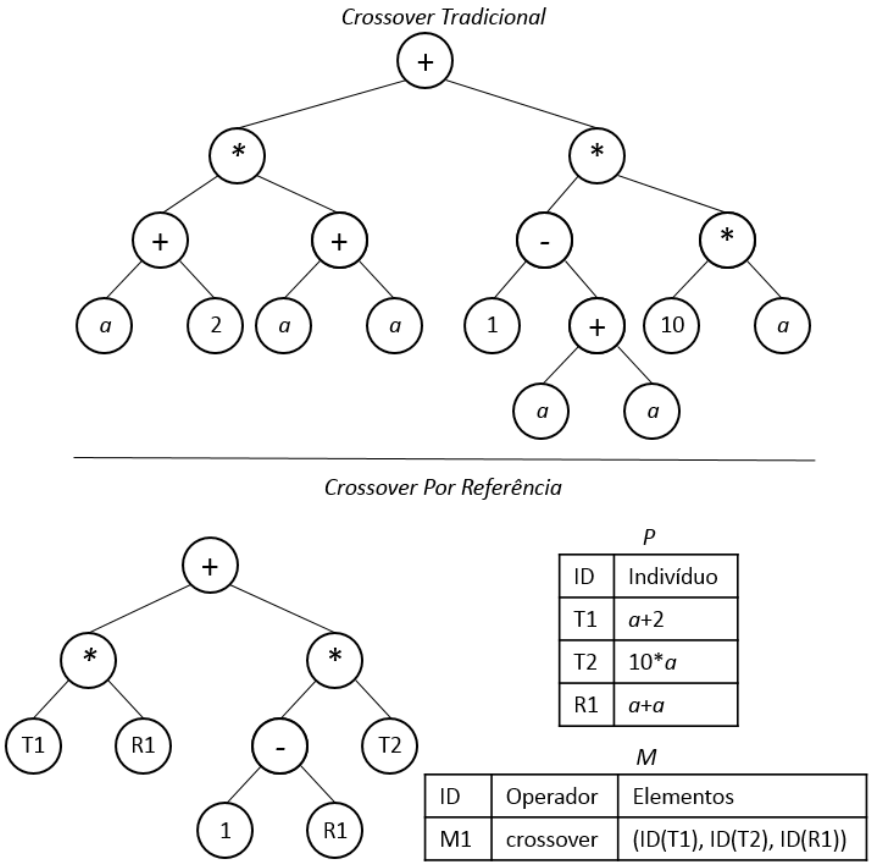


Figura 3.9 – Comparação entre *crossover* "tradicional" e por referência usando *Geometric Semantic Crossover* usando os mesmos elementos.

Na Figura 3.9 esta ilustrada a grande vantagem do uso de referências – o mesmo indivíduo apresenta-se com maior profundidade se representado tradicionalmente (problema claramente mais marcado em iterações mais tardias). O indivíduo representado por referências terá o mesmo formato independentemente da iteração – a única diferença é que as referências envolverão cada vez mais elementos posteriores. Do mesmo modo, o cálculo do vetor semântico do indivíduo esta também simplificado: são conhecidos os vetores de todos os indivíduos referenciados (guardados em *S* e S_{rand}), pelo que o cálculo do vetor semântico do novo indivíduo é relativamente simples (a termo de exemplo, é basicamente equivalente a, para cada caso de *fitness*, substituir os terminais com referências pelo valor de *fitness* do indivíduo referenciado para esse caso de *fitness* particular)[3]. Tal como foi feito para a população inicial, o vetor de *fitness* do novo indivíduo é calculado e guardado na tabela S_{new}, que no final da iteração irá substituir *S* – visto deixa de ser necessário conhecer os vetores de *fitness* da população anterior, pois estão "incluídos" nos vetores da nova população. Em iterações seguintes, a população passará a estar na tabela *M*, correspondendo sempre aos últimos *n* registos da mesma (onde *n* é o tamanho da população original), sendo os indivíduos das gerações seguintes formados a partir de elementos de *M*.

[3] Tal "substituição" é possível porque numa árvore é possível substituir qualquer subárvore pelo resultado que a mesma produz (ou seja, substituir toda uma subárvore por um único terminal) sem que isso altere o valor final da árvore.

Após a execução, um indivíduo pode ser "reconstruído" seguindo a sequência de referências até se atingir a população inicial. Naturalmente, o indivíduo será provavelmente extremamente complexo, mas essa complexidade não é tão aparente durante a execução.

3.8.2.1. Resultados Anteriores de GS-GP

Em (Vanneschi, 2013) é apresentado o desempenho de GS-GP aplicada a problemas de regressão já anteriormente estudados (ver (Archetti, 2007)), que visão prever três parâmetros farmacocinéticos de possíveis medicamentos, com base na sua estrutura molecular. Em (Vanneschi, 2014) é adicionalmente estudado um segundo problema de regressão que visa prever o consumo energético para um dia t, tendo por base informações de todos os dias até ao dia t – 1 (dados como carga da rede, condições meteorológicas, etc.). O leitor interessado é remetido para os respetivos artigos caso pretenda adquirir mais informações sobre a execução dos estudos, sendo que no presente documento apenas serão discutidos os seus resultados (e menos detalhadamente).

Em todos os problemas, é possível verificar que o algoritmo de GS-GP produziu melhores resultados que os métodos tradicionais de PG. Para todos os conjuntos de treino, o erro da previsão não só atingiu valores mais baixos do que os métodos tradicionais (ou seja, o algoritmo produziu melhores soluções), como também o fez num número muito menor de gerações – isto deve-se ao facto de que, como afirmado por Moraglio et al., num algoritmo de GSGP os indivíduos produzidos não são piores que o pior dos pais (Moraglio, 2012); essencialmente, o *fitness* melhora constantemente de geração para geração, ao contrário dos algoritmos tradicionais, onde existem uma probabilidade considerável de serem gerados indivíduos piores que os pais (sendo a mesma probabilidade muitíssimo reduzida em GSGP).

Mais importantemente, onde foi verificado que o algoritmo de GS-GP possui uma capacidade de generalização muito melhor que os métodos tradicionais de PG. É necessário clarificar que isto não significa que, ao contrário dos dados de treino, o *fitness* para os dados de teste seja consistentemente melhor de geração para geração, existindo instâncias onde o mesmo piora, mas quando tal ocorre, é de um modo limitado. Ou seja, comparando com métodos tradicionais de PG, onde o error para os dados de teste pode piorar bruscamente (de uma geração para outra, inclusive), em GS-GP o erro aumenta gradualmente, não se verificando mudanças bruscas do mesmo. Efetivamente, isto significa que o *overfitting* das soluções em GS-GP é controlado.

Em conclusão, os dados referentes aos testes permitem verificar que, embora não sem problemas (como a definição ótima do parâmetro *ms*, ou *mutation step*), o algoritmo de GS-GP verifica as vantagens de evolução semântica de Moraglio et al., mas permitindo que o algoritmo seja executado num espaço de tempo muito mais curto – e portanto, aplicável a problemas reais.

4. ENQUADRAMENTO DO PROJETO

O presente trabalho de tese enquadra-se no domínio da Marinha, nomeadamente na gestão da Zona Económica Exclusiva (ZEE) Portuguesa. Tendo em conta a vastidão da área, o presente capítulo clarifica qual o domínio exato da tese, sendo que podem ser brevemente definidas três grandes áreas a abordar: a área – o atual contexto de segurança marítima e a Marinha Portuguesa; o sistema – Oversee; e o projeto – MaSSGP.

4.1. *SEGURANÇA MARÍTIMA E A MARINHA PORTUGUESA*

Atualmente, grandes esforços são realizados a fim de assegurar a segurança e viabilidade do transporte marítimo. No entanto, o número de ocorrências de acidentes e acidentes marítimos tem continuado a aumentar, sendo registadas anualmente 140 mortes e perdas de €1,5 biliões em bens e danos só na Europa (Trucco, 2008). Globalmente, o Sistema de Transporte Marítimo (MTS) é responsável por 0,33 mortes por 100 milhões de pessoas por quilómetro, sendo 4 vezes mais perigoso que o Sistema de Transporte Aéreo (ATS). Adicionalmente, entre a Europa e a Ásia são estimadas perdas na ordem dos 24,5 biliões para o período de 2008 devido a pirataria (Bensassi, 2012), sendo estes custos agravados pelas maiores despesas que os transportes marítimos têm para se proteger de piratas (seguros, proteção, etc.) (Bensassi, 2013). Embora seja um meio indispensável de transporte e atividade económica, o transporte marítimo enfrenta vários problemas. Para além do esforço internacional para os combater, a generalidade da atividade de prevenção e proteção é realizada pelas forças navais de países costeiros, responsáveis pela proteção da sua Zona Económica Exclusiva (ZEE) – garantindo a sua viabilidade como recurso nacional e internacional. É neste contexto que se insere a Marinha Portuguesa.

É da responsabilidade da Marinha Portuguesa a administração da ZEE Portuguesa. A ZEE Portuguesa é de 3.877.408 km^2 desde a sua expansão (EMEPC, 2009), fazendo dela a décima maior ZEE do mundo. Recai sobre a Marinha Portuguesa suportar a exploração económica, comercial e recreativa da mesma através da manutenção de 3 princípios base:

- Busca e Salvamento – em toda a extensão da ZEE, a Marinha Portuguesa é responsável por monitorizar todas a embarcações presentes e assegurar a segurança das mesmas. Em casos de avaria, acidente, desastre ou interrupção das condições normais de operação, a Marinha deve responder rapidamente para assegurar a preservação de vida e, dentro do possível, minimizar perdas;
- Imposição de Lei Marítima – em toda a extensão da ZEE, a Marinha Portuguesa deverá identificar e impedir atividades ilegais, incluindo pesca ilegal, tráfico, imigração ilegal e pirataria. A não imposição de lei na ZEE trás óbvios prejuízos ao país, quer sociais como económicos;
- Monitorização e Proteção Ambiental – a fim de garantir a exploração a longo prazo dos recursos marítimos, a Marinha deve assegurar o cumprimento de regulamentos internacionais, bem como identificar e prevenir/corrigir possíveis perigos ambientais.

A importância da gestão da ZEE, juntamente com a complexidade e diversidade de tarefas que o mesmo inclui, cria uma constante necessidade de inovação tecnológica, a fim de melhor responder aos problemas existentes. A agravante dos constrangimentos orçamentais faz com que os sistemas

existentes devam ser não só eficazes como eficientes, o que leva a um grande investimento em soluções tecnológicas, tal como o sistema Oversee abaixo descrito.

4.2. *SISTEMA OVERSEE*

O sistema Oversee (http://www.oversee-solutions.com) é a solução informática desenvolvida, pela Critical Software, utilizada atualmente pela Marinha Portuguesa para auxiliar a monitorização da ZEE Portuguesa. É uma ferramenta de suporte de operações, possuindo módulos direcionados aos três princípios base a serem mantidos pela Marinha acima referidos.

O sistema Oversee atua como ponto de controlo centralizado para todos os dados, permitindo juntar informações de trafego, alertas, serviços meteorológicos e oceanográficos, sendo depois correlacionada para melhor monitorizar a ZEE – incluindo sistemas de aviso. O sistema possui também ferramentas de apoio à decisão, incluindo mapas de riscos, diagramas de impacto em missão, etc. Adicionalmente, atua como sistema de gestão de incidentes, permitindo despoletar, gerir, controlar, analisar, reportar e auditar incidentes, bem como indicadores de desempenho dos mesmos.

O sistema Oversee consiste, do lado do utilizador, num mapa da zona desejada, onde é sobreposta a informação desejada. Para o tráfego atualmente em trânsito na ZEE, é possível ver: coordenadas atuais da embarcação; direção da embarcação; velocidade atual; informações sobre a embarcação; etc. A posição é atualizada periodicamente com base das informações que chegam ao sistema – por lei, todos as embarcações na ZEE devem transmitir periodicamente um conjunto de informações, incluindo a sua posição. O sistema integra na imagem informações de fontes diversas: AIS (*Automatic Identification System*); SAT-AIS (*Satelite AIS*); VTS (*Vessel Traffic Service*); sistemas SATCOM (LRIT, MONICAP, etc.), entre outras.

Dada a sua importância para a presente tese, é necessário realçar o AIS, visto ser do mesmo que provêm a maioria dos dados utilizados. O sistema AIS é uma ferramenta de segurança marítima utilizada para comunicação entre navios em curso e entre navios e o litoral, sendo utilizado para evitar colisões e permitir que entidades litorais consigam ter informação sobre os navios que circulam perto da costa (Tetreault, 2005). O AIS transmite três tipos de informação, para além de permitir o envio de curtas mensagens (IALA, 2002): dinâmica, estática e relativa à viagem atual. A informação dinâmica consiste em dados sobre o estado atual do navio – posição (por coordenadas GPS), direção, velocidade, etc. – e é transmitida quase em tempo real (dependendo de condições como velocidade, posição, etc., estas podem ser feitas com segundos de diferença ou com minutos de diferença). A informação estática diz respeito a características do navio – nome, identificação, dimensões, país de registo, etc. – e é enviada, geralmente, de 6 em 6 minutos. A informação relativa à viagem atual é igualmente transmitida de 6 em 6 minutos, sendo constituída por informação como destino atual da embarcação, tipo e quantidade de carga, etc.

As transmissões periódicas do sistema AIS podem ser interrompidas por diferentes razões: o navio em questão está fora do alcance de uma estação capaz de receber as transmissões; falha técnica da transmissão (falha mecânica do equipamento, da rede de comunicação, etc.); o transmissor AIS do navio foi deliberadamente desligado; ou a ocorrência de um problema com o navio. Para esta tese, são relevantes as duas últimas razões, pois podem indicar problemas de segurança, quer do ponto de vista legal (as transmissões são legalmente obrigatórias, pelo que uma embarcação que as desligue

deliberadamente pode não querer ser controlada, possivelmente para fins ilícitos), quer do ponto de vista de segurança (um possível acidente a bordo pode afetar o sistema AIS, pelo que a embarcação em apuro deixa de poder ter a sua posição monitorizada para auxiliar operações de resgate). O facto de que o sistema AIS pode não permitir saber a posição de um navio num caso onde a intervenção da Marinha seja necessária apenas sublinha a necessidade da existência de ferramentas alternativas que permitam determinar qual a real posição do navio.

Finalmente, o sistema Oversee consegue também sobrepor no mapa informações adicionais, como o histórico de posições de uma embarcação, informação definida pelo utilizador (delimitação de áreas, etiquetagem, pontos, etc.), sendo a função mais relevante para a presente tese a capacidade de sobrepor informação gerada pelas ferramentas de apoio à decisão.

4.3. *PROJETO MASSGP*

O projeto MaSSGP (formalmente "Melhorando a Programação Genética Semântica para a Segurança Maritima, a Salvaguarda da Vida Humana e Proteção Ambiental", ou "*Improving Semantic Genetic Programming for Maritime Safety, Security and Environmental Protection*" – página do projeto em http://www.novaims.unl.pt/massgp/project.asp) tem três objetivos principais:

1. Desenvolver um sistema de PG que melhore o estado da arte em problemas de classificação e regressão;
2. Integrar o sistema desenvolvido numa ferramenta industrial de apoio à decisão;
3. Usa o sistema desenvolvido para gerar novos modelos de compreensão e análise de informação para aplicações de busca e salvamento, aplicação da lei e proteção ambiental no contexto marítimo.

Uma resposta às necessidades de eficácia e eficiência da Marinha são sistemas de Inteligência Computacional (IC), dada a sua capacidade de operarem com grandes volumes de dados, retirando dos mesmos informação útil. Os sistemas de IC têm-se tornado mais comuns no contexto da Marinha, existindo vários exemplos de deteção de anomalias na navegação marítima através de: *clustering* de padrões de navegação normais usando *expert systems* (Kazemi, 2013;Laxhammer, 2008); análise visual interativa (Riveiro, 2008); AGs (Chen, 2014), etc.

Embora o interesse no uso de IC para fins de controlo marítimo tenha vinda a aumentar nos últimos anos, a PG em particular não tem ainda muita aplicação. Dada a aplicação de sucesso de AGs (Chen, 2014), é possível ver que PG terá aplicações na área. De fato, ao aplicar algoritmos de PG a problemas de regressão do domínio da Marinha, a PG produz resultados muito satisfatórios, mesmo quando comparados aos produzidos por outros métodos mais frequentemente utilizados (Vanneschi, 2015).

Ao contrário da maioria de implementações de IC na área, o projeto MaSSGP não pretende identificar anomalias, mas sim permitir uma melhor respostas às mesmas. Como visto em 4.2, o AIS permite controlar a posição de um navio graças às transmissões quase em tempo real das mesmas – sendo que a maioria dos sistemas de IC utilizam estas transmissões para estudar o comportamento dos navios em trânsito, ativando alarmes quando os mesmos mostram comportamentos anormais. No entanto, existem várias situações onde o sistema AIS pode deixar de transmitir, nomeadamente: em caso de acidente, onde o transmissor pode ficar inutilizável (especialmente perigoso caso isso se

deva a situações que ponham em perigo a tripulação); ou em caso de atividades ilegais (tráfico de droga, imigração ilegal, pirataria, etc.), onde as transmissões AIS podem ser desligadas para evitar que as autoridades consigam intervir tão eficazmente. Embora a perda de sinal representa um alerta por si só, em todos os casos significa também que as autoridades deixam de ter acesso à posição exata da embarcação, sabendo apenas a última posição antes da mesma desaparecer. O projeto MaSSGP pretende prever qual a localização mais provável de uma embarcação assim desaparecida, tendo por base as posições qua a mesma tomou até as transmissões deixarem de serem registadas. Deste modo, pretende-se permitir uma atuação mais eficiente das autoridades marítimas, reduzindo o tempo de procura da embarcação em questão.

Após o estudo das paisagens de *fitness* dos problemas – classificações e regressões – foi notado que as mesmas são extremamente complexas. Desse modo, serão utilizados operadores genéticos semânticos, que, como visto no capítulo 3.8, já deram provas de serem capazes de simplificar paisagens de *fitness* complexas, permitindo atingir com relativa facilidade a solução desejada. A aplicação do algoritmo de GS-GP produziu resultados preliminares muito promissores (Vanneschi, 2015). De modo a estudar e validar os resultados futuros, identificando problemas do âmbito específico da Marinha, a ferramenta existente deve ser integrada no sistema Oversee (não em ambiente *live*), permitindo que profissionais analisem e validem qualitativa e estatisticamente os resultados produzidos, a fim de identificar possíveis problemas para que os mesmos sejam corrigidos.

O projeto MaSSGP é desenvolvido pela NOVA Information Management School (NOVA IMS, anteriormente ISEGI), com o apoio da Critical Software SA, da Faculdade de Ciências da Universidade de Lisboa (FCUL) e da Universidade de Coimbra (UC). Adicionalmente, atuam como instituições consultantes a Marinha Portuguesa e o Computer Science Group – College of Engineering, Mathematics and Physical Sciences.

O presente trabalho de tese insere-se no projeto MaSSGP tendo por âmbito realizar a primeira integração do algoritmo GS-GP com o sistema Oversee, a fim de permitir visualizar e validar os resultados do algoritmo no sistema. A presente tese pretende também, dentro do tempo disponível, estudar os resultados obtidos e apontar os possíveis problemas que se venham a verificar, bem como acomodar novas necessidades que derivem da implementação (ou seja, novas funções necessárias identificadas após ver o algoritmo a interagir com o sistema Oversee).

5. METODOLOGIA

Serve o presente capítulo para documentar os dados utilizados no presente trabalho de tese (fonte, formato, uso, etc.) bem como as configurações do algoritmo de GSGP utilizado. Adicionalmente, são também apresentados os programas utilizados para desenvolver a tese e as especificações técnicas do meio de desenvolvimento.

5.1. *DADOS*

Como referido em 4.2, os dados utilizados para realizar as previsões provém do AIS, fornecidos pela Critical Software. Os dados guardados no sistema descrevem em detalhe o navio, as suas movimentações e intenções (IALA, 2002), podendo ser organizados em quatro grandes conjuntos de dados (dada a natureza sensível dos dados, os mesmo serão descritos apenas com detalhe suficiente para ser compreendida a sua função em relação ao presente trabalho de tese; o mesmo é aplicável à tabela 5.1, sendo uma representação apenas detalhada o suficiente para a estrutura relevante da base de dados ser compreendida – os nomes das tabelas, por exemplo, não correspondem aos reais nomes presentes na base de dados):

- Posição Atual – coordenadas (latitude e longitude), data do registo, velocidade, direção, etc.;
- Histórico de Posições – idêntico aos dados de Posição Atual, mas referentes a posições passadas;
- Fonte do Tráfego – identificador, nome, descrição, etc. da fonte do tráfego;
- Perfil da embarcação – nome, identificador, características, tipo de embarcação (militar, de pesca, transporte, recreação, etc.), nacionalidade, capacidade de carga, etc.

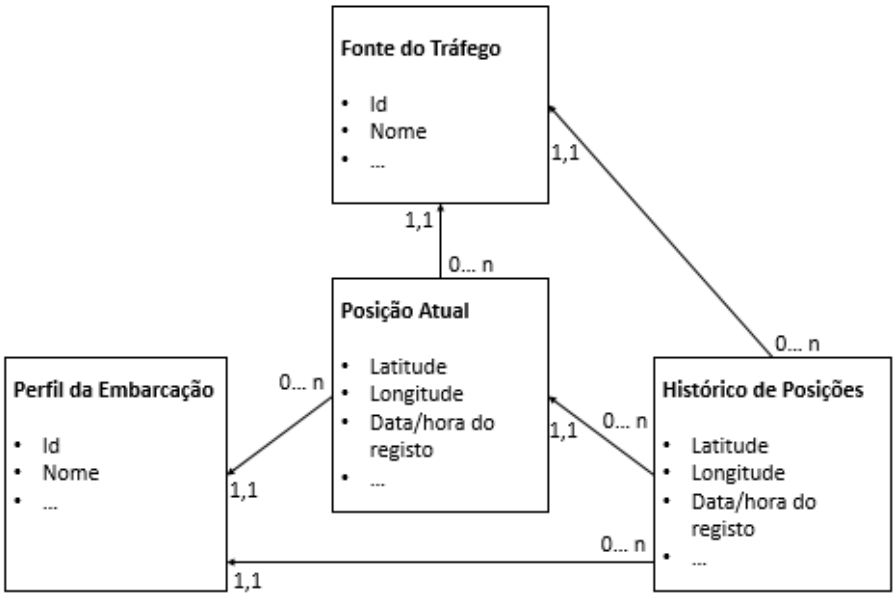


Figura 5.1 – Estrutura (simplificada) da base de dados utilizada.

Para a presente tese, são utilizados apenas os dados referentes à posição da embarcação – latitude e longitude – e a hora do registo dos mesmos. É utilizado o identificador da embarcação apenas para aceder à posição atual e ao histórico da mesma, e o nome para a criação de um novo registo onde

será guardada a previsão. É também importante dizer que estão presentes as seguintes regras de base de dados: não pode existir uma posição sem estar associada a uma embarcação; não é possível existir um registo no histórico sem existir uma posição atual; não é possível existir uma posição atual ou um registo histórico sem existir uma fonte de dados.

O conjunto de dados para realizar a execução consiste na posição atual da embarcação e pelo histórico da mesma, do registo mais recente para o mais antigo. Dado que a execução do algoritmo deve ser realizado no menor tempo possível, foi implementado um máximo de 30.000 registos que podem ser retirados do histórico – especialmente importante porque o histórico pode conter milhões de registos antigos, que se considerados na sua totalidade tornariam a execução excessivamente lenta.

Antes de divididos em conjunto de teste e conjunto de treino, os dados são formatados para o formato esperado pelo programa: um número definido de variáveis independentes (os valores de input para os indivíduos da população) e um valor *target* (o valor que, para as dadas variáveis independentes, o indivíduos deverá produzir – é por isso a base do cálculo do *fitness*).

P_1	P_2	P_3	…	P_{n-1}	P_n	T_1

Figura 5.2 – Exemplo de um registo de *input* para o algoritmo de GS-GP

Na figura 5.2, é possível ver as variáveis independentes P, onde *n* é o número de variáveis independentes definido, e T representa o *target*. É de notar que o valor de *n* é contante para todos os registos criados.

Todos os dados (P e T) correspondem a posições geográficas de latitude e longitude – sendo cada uma delas tratada em conjuntos separados. T representa a posição que se pretende que o individuo seja capaz de prever e P1 é a primeira posição registada *x* horas antes de T. Ou seja, num registo, T é a posição mais recente e P1 é a primeira posição um *x* número de horas antes, sendo P2, P3, etc. as posições imediatamente após P1, até serem atingidos *n* valores. O registo seguinte terá como *target* T2, a posição imediatamente antes de T1. Idealmente, isto significa que o primeiro registo *x* horas antes de T2 será P2, mas dado que as posições no histórico não são transmitidas a intervalos regulares (como visto em 4.2) é possível que seja P1 ou um valor não registado no primeiro registo. É de notar que os registos representam posições progressivamente mais tardias, estando as mais recentes no topo do conjunto. Na figura 5.3 é possível ver uma representação ideal de registos, para $n = 5$.

P_1	P_2	P_3	P_4	P_5	T_1
P_2	P_3	P_4	P_5	P_6	T_2
P_3	P_4	P_5	P_6	P_7	T_3

Figura 5.3 – Registo idealizado. P_1 ocorreu *x* horas antes de T_1, P_2 ocorreu *x* horas antes de T_2, etc.

Após formatados os dois conjuntos, são divididos em 2 subconjuntos: o conjunto de treino (possuindo 70% dos registos); e o conjunto de teste (possuindo os restantes 30%). Não é considerado conjunto de verificação. Adicionalmente, os registos em ambos oc conjuntos estão ordenado do mais recente (no topo) para o mais antigo (no final), sendo que o conjunto de teste possui os registos mais recentes (essencialmente, possui os primeiros 30% de registos do conjunto original). Isto significa que para ambos as posições (latitude e longitude), os mesmos registos de cada conjunto equivalem à mesma posição (por exemplo, os *targets* dos primeiros registos do conjunto de treino para latitude e os *targets* dos primeiros registos do conjunto de treino longitude representam uma única posição existente no histórico).

A fim de serem utilizados pelo algoritmo de GS-GP, os dados são formatados no formato visto na Tabela 5.1 e guardados em ficheiros de texto separados. Por execução são gerados 4 ficheiros: conjunto de teste para latitude; conjunto de treino para latitude; conjunto de teste para longitude; conjunto de treino para longitude.

Conjunto de Treino	Conjunto de Teste
Nº de pontos (*n*)	Nº de pontos (*n*)
Nº de registos	Nº de registos + 1
P_1 P_2 P_3 … P_n T_1	T_1 T_2 T_3 … T_n 0
P_2 P_3 P_4 … P_{n+1} T_2	P_1 P_2 P_3 … P_n T_1
	P_2 P_3 P_4 … P_{n+1} T_2

Tabela 5.1 – Exemplo de ficheiros de *input* para o algoritmo de GS-GP

Em maior detalhe, os dados de *input* possuem sempre um cabeçalho constituído por dois valores inteiros: o número de pontos, ou seja, o número de variáveis independentes por registo (quantos valores existem por linha para lá do *Target*); e o número de registos (ou o número de linhas em cada conjunto). O conjunto de treino é formado pelo simples acrescento deste cabeçalho aos 70% do total dos registos selecionados. No entanto, o conjunto de teste possui um registo adicional (e correspondente incremento ao nº de registos): este registo representa os *n* pontos mais recentes como variáveis independentes (que constituem os *targets* dos registos seguintes), sendo o *target* zero. Este registo adicional representa a previsão que se quer realizar – o valor zero atua como *placeholder* de um valor não conhecido – a previsão desejada, a *x* horas do registo mais recente (neste caso, T1). Como o conjunto de teste apenas é aplicado após a seleção do melhor individuo produzido pelo algoritmo, o valor zero não vai enviesar a produção do individuo, apenas o grau de *fitness* que o conjunto de teste produz para o último individuo (que neste ponto, para a presente tese, não é relevante).

A existência do registo adicional mencionado deriva do formato do *output* do algoritmo de GS-GP. No momento de escrita deste documento, não é possível retirar do algoritmo a expressão que forma o melhor individuo da população devido a sua complexidade. É no entanto possível submeter o individuo a um conjunto de teste e aceder aos resultados que o mesmo produz, que são expressos num ficheiro de texto gerado pelo algoritmo. Ao acrescentar o registo adicional no topo do conjunto

de teste, o primeiro resultado produzido pelo algoritmo é a previsão desejada, tendo por input as posições mais recentes (e o valor target acaba por não influenciar o cálculo de modo algum). Essencialmente, o melhor individuo atua em regime de *black box*, visto que a sua expressão nunca é em ponto algum conhecida.

Finalmente, é de notar que na 4ª iteração da tese, a estrutura de dados é alterada devido à necessidade de testar novas configurações. Qualquer mudança será devidamente identificada, mas não serão utilizados dados adicionais do AIS.

5.2. *FERRAMENTA DE GSGP*

Durante o presente trabalho de tese, foram utilizadas três ferramentas para implementar algoritmos de PG: GPLAB (Silva, 2003); GSGP (Vanneschi, 2015); e GSGP-LIN (uma nova versão de GSGP).

O GPLAB é uma *toolbox* para MATLAB que permite a utilização de funções de PG na plataforma, que até à sua introdução possuía apenas *toolboxes* direcionadas apenas para AGs (Silva, 2003). Para além das funções tradicionais de PG, implementa também técnicas de controlo de *bloat* e adaptação automática das probabilidades de operadores genéticos durante a execução do algoritmo. Sendo altamente modular, permite a criação e teste de novas funcionalidades, e, graças ao ambiente MATLAB, permite uma análise completa da execução e dos resultados, através da criação de gráficos representativos do desempenho do algoritmo (evolução de *fitness*, representação em árvore dos indivíduos, etc.).

A ferramenta GPLAB foi utilizada numa fase inicial do presente trabalho de tese devido à facilidade em realizar testes com os dados. No entanto, a mesma foi trocada pelo algoritmo GSGP por duas principais razões: o GPLAB não possuía na altura acesso a operadores genéticos, essenciais para a realização da presente tese; e devido ao enfâse da ferramenta na análise da execução, o seu tempo de desempenho é mais elevado do que uma especializada na execução apenas – adicionalmente, a vertente analítica do GPLAB perde relevância dado que a presente tese se centra na implementação de um algoritmo que já deu provas de bons resultados (Vanneschi, 2015;Vanneschi, 2013;Vanneschi, 2014).

A ferramenta GSGP em C++ (Vanneschi, 2015) implementa os operadores genéticos propostos por Moraglio et al. (Moraglio, 2012), através dos métodos de referenciação propostos por Vanneschi et al. (Vanneschi, 2013). Embora não possua os elementos analíticos do ambiente MATLAB, estão documentados os seus bons resultados na aplicação de GSGP a problemas reais, bem como tempos de execução relativamente pequenos (Vanneschi, 2015). Estas características tornam a ferramenta apropriada para a implementação pretendida, visto ser importante não só a qualidade das soluções produzidas, como também o tempo necessário para as produzir. A ferramenta GSGP aceita como *input* os ficheiros descritos em 5.1, atuando como *black box* e disponibilizando apenas os resultados da execução, não a função que a produziu (para a presente tese, é considerado aceitável não possuir a função do individuo produzido, sendo o foco os resultados).

Numa fase final da tese, a ferramenta GSGP foi substituída pela versão mais recente da mesma que implementa escalamento linear, ou *linear scaling*, proposto por Keijzer (Keijzer, 2003). Denominada GSGP-LIN, itera sobre a ferramenta GSGP ao alterar o cálculo do erro (em GSGP expresso por RMSE)

para fazer uso de escalamento linear. Como provado em (Keijzer, 2003), o escalamento linear pode ser utilizado para determinar o declive (*b*) e a ordenada (*a*) da função de um individuo de PG:

$$b = \frac{\sum_{i=1}^{m}[(y_i - \bar{y})(t_i - \bar{t}))]}{\sum_{i=1}^{m}(t_i - \bar{t})}$$

$$a = \bar{t} - b\bar{y}$$

Onde, para determinado caso de *input* para um individuo, *m* é o nº de casos de fitness, y_i o output do individuo para o *input* em questão, $\bar{y}$ a média dos outputs do individuo, t_i o *target* esperado do input e $\bar{t}$ a média dos *targets*. Calculados *a* e *b*, podem ser utilizados para minimizar a soma dos erros quadráticos. Assim, o *fitness* de um individuo anteriormente calculado como RMSE simples passaria a ser calculado como:

$$RMSE(t, a + by) = \sqrt{\frac{\sum_{i=1}^{m}(a + by_i - t_i)^2}{m}}$$

Onde *y* representa o output de um individuo para um conjunto de *inputs*. A aplicação de escalamento linear remove a necessidade de o algoritmo de PG ter de aprender as duas constantes por si só, tendo apenas de procurar a expressão que mais se aproxima da função pretendida (a solução ótima) (Keijzer, 2003). Embora não represente necessariamente em todas as aplicações uma melhoria de desempenho, a aplicação de escalamento linear garantidamente não produz resultados piores do que a não aplicação (Vanneschi, 2015). Desse modo, embora nem para todos os casos o desempenho de GSGP-LIN seja melhor do que a primeira versão da ferramenta GSGP, será pelo menos equivalente (nunca pior).

5.3. *CONFIGURAÇÃO DE GSGP*

O presente trabalho parte das conclusões retiradas por Vanneschi et al. (Vanneschi, 2015), tendo as configurações usadas nas experiências já realizadas produzido bons resultados. Como tal, a maioria do presente trabalho utiliza as mesmas configurações para as execuções de PG. Nos casos onde as configurações são alteradas, todas as alterações serão mencionadas – os campos não mencionados continuaram com os valores definidos na tabela 5.2.

Parâmetro	Valor(es)
Conjunto de Terminais	100 Pontos de localização cronologicamente ordenados (1 registo)
Conjunto de Funções	{+, -, *, %[4]}
Divisão entre conjunto de treino/teste (% do total de registos)	70% e 30%, respetivamente
População	100 Indivíduos
Tipo de *Crossover*	*Geometric Semantic Crossover*

[4] % representa divisão protegida, onde é retornado 1 cado ocorra um divisão por zero.

Probabilidade de *Crossover*	70%
Tipo de Mutação	*Geometric Semantic Mutation*
Probabilidade de Mutação	30%[5]
Mutation Step	0,1
Tipo de Inicialização	Ramped Half-and-Half
Profundidade máxima inicial	6
Algoritmo de Seleção	Torneio (tamanho 10)
Condição de Fim	Execução de 100 gerações
Tipo de Problema	Minimização
Função de Fitness	RMSE, ajustado com *Linear Scaling*

Tabela 5.2 – Configuração padrão de GSGP.

Adicionalmente, todos os resultados apresentados derivam de 30 execuções para a configuração apresentada. No caso de representações de mapas, é escolhida a que melhor representa a média dos resultados obtidos (sendo uma análise qualitativa dos mesmos). No caso de existirem no conjunto de soluções geradas *outliers* (soluções excessivamente boas ou más), a configuração é executada por 30 vezes adicionais (sem qualquer alteração de parâmetros), a fim de despistar a ocorrência de inicializações particularmente boas ou más no resto execuções originais (embora altamente improvável, a natureza não determinística da PG faz com que tal seja possível).

5.4. *CÁLCULO DE DISTÂNCIAS GEOGRÁFICAS*

Para proceder ao cálculo de distâncias geográficas (em quilómetros), necessário para avaliar a qualidade das soluções produzidas pelo algoritmo, foi utilizado a fórmula do cálculo da distância entre dois pontos geográficos (considerando a sua latitude e longitude) usada por produtos GeoDataSource (http://www.geodatasource.com), utilizando o algoritmo abaixo descrito.

Sendo Lat1 e Lon1, respectivamente, a latitude e longitude de um ponto geográfico, Lat2 e Lon2 as mesmas coordenadas de um segundo ponto e $\alpha = Lon1 - Lon2$, todos os valores são transformados de graus para radianos segundo o seguinte cálculo (onde *rad* equivale ao valor em radianos de *deg*, o valor original em graus decimais):

$$rad = deg * \pi \div 180$$

Mantendo a mesma nomenclatura para os valores transformados, a distância é então calculada, em radianos, segundo a seguinte fórmula (sendo *d* a distância em radianos):

[5] Dado que a soma das probabilidades de *crossover* e de mutação é 100%, não existe reprodução.

$$d = \sin(Lat1) * \sin(Lat2) + \cos(Lat1) * \cos(Lat2) * \cos(\alpha)$$

d é depois transformado novamente para graus, seguindo a fórmula (sendo *dDeg* o valor em graus decimais):

$$dDeg = d * 180 \div \pi$$

Finalmente, o resultado é ajustado para quilómetros segundo a fórmula (sendo *dKm* o valor da distância em quilómetros):

$$dKm = (dDeg * 60 * 1{,}1515) * 1.609344$$

5.5. *ESPECIFICAÇÕES TÉCNICAS*

O presente trabalho de tese foi desenvolvido em Java, através do IDE Eclipse, versão Luna 4.4.0 (https://eclipse.org/). As interações com o programa de GSGP (desenvolvido em C++) foram realizadas através do terminal de comandos Cygwin (https://www.cygwin.com/). A primeira versão da ferramenta de GSGP encontra-se disponível em http://gsgp.sourceforge.net/, a versão GSGP-LIN não se encontra disponível para *download* aquando a escrita deste documento, tendo sido gentilmente cedida por Leonardo Vanneschi, Mauro Castelli e Sara Silva.

Foi utilizada a versão 2010 do MATLAB (https://www.mathworks.com/products/matlab), juntamente com a *toolbox* GPLAB (http://gplab.sourceforge.net/).

O ambiente que produziu os resultados expressos no presente documento possuía as seguintes especificações:

- Processador Intel Core i7 4790
- 8 GB DDR3 1800MHz de RAM
- Windows 8.1 Professional, executado em SSD

6. FASE DE IMPLEMENTAÇÃO

No presente capítulo será descrita a implementação do algoritmo GS-GP no programa Oversee da Critical Software. A implementação está dividida em quatro iterações distintas, sendo em cada uma delas realçados os objetivos, procedimentos, problemas e resultados mais relevantes. Devido à constante mudança do *software* produzido, a composição técnica do mesmo apenas será explorada detalhadamente na última iteração, visto todas as anteriores serem a esse ponto obsoletas – serão descritas tecnicamente apenas em relação às suas componentes relevantes, não em detalhe.

6.1. *ITERAÇÃO 0*

A primeira iteração foi desenvolvida fora da Critical Software, sendo maioritariamente uma iteração de exploração de dados. Foi utilizado um conjunto de dados de teste para tentar replicar os resultados obtidos em estudos anteriores, produzindo entretanto módulos que viriam a ser usados na aplicação ainda a ser produzida. Considera-se a presente iteração como zero pois não lida diretamente com o *software* produzido para a Critical Software.

6.1.1. Objetivos

Os objetivos da iteração zero foram os seguintes:

- Desenvolvimento de um algoritmo capaz de formatar corretamente os dados de teste para um formato legível pelo algoritmo de GS-GP (recorrendo ao MatLab);
- Realizar execuções de controlo com dados de teste cujo os resultados já se encontravam documentados, a fim de garantir que as configurações, formato de dados, etc. haviam sido feitas corretamente;
- Preparação dos módulos criados para integração com a aplicação real.

6.1.2. Procedimentos

O principal objetivo da iteração zero era automatizar a conversão de dados para um formato legível pela aplicação – independentemente do formato original dos mesmos. Como foi visto no capítulo 5.1, o algoritmo de GS-GP foi criado para funcionar apenas com um determinado formato de *input*, como pode ser visto na tabela 3.

2				
4				
P_1	P_2	P_3	P_4	T_1
P_2	P_3	P_4	P_5	T_2

Tabela 6.1 – Exemplo de *input* para o programa de GS-GP.

Na tabela 3, podemos ver os vários elementos necessários à execução: a primeira linha conta o número de casos de *fitness* existentes no ficheiro de input (neste caso, 2 registos); a segunda linha conta o número de variáveis independentes por caso de *fitness* (sendo este número constante para todos os registos), sem contar com o target (neste caso, cada registo possui 4 variáveis

independentes, representadas por P – o target, representado por T, não é contabilizado); e finalmente os vários casos de *fitness*, representados por um número constante de variáveis independentes (como expresso na segunda linha) e um target, ou variável dependente – todos separados por tabulações (*tabs*).

Para o presente trabalho de tese, o formato de *input* desejado possui casos de fitness compostos por um número a determinar de posições geográficas (latitude ou longitude, sendo que cada uma necessita de uma execução separada, nunca são utilizadas simultaneamente), organizadas cronologicamente da mais recente para a mais antiga, sendo o *target* a posição que a embarcação ocuparia *x* horas após a primeira posição do determinado caso de *fitness*. Por exemplo, considere-se o seguinte caso de *fitness*:

$$P_1 \qquad P_2 \qquad P_3 \qquad P_4 \qquad T_1$$

Neste caso, assumindo que se está a realizar um previsão para 2 horas, T_1 foi registado no AIS 2 horas após P_1. Já P_2 foi registado imediatamente antes de P_1, P_3 imediatamente antes de P_2, etc.

Os dados utilizados nesta iteração eram compostos por uma lista de elementos referentes a um único navio – posição, tempo da transmissão, etc. – organizados cronologicamente do mais recente para o mais antigo. Visto que o algoritmo de GS-GP necessita dos dados em ficheiros de texto simples (.txt) para realizar a leitura dos dados, o output do programa criado corresponderia a dois ficheiros, ambos com o formato acima referido: um conjunto de treino e um conjunto de teste.

O algoritmo criado para criar os ficheiros necessários atua da seguinte forma:

1. Definir o intervalo de horas para a previsão (y) e o número de variáveis independentes (x)
2. Criar dois ficheiros temporário onde serão guardados os casos de *fitness* de latitude e de longitude
3. Para cada registo (ponto geográfico) dos dados disponíveis:
    1. Ler a posição (latitude e longitude) e a hora de registo – este passa a ser o registo atual
    2. Para todos os registos após o registo atual, até ser criado um conjunto de x variáveis independentes ou ser atingido o final do ficheiro:
        1. Ler a posição (latitude e longitude) e a hora de registo do registo seguinte
        2. Caso o registo ocorra y horas após o registo atual:
            1. Gravar o registo e os x-1 registos seguintes como caso de *fitness* para latitude e outro para longitude, tendo os respetivos valores de latitude e longitude do registo atual como *target*
            2. Continuar a leitura do ficheiro, definindo um novo registo atual (o registo seguinte ao presente registo atual)
        3. Caso o registo ocorra y horas após o registo atual, mas não existirem x-1 registos seguintes, terminar a leitura do ficheiro
        4. Caso não exista um registo y horas após o atual, terminar a leitura do ficheiro
4. Para cada ficheiro temporário, dividir os registos em dois ficheiros – conjunto de teste (30% dos casos de *fitness* criados) e conjunto de treino (70% dos casos de *fitness* criados)
5. Para cada ficheiro criado, acrescentar o cabeçalho com o número de casos de *fitness* existentes e o número de variáveis independentes (x)

Após a criação dos ficheiros de *input*, os mesmos eram utilizados com input para o GPLAB, que produzia então os resultados finais, a serem comparados com os que já existiam anteriormente.

Embora não necessariamente neste formato, os dados nas bases de dados da Critical Software possuem os mesmos elementos e organização, por isso a transição entre fontes de dados no futuro iria apenas requerer um alteração no formato esperado dos dados de origem – o processamento dos mesmos seria idêntico.

6.1.3. Problemas e Limitações

Durante a iteração zero, o único problema registado foi o tempo necessário para que o MATLAB produzisse resultados, visto que, dada a natureza não determinística da PG, qualquer resultado teria ser estatisticamente significativo – sendo para tal executadas 30 execuções de cada configuração. Embora seja uma excelente ferramenta analítica, a certo ponto da iteração foi substituído por um algoritmo em C++, que embora não permita analisar tão detalhadamente os resultados e o seu processo de criação, representa uma redução significativa no tempo de desempenho da solução.

6.1.4. Resultados

No final desta iteração, foi determinado que os resultados produzidos pelo MATLAB e pelo algoritmo de C++ se assemelham qualitativamente aos anteriormente produzidos, confirmando a validade das configurações já definidas. Mais importante para a tese foi o correto desenvolvimento de um módulo de conversão de formato de dados para ficheiros de input aceitados pelo algoritmo de GS-GP, sendo este módulo reutilizado (embora com algumas adaptações) em iterações seguintes do trabalho de tese.

6.2. *ITERAÇÃO 1*

A iteração um é considerada como prova de conceito do trabalho de tese, tendo sido realizada na Critical Software, no Campus do Lumiar, durante o período de uma semana. Pretende demonstrar a funcionalidade básica do software, providenciando à equipa da Critical Software a possibilidade de realizar uma avaliação preliminar dos resultados no sistema Oversee, criando também uma base de trabalho para iterações seguintes.

6.2.1. Objetivos

Os objetivos da iteração um foram os seguintes:

- Retirar dados da base de dados da Critical Software de maneira a construir os conjuntos de teste e treino necessários à operação do algoritmo de GS-GP;
- Ler os resultados produzidos pelo algoritmo de GS-GP e guarda-los na base de dados da Critical Software, produzindo um registo legível pelo sistema Oversee, de modo a que o mesmo possa representar a posição prevista no mapa usado para controlo;

6.2.2. Procedimentos

O desenvolvimento da primeira versão da aplicação desenvolvida para a Critical Software tem por objetivos, como visto acima, implementar apenas as funcionalidades básicas. Tal deve-se ao tempo limitado de desenvolvimento disponibilizado na empresa, onde existe a possibilidade de acesso às bases de dados (dada a natureza dos dados, o acesso fora da empresa não é permitido). Como tal, as funcionalidades abrangidas por esta iteração dizem maioritariamente respeito à leitura e escrita de registos nas bases de dados.

Funcionalmente, cada execução da aplicação diz respeito a apenas uma previsão para uma embarcação – ou seja, duas execuções do algoritmo de GS-GP, uma para a previsão da latitude e outra para a longitude. Adicionalmente, cada execução considera apenas um intervalo de tempo (ou seja, previsões para a mesma embarcação para intervalos de tempo diferentes requerem execuções separadas).

Considerando o parágrafo anterior, a execução inicia-se com a recolha dos dados referentes a uma única embarcação. Através do identificador única da embarcação, presente na tabela 'Perfil da Embarcação', é definido se a mesma existe. Em caso afirmativo, acede-se à tabela 'Histórico de Posições', levantando até 30.000 posições registadas – este limite é imposto visto que o histórico de uma embarcação pode conter milhares de registos antigos, pouco relevantes pois: para embarcações com rotas regulares, os 30.000 registos são geralmente suficientes para o algoritmo de GS-GP

determinar o padrão da rota; para embarcações com rotas irregulares, os registos antigos podem criar ruído na análise, acrescentando dados que não acrescentam informação à previsão, podendo até dificultar a mesma (alterações irregulares no final da rota podem ser "ocultadas" por um excesso de informação anterior). Nesta versão da aplicação, os registos são formatados enquanto são lidos da base de dados, colocando-os num formato esperado pelo módulo desenvolvido na iteração zero, sendo provisoriamente guardados num ficheiro de texto simples (.txt).

Tendo os dados relativos à embarcação, os mesmos são submetidos ao módulo desenvolvido na iteração zero. Terminada a sua execução, os conjuntos de teste e treino criados são submetidos ao algoritmo de GS-GP. Nesta iteração o processo é realizado manualmente, tendo o algoritmo de ser executado ambas as vezes pelo utilizador – tal medida foi implementada para permitir um controlo e verificação dos ficheiros intermédios durante o desenvolvimento. Após a execução do algoritmo, para a longitude e para a latitude, a aplicação retoma a normal execução.

Finalmente, a aplicação cria uma embarcação provisória na base de dados à qual será associada a previsão realizada (neste iteração, é usada sempre a mesma embarcação provisória). A posição é gravada como posição atual, sem considerar dados como velocidade, direção, etc.

De um modo resumido, pode considerar-se o algoritmo seguinte para esta iteração da aplicação:

1. Inserir identificador da embarcação pretendida
2. Caso a embarcação exista, proceder ao levantamento de até 30.000 das posições mais recentes da embarcação
3. Gerar conjuntos de teste e treino para latitude e longitude
4. Executar (manualmente) o algoritmo de GS-GP, para latitude e longitude (a execução do programa fica suspensa até input do utilizador)
5. Retomada a execução do programa, leitura dos resultados produzidos pelo algoritmo de GS-GP
6. Criação de uma embarcação provisória na base de dados (tabela Perfil da Embarcação)
7. Gravar como posição atual da embarcação provisória criada no passo 6 a posição prevista pelo algoritmo de GS-GP (tabela Posição Atual)

Seguidamente, o ponto pode ser visualizado no sistema Oversee, permitindo representar graficamente a previsão criada num mapa do território marítimo.

6.2.3. Problemas e Limitações

Na iteração um, o principal problema foi o tempo limitado disponibilizado para trabalhar na Critical Software. Numa primeira fase de desenvolvimento, não era possível proceder à otimização do código produzido. Como tal, e especialmente devido à natureza e intensidade computacional da PG, as execuções consumiam uma quantidade considerável de tempo. Como tal, os testes (especialmente aqueles que necessitavam de resultados), eram excessivamente demorados, dificultando o rápido desenvolvimento do programa.

6.2.4. Resultados

No final da iteração um, ambos os objetivos definidos foram cumpridos. O programa desenvolvido permitia selecionar uma embarcação da base de dados da Critical Software e aceder às posições da mesma, usando-as para criar os ficheiros necessários à execução do algoritmo de GS-GP. Executado o algoritmo, colocava nas bases de dados o ponto previsto, permitindo a sua visualização no sistema Oversee.

Os resultados produzidos foram aprovados pela equipa da Critical Software, permitindo a continuação dos trabalhos. Consideram-se para a seguinte iteração a necessidade de otimização da aplicação (diminuindo o tempo de execução, considerado pela Critical Software como excessivo), a automatização da execução do algoritmo de GS-GP, e a necessidade de permitir a previsão de mais do que uma posição futura por execução da aplicação.

6.3. *ITERAÇÃO 2*

O principal objetivo da iteração 2 foi aumentar o grau de desempenho da solução desenvolvida na iteração 1. Tal como verificado na iteração anterior, dado o âmbito das previsões – a atuação de recursos da Marinha Portuguesa no local e na hora certa – é verificado que para além da exatidão da previsão, a mesma também deve ser gerada rapidamente. Visto a iteração anterior ter validado a qualidade das previsões guardadas pela solução desenvolvida, a mesma deve agora atuar o mais depressa possível. A iteração 2 foi realizada na Critical Software, no Campus do Lumiar, durante o período de dois dias, sendo o segundo reservado para apresentação de resultados.

6.3.1. Objetivos

Os objetivos da iteração dois foram os seguintes:

- Otimizar a aplicação desenvolvida – ou seja, todos os processos para lá do algoritmo de GS-GP – a fim de diminuir o tempo total de execução;
- Automatizar a execução do algoritmo de GS-GP;
- Aumentar o número de posições previstas por execução, permitindo que uma só execução produza posições para diferentes intervalos de tempo;
- Dado que o algoritmo de GS-GP produz resultados para todo o conjunto de testes, permitir que os mesmos sejam acrescentados ao sistema Oversse, possibilitando uma validação da rota anteriormente verificada.

6.3.2. Procedimentos

A iteração dois pode ser dividida em dois grandes componentes, um de otimização e outro de novas funcionalidades.

Relativamente à otimização do código existente, a mesma diz respeito a todos os elementos para lá do algoritmo de GS-GP, que já recai para lá do âmbito da presente tese. Como tal, apenas a solução desenvolvida, a integração do algoritmo de GS-GP, é considerada.

Após análise do código produzido, foi definido como ponto de melhoria o módulo referente à leitura de registos da base de dados e criação de conjuntos de teste e treino. Ambos os módulos dependiam inicialmente de ficheiros – usando-os para guardar os registos lidos, as várias fases de transformação

dos dados e os ficheiros finais contendo os conjuntos de teste e treino. Os ficheiros referentes aos conjuntos de teste e treino são necessários à execução do algoritmo de GS-GP, pelo que não podem ser implementados de diferente modo – o algoritmo de GS-GP está criado para ser executado com recurso a ficheiros de texto. No entanto, os restantes ficheiros foram todos removidos, visto a criação e leitura dos mesmos consistirem passos com elevado custo computacional devido ao volume de dados usados pelo programa, tendo sido implementados de maneira a permitir um controlo da transformação dos dados. Na presente iteração, visto que os módulos de transformação de dados estarem confirmadamente a operar como pretendido, os ficheiros intermédios foram retirados, passando os dados a ser geridos através de um sistema de *arrays*. O método de transformação de dados (como referido em 6.1.2 e 6.2.2) é exatamente igual, sendo a única diferença a estrutura onde os dados são guardados. De tal modo, uma previsão realizada para 2 horas (utilizando cerca de 30.000 registos) que anteriormente demorava cerca de 10 minutos a ser concluída passou a demorar cerca de 3 minutos, registando-se uma melhoria do tempo de execução de 70%. A mesma melhoria manteve-se constante para outras configurações (var Capítulo 7.2.1).

Adicionalmente ao ponto acima, dado que deixa de existir necessidade de controlar a criação dos ficheiros de teste e treino, o algoritmo de GS-GP passa a ser executado automaticamente, deixando de ser necessária a sua ativação manual.

Relativamente a funcionalidades acrescentadas, o desenvolvimento focou-se em responder às necessidades levantadas durante a última avaliação da tese – nomeadamente, o aumento do número de pontos gerados por uma única execução.

A primeira funcionalidade a ser acrescentada foi a possibilidade de gerar várias previsões numa única execução. É definido um intervalo de tempo máximo, que representa a previsão mais distante no tempo a ser realizada. Adicionalmente, são realizadas previsões para todas as horas anteriores até ser atingida 1 hora após a última posição registada (a termo de exemplo, suponha-se que é definida como intervalo máximo um intervalo de 8 horas – serão realizadas previsões para 8, 7, 6, 5, 4, 3, 2 e 1 horas após a última posição registada). Deste modo é produzido uma "rota" prevista após a última posição registada. Devido o modo de operação do algoritmo de GS-GP, não é possível juntar estas previsões numa única execução do algoritmo, visto que os conjuntos de treino e teste, apesar de os dados base serem os mesmos, têm de ser configurados diferentemente para intervalos diferentes (como é possível observar pela estrutura definida no capítulo 5.1) – os conjuntos de treino e teste para um intervalo de 1 hora não são idênticos aos conjuntos de treino e teste para um intervalo de 12 horas, etc. Para tal, foi implementado um ciclo iniciado no maior intervalo de tempo definido e terminado num intervalo de 1 hora, sendo repetido o processo de preparação de dados, a execução do algoritmo de GS-GP e a gravação dos dados. De um modo simplificado, a execução do programa é repetida tantas vezes como são necessários intervalos de tempo únicos, como se pode ver no algoritmo abaixo:

1. Inserir identificador da embarcação pretendida
2. Definir o intervalo máximo de horas pretendido (*gap*)
3. Caso a embarcação exista, proceder ao levantamento de até 30.000 das posições mais recentes da embarcação
4. Criação de uma embarcação provisória na base de dados (tabela Perfil da Embarcação)
5. Para o valor atual de *gap,* até 1 (inclusive):
    1. Gerar conjuntos de teste e treino para latitude e longitude para o *gap* definido
    2. Executar o algoritmo de GS-GP, para latitude e longitude
    3. Leitura dos resultados produzidos pelo algoritmo de GS-GP e gravação dos mesmo como:
        1. Se o *gap* atual corresponder ao valor inicialmente definido, guardar a posição como posição atual (tabela Posição Atual)
        2. Se o *gap* atual não corresponder ao valor inicialmente definido, guardar a posição no histórico de posições (tabela Histórico de Posições)
    4. Subtrair 1 ao valor do *gap* atual

Naturalmente, as previsões adicionais fazem aumentar o tempo de execução da solução, sendo este incremento quase linear (se uma previsão demora 3 minutos, duas previsões demorarão 6, etc.). Adicionalmente, no algoritmo acima é possível também verificar que o algoritmo de GS-GP é agora executado automaticamente, visto já não ser necessário controlar os ficheiros intermédios durante a execução. Deste modo, o utilizador apenas necessita de definir os parâmetros iniciais, não sendo necessárias outras interações durante a execução.

A última funcionalidade acrescentada durante esta execução é a possibilidade de ignorar as previsões relativas a posições futuras da embarcação em questão e adicionar ao mapa do sistema Oversee as previsões relativas a posições já antes registadas pelo sistema. Deste modo, é possível comparar a rota tomada pela embarcação (os pontos realmente registados) com a rota que o algoritmo de GS-GP previu que a mesmo tomasse com base no seu histórico de posições (os pontos previstos). Tal é possível pois o algoritmo de GS-GP produz previsões para todos os pontos do conjunto de teste, sendo o primeiro a previsão da posição no futuro e todos os outros referentes a posições já existentes. Assim sendo, é possível guardar a primeira posição (a previsão futura) como posição atual da embarcação e as restantes posições como histórico da mesma. Para realizar esta operação, não é possível realizar mais do que uma previsão futura. O âmbito desta funcionalidade é verificar se a rota tomada pela embarcação corresponde àquilo previsto pelo algoritmo de GS-GP – caso existam grandes diferenças, tal pode assinalar um possível problema com a embarcação.

6.3.3. Problemas e Limitações

Na iteração dois, foi registada a mesma limitação da iteração anterior, visto que o tempo disponível para trabalhar na Critical Software foi limitado (efetivamente, um dia). Como tal, e embora o tempo de execução do algoritmo tenha sido reduzido, o desenvolvimento de funções acaba por ter uma maior enfâse na entrega das mesmas do que na sua otimização, tendo a mesma de ser realizada em iterações seguintes.

6.3.4. Resultados

No final da iteração dois, é possível afirmar que os objetivos foram cumpridos – as novas funções implementadas funcionam como pretendido e foi verificada uma melhoria no tempo de execução, a melhoria mais importante apontada pela Critical Software na iteração anterior.

Os resultados foram novamente aprovados pela equipa da Critical Software, tendo no entanto sido levantados alguns pontos de melhoria. No âmbito da melhoria adicional do tempo de execução da solução, é pretendido verificar se é possível executar o algoritmo de GS-GP apenas uma vez por execução, mantendo no entanto a produção de várias previsões. Adicionalmente, é necessário também proceder à parametrização ótima do algoritmo, identificando quais os parâmetros que criam a melhor combinação de qualidade de soluções produzidas e tempo de execução.

6.4. *ITERAÇÃO 3*

A iteração 3, tal como a anterior iteração, pretende aumentar o desempenho da aplicação desenvolvida. Dado que a presente iteração não foi realizada na Critical Software, foi maioritariamente utilizada para a parametrização do algoritmo de GS-GP, bem como da aplicação desenvolvida – processo que não foi possível realizar anteriormente devido à necessidade de capitalizar o tempo disponível nas instalações da Critical Software para desenvolver e testar as funcionalidades da aplicação, sendo que o processo de parametrização necessita de tempo para produzir resultados que permitam retirar conclusões relevantes. A presente iteração não foi portanto realizada num ambiente *live*, fazendo uso de dados de teste já existentes.

O presente capítulo centra-se na descrição da iteração. É no entanto importante notar que a mesma foi desenvolvida com a Fase de Investigação (descrita no Capítulo 7), usando as conclusões da mesma para proceder à parametrização.

6.4.1. Objetivos

Os objetivos da iteração três foram os seguintes:

- Definir quais os melhores parâmetros de funcionamento para a aplicação desenvolvida, equilibrando as necessidades de qualidade das soluções e tempo de execução da aplicação;
- Explorar a possibilidade de executar o algoritmo de GS-GP apenas uma vez para gerar várias previsões simultaneamente, tal como pedido na iteração anterior;
- Otimizar a aplicação desenvolvida e documentar a mesma;
- Num projeto paralelo à presente tese, aplicar a aplicação desenvolvida a um novo conjunto de dados de teste, a fim de comparar os resultados obtidos com outras técnicas de PG (descrito no Capítulo 7.3).

6.4.2. Procedimentos

A presente iteração assume em grande parte um carácter exploratório, onde são testadas várias possíveis configurações para a aplicação. Estas afetam maioritariamente os parâmetros do algoritmo de GS-GP (nomeadamente, o número de elementos por registo, profundidade inicial, a versão do algoritmo utilizado, o tipo de ficheiros de input utilizados), seguindo recomendações dadas pela equipa da Critical Software e do projeto. Adicionalmente, é também procedido à otimização e

documentação da aplicação produzida para a mesma poder ser entregue e futuramente trabalhada. Este processo inclui a integração num único programa de todas as funcionalidades desenvolvidas anteriormente e durante a presente iteração, e a revisão do código produzido, a fim de tornar a manutenção e utilização da aplicação mais acessível no futuro. Como tal, podemos dividir os trabalhos da presente iteração em três categorias: parametrização; desenvolvimento e revisão; documentação.

6.4.2.1. Parametrização

Um dos focos da presente iteração foi, a pedido da Critical Software, determinar se seria possível produzir várias previsões, para diferentes intervalos de tempo, através de uma única execução do algoritmo de GS-GP (em vez do modelo implementado na iteração anterior, onde cada previsão para um intervalo de tempo implica uma execução separada). A proposta apresenta várias possíveis vantagens:

- A possibilidade de desenvolver e aplicar um único modelo para todas as previsões, possivelmente aumentando a sua coesão como conjunto (na iteração anterior, cada previsão possui o seu próprio modelo, pelo que podem existir diferenças marcadas entre eles, dada a natureza não determinística da PG);
- A possibilidade de realizar várias previsões numa única execução pode levar a melhores tempos de execução, embora implique a criação de ficheiros de *input* mais complexos (dado que consideram vários intervalos de tempo simultaneamente em vez de um único);
- A junção de vários intervalos de tempos numa única execução leva à criação de ficheiros de *input* com mais dados – dado que a PG pode produzir melhores soluções se os dados forem abundantes, descrevendo em maior detalhe o fenómeno estudado, existe a possibilidade de serem criados melhores modelos de previsão.

Este novo modelo de *input* encontra-se descrito detalhadamente no Capítulo 7.1.1, mas resumidamente o mesmo junta num único ficheiro de *input* todos os intervalos de tempos que antes implicariam execuções separadas. Para distinguir os vários registos, são acrescentadas duas variáveis: Tempo (que indica o tempo em minutos do dia em que a posição considerada foi registada) e Intervalo (que indica em horas qual o intervalo de tempo que o registo representa).

Tal como descrito no Capítulo 5.3, a presente tese utiliza com parâmetros base para o algoritmo de GS-GP os parâmetros utilizados por Vanneschi et al. (Vanneschi, 2015). No entanto, considerando o novo modelo de *input* desenvolvido, alguns parâmetros foram revistos, a fim de determinar se continuam a revelar a melhor relação de tempo de execução para qualidade das soluções geradas. Do mesmo modo, foi dada uma maior ênfase ao modo como os vários parâmetros afetam o tempo de execução do algoritmo, a fim de encontrar um equilíbrio aceitável entre tempo de execução e a qualidade das soluções geradas (visto que o tempo de execução representa uma prioridade no presente trabalho de tese). A fim de melhor organizar o presente documento, o processo de testes e os resultados deles decorrentes encontram-se documentados no Capítulo 7, sendo abaixo listados apenas os parâmetros reavaliados e as conclusões decorrentes do estudo dos mesmos:

- Modelo de *input* (ver Capítulo 7.1) – após a implementação do novo modelo de *input* descrito acima, foi necessário avaliar o seu desempenho comparativamente ao modelo anteriormente utilizado. Foi determinado que o novo modelo possui um desempenho

semelhante ao modelo antes implementado, pelo que foi acrescentado como opção à solução desenvolvida, dando ao utilizador a hipótese de escolher qual o modelo a utilizar para realizar as previsões.

- Número de elementos por registo (ver Capítulo 7.2.1) – o número de elementos (ou posições) por registo do ficheiro de input possui um grande impacto no desempenho do algoritmo de GS-GP (geralmente, um maior número de registos leva a melhores resultados, mas tempos de execução mais longos). Foi determinado que o número ideal de elementos por registo é de 100, apresentando um bom equilíbrio entre desempenho e tempo de execução.
- Versão do algoritmo de GS-GP (ver Capítulo 7.2.2) – durante o desenvolvimento da presente tese, foi desenvolvida uma nova versão do algoritmo de GS-GP utilizado. Foram realizados testes comparativos com a versão até então utilizada, sendo determinado que o novo modelo produz melhores resultados e apresenta tempos de execução menores, sendo portanto implementado na versão final da aplicação desenvolvida.
- Profundidade inicial das soluções – devido à implementação de um novo modelo de *input*, foi reavaliada a profundidade inicial das soluções geradas pelo algoritmo, a fim de determinar se o mesmo poderia beneficiar de um valor menor. Foi determinado que o valor inicialmente definido por Vanneschi et al. (Vanneschi, 2015), 6, continua a registar melhores resultados em tempos aceitáveis, não sendo por isso alterado.

6.4.2.2. Desenvolvimento e Revisão

Como visto no capítulo anterior, a presente iteração pretende acrescentar um novo modelo de *input* à aplicação. Para tal, são realizadas algumas alterações ao algoritmo de modo a ser possível produzir os ficheiros de *input* necessários. Nomeadamente, o algoritmo de GS-GP é executado apenas uma vez para cada coordenada para qualquer número de previsões pretendidas (em vez de ser executado um igual número de vezes por coordenada). Para tal, o processo de criação de ficheiros é realizado apenas uma vez, mas considera um número muito maior de registos, visto que está essencialmente a agrupar num único par de ficheiros todos os ficheiros de *input* que antes seriam gerados (ou seja, um par de ficheiros para cada intervalo, existindo agora um único par de ficheiro com todos os intervalos). Veja-se o algoritmo abaixo:

1. Inserir identificador da embarcação pretendida
2. Definir o intervalo máximo de horas pretendido (*y*) e o número de variáveis independentes (*x*)
3. Caso a embarcação exista, proceder ao levantamento de até 30.000 das posições mais recentes da embarcação
4. Criação de uma embarcação provisória na base de dados (tabela Perfil da Embarcação)
5. Para gerar os conjuntos de teste e treino para latitude e longitude e *i* igual a 1, enquanto *i* for menor ou igual a *y*:
    1. Para cada registo (ponto geográfico) dos dados disponíveis:
        1. Ler a posição (latitude e longitude) e a hora de registo – este passa a ser o registo *T*
        2. Para todos os registos após o registo *T*, até ser criado um conjunto de *x* variáveis independentes ou ser atingido o final do ficheiro:
            1. Ler a posição (latitude e longitude) e a hora de registo do registo seguinte – denominado de registo *C*
            2. Caso o registo *C* ocorra *y* horas após o registo atual:
                1. Gravar o registo *C* e os *x*-1 registos seguintes como caso de *fitness* para latitude e outro para longitude, seguidos pela hora de registo do registo *T* (em minutos do dia) e pelo valor do intervalo entre o registo *T* e o registo *C* (em horas), tendo os respetivos valores de latitude e longitude do registo *T* como *target*
                2. Continuar a leitura do ficheiro, definindo um novo registo atual (o registo seguinte ao presente registo *T*)
            3. Caso o registo *C* ocorra *y* horas após o registo *T*, mas não existirem *x*-1 registos seguintes, terminar a leitura do ficheiro
            4. Caso não exista um registo *y* horas após o registo *T*, terminar a leitura do ficheiro
    2. Incrementar *i* por 1
6. Acrescentar ao início dos conjuntos de teste e treino *y* registos com os últimos *x* pontos conhecidos como variáveis independentes, a hora esperada do *target* (tempo do primeiro registo + intervalo entre o primeiro registo e o target em horas), o intervalo desejado (o primeiro destes registos terá um intervalo de 1 hora, o segundo 2 horas, etc. até ser atingido um intervalo de *y* horas) e um target de 0 (este target não é conhecido, servindo apenas de *placeholder*) – estes registos adicionais dizem respeitos às previsões que serão geradas
7. Executar o algoritmo de GS-GP, para latitude e longitude
8. Selecionar os *y* primeiros resultados gerados pelo algoritmo, gravando o último como posição atual (tabela Posição Atual) e os restantes como posições no histórico de posições (tabela Histórico de Posições)

Usando o algoritmo acima, é possível gerar um qualquer número de previsões com uma única execução do algoritmo de GS-GP para cada coordenada.

A presente iteração acrescentou também uma outra funcionalidade à solução desenvolvida: a possibilidade de gerar a previsão de posições passadas, devido ao interesse suscitado pela Critical Software nesta possibilidade. Este modo de operação não realiza a previsão de posições futuras, mas sim de posições já conhecidas (ou seja, posições onde a embarcação já esteve). Permite portanto comparar a rota que, com base na sua deslocação anterior, o algoritmo de GS-GP prevê que a embarcação deveria ter tomado, com a rota que a embarcação realmente tomou. Isto auxilia à deteção de anormalidades na rota de embarcações, podendo ser sinais de atividade que mereça intervenção da Marinha (por exemplo, uma embarcação que decida não desligar o AIS para não levantar suspeitas sobre intenções ilícitas, mas que as realize na mesma). Este modo de operação funciona de modo igual ao modo tradicional descrito na iteração 2, com a diferença de aceitar apenas um único intervalo de tempo (ou seja, o algoritmo de GS-GP é executado apenas uma vez por coordenada). No final da execução, para além da posição prevista ser guardada como posição atual (tabela Posição Atual), todos os restante resultados produzidos para os dados do ficheiro de teste (até então não utilizados) são guardados como posições no histórico de posições (tabela Histórico de Posições). Deste modo, é gravada uma rota prevista para posições passadas, podendo depois ser comparada com a real rota da embarcação.

Finalmente, a presente iteração passou também pela revisão do código criado, e a implementação simultânea dos três modos de operação desenvolvidos na aplicação final: o modelo tradicional (uma execução por coordenada do algoritmo de GS-GP por cada intervalo de tempo); o modelo novo (uma única execução do algoritmo de GS-GP por coordenada para qualquer número de previsões); o modelo de rota passada (previsão de posições já registadas). Comparativamente à iteração anterior, podem ser verificadas as seguintes principais diferenças (para além de correções e otimizações menores):

- Implementação de 3 formas de funcionamento (descritas acima);
- Criação de uma nova embarcação antes da execução das previsões, em vez de após, simplificando o processo de gravação de dados;
- Levantamento de posições relativas à embarcação antes da execução das previsões, em vez de durante. Para o modelo tradicional, isto significa que o levantamento é realizado apenas uma vez, em vez de ser feito uma vez por cada intervalo de tempo;
- Para o modelo tradicional, implementar um parâmetro que permite ao utilizador definir “saltos” no intervalo a considerar para as previsões (quando há mais do que um). Ou seja, anteriormente o processo calculava previsões para todas as horas entre a definida e 1 (ou seja, para um intervalo máximo de 5, seriam calculadas previsões par 5, 4, 3, 2 e 1 hora de intervalo); agora é possível definir o ritmo a que o intervalo diminui (por exemplo, calcular apenas para 5, 3 e 1 hora de intervalo, definindo um salto de 2 horas);
- Diminuição do número de parâmetros que o utilizador tem de configurar para executar as previsões, deixando apenas 7: identificador da embarcação; número máximo de registos a utilizar; intervalo máximo da previsão; número de elementos por registo; tipo de execução (tradicional, nova ou rota passada); salto do intervalo (para o modelo tradicional apenas); sequencialidade (para o modelo novo apenas, se definido como verdadeiro, o modelo novo realiza previsões para todos os intervalos de tempo entre o intervalo máximo definido e 1). Os restantes parâmetros são definidos automaticamente tendo os 7 acima referidos como base.

6.4.2.3. Documentação

Serve o presente capítulo para apresentar uma descrição funcional da aplicação desenvolvida, a fim de facilitar o uso e entendimento da mesma. Documentação mais tecnicamente detalhada está presente no próprio código, sendo o presente capítulo uma descrição que visa apenas facilitar o uso da solução desenvolvida.

Tal como descrito no Capítulo 6.4.2.2, o número de variáveis configuráveis foi reduzido para 7, sendo que este número poderá ser futuramente mais reduzido se for desejado retirar a possibilidade de alterar fatores como o número de registos da base de dados a usar ou o número de elementos por registo dos ficheiros de teste e treino (cujo valor já foi estudado no presente trabalho de tese, como é possível ver no Capítulo 7.2.1). Encontra-se abaixo a lista de variáveis disponíveis ao utilizador:

- vessel_id – *integer*; o identificador único da embarcação cujas posições se desejam estudar. Este identificador é o mesmo identificador único presente na tabela "Perfil da Embarcação";
- num_Of_Records – *integer*; o número máximo de registos levantados da base de dados para formar os conjuntos de teste e treino. Maiores números podem aumentar a qualidade da solução, aumentando o tempo de execução devido à maior quantidade de registos a processar. Por defeito, esta variável encontra-se definida para 30.000;
- hourGap – *integer*; intervalo máximo de tempo, em horas, para o qual será realizada uma previsão (ou seja, para quantas horas após a última posição conhecida será prevista uma posição). Previsões seguintes, se desejado, serão sempre menores que este valor. Valores inferiores a 1 não são considerados válidos, não produzindo quaisquer resultados;
- numOfPoints – *integer*; número de elementos utilizados em cada registo dos ficheiros de teste e treino. Maiores números poderão levar a melhores resultados, aumentando o tempo de execução. Valores inferiores a 1 impossibilitam a execução do algoritmo de GS-GP. Por defeito definido para 100 elementos (ver Capítulo 7.2.1);
- exec_Type – *integer*; define o tipo de análise a realizar. São definido 3 tipos de execução (descritas em maior detalhe no Capítulo 6.4.2.2), usando os valores abaixo para selecionar a execução correspondente (caso inserido um valor que não 1, 2 ou 3, o programa irá executar por defeito a análise 1):
    - 1: Análise segundo o modelo tradicional de *input* (sem dados de Tempo e Intervalo), onde o algoritmo de GS-GP é executado um número de vezes igual ao total de posições pretendidas;
    - 2: Análise segundo o "novo" modelo de *input* (com dados de Tempo e Intervalo), onde o algoritmo de GS-GP é executado uma única vez para qualquer número de previsões;
    - 3: Análise de posições passadas, onde não é realizada qualquer previsão para o futuro, mas é recalculado o trajeto recente da embarcação segundo um modelo baseado na sua deslocação até à última posição registada.
- prediction_gap – *integer*; apenas aplicável caso se esteja a realizar uma análise de tipo 1. Define a diferença entre intervalos de tempo das várias previsões realizadas (ou seja, para um *gap* de 5, um *prediction_gap* de 1 resulta na previsão dos intervalos de 5, 4, 3, 2 e 1 horas; um *prediction_gap* de 2 faz com que se realizem apenas previsões para 5, 3 e 1 horas). Se o valor definido for inferior a 1, o mesmo é colocado por defeito a 1;

- sequential – *interger*; apenas aplicável caso se esteja a realizar uma análise de tipo 2. Caso *true*, é realizada uma análise para todas as horas entre *gap* e 1; caso *false*, é realizada uma análise apenas para o intervalo definido em *gap*.

Adicionalmente, as configurações relativas à base de dados estão presentes no pacote *utils*, classe DataBaseUtils. É importante referir que, dada a base de dados da Critical Software, a solução foi desenhada tendo em vista ser utilizada com PostgreSQL.

Finalmente, a solução encontra-se totalmente contida na diretoria da aplicação. Dadas as necessidades de fornecer ficheiros de treino e teste ao algoritmo de GS-GP, não é recomendado que a estrutura de diretorias seja alterada. O ficheiro relativo às configurações do algoritmo de GS-GP (descritas no capítulo 5.3) encontra-se na diretoria "GSGP_LS": configuration.ini. É também nesta diretoria que são gerados os ficheiros de teste, treino e de resultados, sendo que a solução os deverá apagar após a execução. Finalmente, relembra-se que o algoritmo de GS-GP deve ser compilado antes de executado pela primeira vez (ver ficheiro "set up.txt" presente na mesma diretoria).

6.4.3. Problemas e Limitações

Na iteração 3, ao contrário das iterações anteriores, não foi possível realizar o desenvolvimento na Critical Software. Como tal, não foi possível acompanhar o desenvolvimento da solução produzida com testes num ambiente *live*. Em contrapartida, foi possível realizar testes com durações mais elevadas, anteriormente impraticáveis devido ao curto espaço de tempo disponível para implementação de funções. No entanto, é também necessário realçar que, dada a necessidade de obter resultados estatísticamente relevantes associada aos tempos de execução de PG, foi necessário realizar uma escolha relativamente a quais os parâmetros a a testar, não sendo possível realizar testes à totalidade dos parâmetros. Foram para tal estudados apenas os que foram considerados como plausíveis de terem um maior impacto na qualidade ou no tempo de execução da solução, sendo os restantes deixados como foram definidos em projetos anteriores por Vanneschi et al. (Vanneschi, 2015).

6.4.4. Resultados

No final da iteração 3, consideram-se os objetivos definidos para a mesma como cumpridos. As funcionalidades pedidas encontram-se implementadas na solução, e a mesma encontra-se parametrizada de modo a permitir um melhor desempenho considerando os fatores de tempo de execução e qualidade dos resultados produzidos. Consideram-se o tempo necessário para gerar uma solução bem como a sua qualidade aceitáveis pelos padrões discutidos com a Critical Software. Adicionalmente, o código da solução encontra-se novamente otimizado e documentado, permitindo a entrega da solução à Critical Software para testes (e possível desenvolvimento futuro).

Adicionalmente, a investigação paralela ao desenvolvimento comprova a qualidade do desempenho do algoritmo de GS-GP, podendo os resultados da mesma ser vistos no Capítulo 7.3.

7. FASE DE INVESTIGAÇÃO

A fase de investigação foi realizada em paralelo com a iteração 3 da Fase de Implementação. Destina-se a definir quais as configurações ótimas para o algoritmo de GS-GP, levando à otimização do tempo e qualidade da execução do mesmo. Tal otimização inclui a definição dos parâmetros a utilizar, bem como testes relativos a uma nova versão do algoritmo de GS-GP e um novo formato de *input*. Adicionalmente, são também apresentados resultados de um estudo desenvolvido em paralelo, sendo os mesmos demonstrativos da qualidade das soluções produzidas pelo algoritmo de GS-GP.

Embora realizada em paralelo com a iteração 3, a Fase de Investigação é considerada um capítulo diferente do presente documento devido à sua diferente natureza: enquanto o capítulo 6 é maioritariamente referente à descrição da implementação, o presente capítulo adota um cariz mais exploratório, apresentando diferentes testes e os seus resultados, devendo ser visto como uma base para as decisões relativas às configurações realizadas na iteração 3 da Fase de Implementação.

Os dados foram obtidos com uma versão de teste da solução desenvolvida, feita para trabalhar com dados de teste, visto não ser possível aceder às bases de dados da Critical Software nesta fase da tese. Os dados de teste diz respeito a uma única embarcação, de rota regular entre dois portos. Apenas no Capítulo 7.3 são utilizados dados referentes a outras embarcações, sendo assinalados como tal.

7.1. *NOVO MODELO DE INPUT PARA O ALGORITMO GS-GP*

Um dos requisitos levantados pela Critical Software no final da iteração 2 (Capítulo 6.3.4) foi de explorar a possibilidade de o algoritmo de GS-GP ser executado apenas uma vez por execução, gerando no entanto várias previsões (para este fim, no final da iteração 2 o algoritmo teria de ser executado um número de vezes igual ao número pretendido de previsões). O presente capítulo 7.1 descreve a solução encontrada, apresentando também resultados relativos à sua comparação com o tradicional método de obtenção de resultados. A solução encontrada passa pela alteração dos conjuntos de teste e treino utilizados para a execução do algoritmo de GS-GP.

7.1.1. Descrição do Novo Modelo

O antigo modelo de *input* (o formato dos conjuntos de treino e teste utilizados pelo algoritmo de GS-GP), a partir de agora designado como modelo tradicional, consiste em, para a latitude e para a longitude, criar registos compostos por um determinado número n de posições e uma posição target, registada x horas após a primeira posição do registo em causa (como pode ser visto em detalhe no Capítulo 5.1). O número de posições por registo é constante para todos os registos – sendo um requisito do algoritmo de GS-GP. Como tal, cada execução lida apenas com um intervalo de tempo, podendo apenas produzir previsões de x horas para cada conjunto de n posições. Ou seja, limitando as previsões às últimas n posições registadas, é apenas possível fazer uma previsão relativa às mesma para um único intervalo de tempo – podendo apenas ser criado um ponto "futuro" (ou seja, um ponto que não existe nos dados iniciais, que o sistema desconhece devido à cessação das transmissões de AIS; por exemplo, "onde é que a embarcação deverá estar 1 hora após as transmissões serem interrompidas?"). Isto significa que caso sejam necessárias previsões para mais do que um intervalo de tempo ("Onde está a embarcação 1 hora após as transmissões serem

interrompidas? E 2 horas? E 12 horas?"), são necessárias várias execuções do algoritmo, utilizando diferentes ficheiros de teste e de treino em cada uma.

A termo de exemplo, considere-se que cada registo do conjunto de teste/treino tem 5 posições mais uma posição *target*. Considerando também 7 posições P_n distribuídas por 3 registos, como explicado no Capítulo 5.1, as posições estão ordenadas cronologicamente, sendo P_1 a mais recente e P_7 a mais antiga. Para cada registo, existe um *target* T, estando os mesmos também ordenados cronologicamente – T_1 o mais recente, T_3 o mais antigo. O conjunto descrito encontra-se representado abaixo, denominado conjunto A:

P_1	P_2	P_3	P_4	P_5	T_1
P_2	P_3	P_4	P_5	P_6	T_2
P_3	P_4	P_5	P_6	P_7	T_3

Assumindo que o conjunto A pretende prever posições para um intervalo de 1 hora, T_1 ocorreu 1 hora após P_1, T_2 1 hora após P_2 e T_3 1 hora após P_3. Ou seja, com o conjunto A, é possível realizar previsões para 1 hora após a embarcação ter terminado as suas transmissões AIS.

Suponha-se agora que é necessário saber também a posição da mesma embarcação 6 horas após a mesma ter terminado as suas transmissões AIS. O conjunto A apenas representa intervalos de 1 hora entre a primeira posição do registo e o *target*, pelo que não pode ser utilizado para realizar previsões a 6 horas. Para tal, é necessário alterar os *targets* de um intervalo de 1 hora para 6 horas com a primeira posição do registo – tal pode ser feito alterando tanto o conjunto de posições P ou o conjunto de *targets* T. De modo a simplificar o exemplo, assuma-se que se tem acesso a 3 pontos que ocorreram 6 horas após P_1, P_2 e P_3, respetivamente (sendo por isso mais recentes que T_1, T_2 e T_3). Assim sendo, são substituídos os *targets* de modo a serem representados intervalos de 6 horas em vez de 1, como se pode ver no conjunto abaixo, denominado conjunto B:

P_1	P_2	P_3	P_4	P_5	T_4
P_2	P_3	P_4	P_5	P_6	T_5
P_3	P_4	P_5	P_6	P_7	T_6

Embora as posições P do conjunto B sejam as mesmas, os *targets* mudaram, permitindo agora realizar previsões para 6 horas após a embarcação ter terminado as suas transmissões – sendo que, como se trata de um conjunto diferente, é necessário realizar uma nova execução do algoritmo de GS-GP para o novo *input*.

No entanto, o que é pretendido é explorar se é possível realizar o mesmo número de previsões numa única execução. Dado que não é possível alterar o funcionamento do algoritmo de GS-GP para aceitar mais do que dois ficheiros por execução, é necessário alterar os ficheiros em si, fazendo com que os mesmos representem mais do que um intervalo de tempo.

Para representar mais do que um intervalo, é necessário juntar os vários ficheiros gerados para diferentes intervalos de tempo. No entanto, simplesmente unir os ficheiros existentes não é produz resultados aceitáveis. Retomando o exemplo, veja-se abaixo a união dos conjuntos A e B, denominado como conjunto C:

P_1	P_2	P_3	P_4	P_5	T_1
P_2	P_3	P_4	P_5	P_6	T_2
P_3	P_4	P_5	P_6	P_7	T_3
P_1	P_2	P_3	P_4	P_5	T_4
P_2	P_3	P_4	P_5	P_6	T_5
P_3	P_4	P_5	P_6	P_7	T_6

O conjunto C não representa um bom conjunto de treino/teste pois existem registos com as mesmas posições ou variáveis independentes (1º e 4º registos, 2º e 5º, 3º e 6º) e *targets* ou variáveis dependentes diferentes. Dado que o objetivo do algoritmo de GS-GP é chegar a uma função capaz de descrever com exatidão o comportamento da embarcação em questão, ou seja, para n posições de *input*, prever exatamente qual a posição ocupada x horas após as posições em questão. Tal não é possível quando o mesmo conjunto de variáveis independentes possui dois resultados diferentes. Como tal, é necessário arranjar dados adicionais a fim de criar diferenciação entre os dois casos.

Aumentar a diferenciação entre registos, foram acrescentados dois novos dados:

- O Intervalo – representa quantas horas existem entre a primeira posição e o *target*;
- A Hora – representa a hora em que o *target* foi (ou será) registado, em minutos do dia.

Ao acrescentar os dados acima, é possível diferenciar entre dois registos anteriormente idênticos. A termo de exemplo, considerem-se abaixo os registos referentes a T_1 e T_4 do conjunto C, assumindo que P_1 ocorreu às 00:00 (meia-noite):

P_1	P_2	P_3	P_4	P_5	60	1	T_1
P_1	P_2	P_3	P_4	P_5	360	6	T_4

Ao acrescentar o Intervalo, facilmente se faz a distinção entre os dois registos antes idênticos, havendo uma nova variável independente que permite diferenciar os registos de resto idênticos. No entanto, foi determinado que reforçar a ideia de tempo presente no registo era importante, pelo que foi acrescentada a hora, em minutos, de registo do *target* (caso o *target* corresponda a um registo que ainda não exista, a hora do mesmo é calculada como *hora de registo do primeiro ponto + intervalo de tempo*). Tal foi feito porque mais um dado relativo a tempo permite dar ao algoritmo uma maior perceção da distinção entre os dois registos com iguais posições, não deixando apenas um dado relativo a tempo entre as posições (podendo atingir as centenas, se necessário), sendo também um fator adicional para marcar a diferença entre os dois registos – dado que os dados se

encontram todos juntos, é de grande importância que o algoritmo consiga "aprender" o impacto do tempo no modelo a criar. Adicionalmente, para embarcações com percursos regulares (por exemplo, barcos que realizam ligações horárias entre dois portos), torna-se um fator que mais facilmente permite ao algoritmo atingir o modelo que representa o percurso, logo, sendo mais fácil de identificar alterações significativas no mesmo (tendo em conta que tais embarcações costumam seguir horários relativamente regulares).

Finalmente, as previsões futuras são realizadas do mesmo modo que eram no modelo tradicional, utilizando as n posições mais recentes como variáveis independentes e tendo 0 como *target*. A única diferença a assinalar, para além dos dois dados adicionais (Intervalo e Hora), é o facto de cada ficheiro ter mais do que um registo destinado à previsão de posições futuras. Tal como antes, encontram-se todos no início do ficheiro, estando ordenados pelo seu intervalo de tempo, sendo o primeiro registo aquele com o maior intervalo.

7.1.2. Comparação com o Modelo Tradicional

A fim de definir a qualidade do novo modelo, foram realizados teste de maneira a recolher dados sobre o seu desempenho – qualidade das previsões, consistência, tempo de execução, etc. Dado que o mesmo é proposto como solução para ser necessário executar o algoritmo apenas uma vez, o mesmo é também comparado com resultados produzidos pelo modelo tradicional. Os testes foram realizados com 100 posições em cada registo, 100 gerações e uma população de 50 indivíduos (todos os parâmetros não referidos são iguais aos descritos no Capítulo 5.3). As distâncias apresentadas resultam do cálculo da distância entre o ponto real (ou seja, o ponto onde a embarcação estava realmente após o intervalo de tempo considerado) e o ponto previsto pelo algoritmo de GS-GP, sendo o cálculo realizado segundo a fórmula apresentada no Capítulo 5.4.

Com o modelo novo foram realizadas execuções para um intervalo de tempo de 2 horas, cada execução produzindo previsões para 1 e 2 horas após a última posição conhecida. O novo modelo utilizou um total de 17.526 registos, dividido em 5.504 registos no conjunto de teste e 12.022 registos no conjunto de treino.

Para cada execução do modelo novo, o modelo tradicional foi executado duas vezes, uma para um intervalo de tempo de 2 horas e outra para um intervalo de tempo de 1 hora, produzindo assim o mesmo total de duas previsões – como tal, considera-se como tempo total de execução a soma do tempo de execução das duas execuções (para 1 hora e para 2 horas). A execução do modelo tradicional para um intervalo de 1 hora utilizou 9.189 registos, dividido em 2.887 registos no conjunto de teste e 6.302 registos no conjunto de treino. A execução do modelo tradicional para um intervalo de 2 horas utilizou 8.170 registos, dividido em 2.451registos no conjunto de teste e 5.719 registos no conjunto de treino. O total de registos usados para gerar um conjunto de duas previsões é portanto de 17.359 registos.

Abaixo, na Tabela 4, são apresentados os resultados calculados após as 30 execuções consideradas. Todos os resultados estão apresentados em quilómetros, exceto o tempo médio de execução (hh:mm:ss).

	Novo		Tradicional	
	2 Horas	**1 Hora**	**2 Horas**	**1 Hora**
Desvio Padrão	10,37678	4,440331	11,1618	13,43465
Mediana	21,65023	8,097777	18,15391	4,097258
Média	22,268	8,550861	18,5226	6,954876
Erro Máximo	44,32204	18,10178	49,87939	76,15875
Erro Mínimo	4,17945	1,022028	2,102883	0,442755
Tempo Médio de Execução	00:04:14		00:03:54	

Tabela 7.1 – Resultados de Execução dos Modelos de Input, Novo e Tradicional

Como é possível ver na Tabela 4, ambos os modelos apresentam desempenhos semelhantes, embora o modelo tradicional tenda para melhores resultados. Em detalhe, tanto para 1 hora como para 2 horas, o modelo tradicional apresenta valores mais baixos que o modelo novo.

Embora para o erro mínimo registado, o modelo tradicional apresente também valores menores, a tendência inverte-se quando considerado o erro máximo – em ambos os casos, o modelo novo regista erros máximos menores que o modelo tradicional. É possível considerar o erro de 76km um resultado aberrante, e ignorando o mesmo o valor do desvio padrão desce para cerca de 3,11km – novamente melhor que o modelo novo. O mesmo pode ser dito para o intervalo de 2 horas, onde ignorando o erro máximo, não sendo um valor considerável aberrante tal como o erro máximo registado para o primeiro intervalo, se podem obter valores de desvio padrão mais baixos do que os do modelo novo (cerca de 9,6km).

Finalmente, o tempo de execução regista uma diferença média de 20 segundos, sendo o modelo tradicional o que possui a execução mais curta.

7.1.3. Conclusões

Após uma primeira observação dos resultados obtidos, é aconselhável recomendar a continuação da utilização do modelo tradicional de *input*, visto apresentar resultados e tempos de execução ligeiramente melhores que os do modelo novo. Deve-se notar no entanto que, embora os valores se apresentem melhores, a diferença entre os resultados dos dois modelo não é muito marcada, não podendo ser dito que o modelo novo fiquem muito atrás dos resultados do modelo tradicional.

No entanto, devem também ser levadas em conta as seguintes considerações:

- Em ambos os casos, o modelo tradicional registou um valor de erro máximo anormalmente elevado, que por si fizeram com que o modelo novo registasse melhores valores de desvio padrão do que o modelo tradicional. Dada a natureza não determinística da PG, é expectável a produção de resultados aberrantes, embora sejam também raros. No entanto, em 60 execuções do modelo tradicional, foram registados dois valores largamente diferentes dos restantes. O mesmo não ocorreu com o modelo novo. Embora não seja possível, com os dados disponíveis, não atribuir estes valores ao simples acaso, existe a possibilidade de que o modelo novo possam ter uma execução menos propensa à criação de valores aberrantes;
- O facto de o modelo novo produzir todas as previsões numa única execução faz com que todas sejam geradas a partir do mesmo modelo. O modelo tradicional gera um novo modelo para cada previsão, o que faz com que exista a possibilidade de que uma previsão seja gerada a partir de um modelo "bom" (com melhor desempenho) e outra previsão seja

gerada a partir de um modelo "mau" (com pior desempenho) – podendo resultar num resultado final (o conjunto de previsões) pouco consistente. O modelo novo usa um único modelo, pelo que o resultado final será consistente (podendo ser "bom" ou "mau", não existem no entanto pontos gerados por modelos de diferente qualidade);

- Dado que o modelo novo junta todos os intervalos de tempo numa única execução, são utilizados muito mais dados para a mesma. Isto justifica o maior tempo de execução comparativamente ao modelo tradicional. Dado que a PG tem maiores probabilidades de atuar melhor dispondo de um grande volume de dados (de qualidade), existe a possibilidade de o modelo novo ser capaz de atuar melhor que o modelo tradicional em casos onde os dados são mais escassos.

Em última instância, apesar de não apresentar resultados igualmente bons ao modelo tradicional, a diferença de desempenho entre os dois modelos não é suficiente para afirmar que o modelo novo apresenta um mau desempenho. Considerando juntamente os pontos acima levantados, o estudo do modelo novo será continuado durante o resto do presente trabalho de tese, sendo os teste seguintes realizados com o modelo novo, servindo também para produzir novos dados que permitam melhorar o modelo novo.

7.2. *PARAMETRIZAÇÃO*

Um dos elementos importantes para o desempenho do algoritmo é a parametrização do mesmo. Para o presente trabalho de tese, têm sido utilizadas configurações definidas por Vanneschi et al. (Vanneschi, 2015), que já produziram bons resultados em problemas semelhantes. No entanto, certos elementos da presente tese devem ser reavaliados a fim de ser encontrado um equilíbrio entre a qualidade do desempenho e o tempo de execução da solução desenvolvida. Qualquer parâmetro do Capítulo 5.3 não referido no presente capítulo fica inalterado face ao seu valor original. Para realizar os testes do presente capítulo é utilizado o mesmo conjunto de dados (17.526 registos, dividido em 5.504 registos no conjunto de teste e 12.022 registos no conjunto de treino).

7.2.1. Número de Elementos por Registo

Como já visto, o algoritmo de GS-GP necessita de um conjunto de teste e um conjunto de treino para ser executado, sendo cada conjunto constituído por um número de registos (ou linhas), cada um deles possuindo um número fixo de elementos, ou variáveis independentes (neste caso, posições geográficas), mais um *target*, ou variável dependente. O número de elementos tem um grande impacto na qualidade da solução produzida (basicamente, mais dados equivalem a mais informação), tendo também um impacto no tempo de execução (um maior volume de dados leva uma maior necessidade de computação dos mesmos). Para além do número de posições geográficas, o presente capítulo explora também a utilização dos dados extra do modelo novo definido no capítulo 7.1 – denominados de Intervalo (número de horas entre o *target* e o primeiro registo) e Tempo (hora, em minutos do dia, do *target*) – servindo também para determinar se o seu uso é benéfico ou prejudicial para o desempenho do modelo. Como tal, os dados a utilizar são os mesmo utilizados pelo modelo novo no capítulo 7.1, a única alteração entre as execuções deste capítulo são que dados estão presentes (não o formato dos mesmos) – ou seja, uma execução com *n* elementos dita com Intervalo e com Tempo tem *n+2* elementos por registo, enquanto uma execução com Tempo e sem Intervalo possui *n+1* elementos por registo, sendo de resto idênticas (número de registos, ordem dos mesmos, etc.).

Dos dados utilizados para realizar os testes, são definidas duas vertentes cujas configurações são alteradas – o número de elementos referentes a posições geográficas por registo (10, 20, 50, 100, 200 e 500 posições); e o número de elementos adicionais do modelo novo:

- Sem Intervalo; Sem Tempo – configuração de controlo, sem a presença de nenhum dos elementos adicionais nos registos (é de notar que esta configuração não é equivalente ao modelo tradicional do Capítulo 7.1);
- Intervalo; Sem Tempo – esta configuração contem apenas um elemento adicional por registo, correspondente ao intervalo de tempo, em horas, entre o *target* e o tempo de registo da primeira posição do registo em questão;
- Sem Intervalo; Tempo – esta configuração contem apenas um elemento adicional por registo, correspondente ao tempo de registo da posição *target*;
- Intervalo; Tempo – esta configuração contem ambos os elementos adicionais descritos acima nos seus registos.

Abaixo são apresentados os resultados relativos ao tempo de execução da solução desenvolvida, para as diferentes configurações de pontos e elementos adicionais.

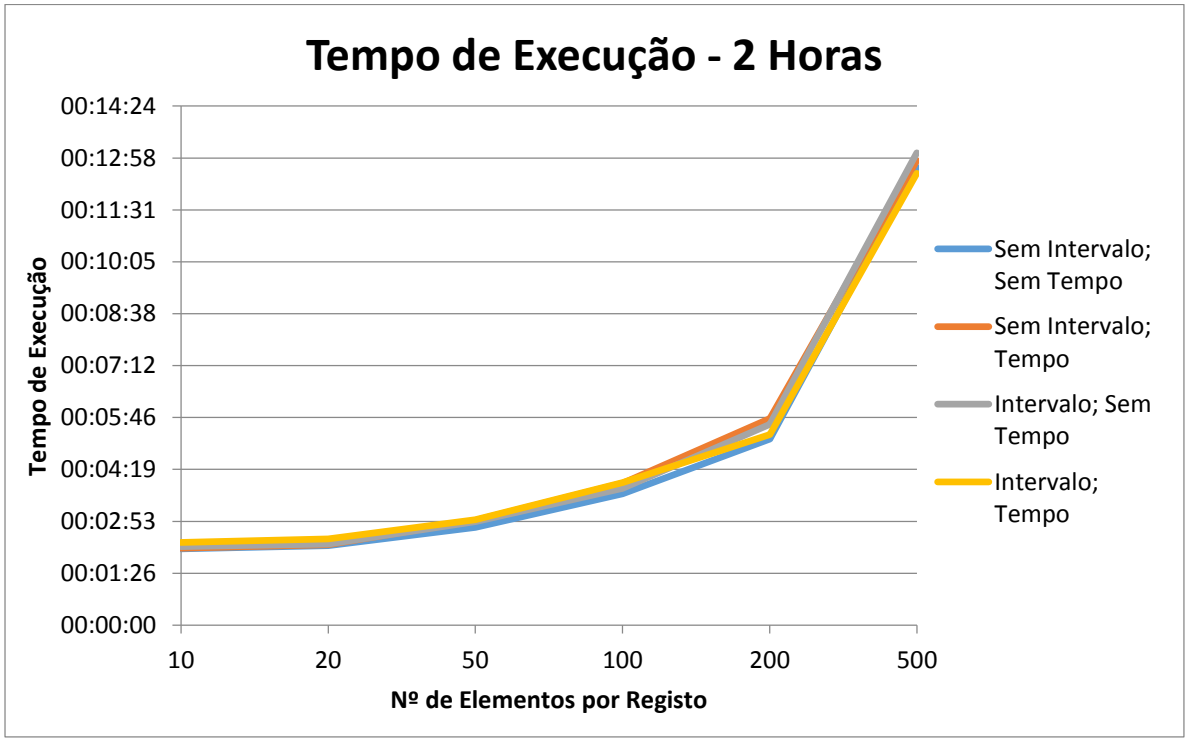


Figura 7.1 – Gráfico de tempos de execução para intervalo de 2 horas.

TEMPO DE EXECUÇÃO - 2 HORAS

	10	20	50	100	200	500
SEM INTERVALO; SEM TEMPO	00:02:07	00:02:12	00:02:43	00:03:39	00:05:10	00:12:42
SEM INTERVALO; TEMPO	00:02:08	00:02:14	00:02:54	00:03:56	00:05:44	00:12:53
INTERVALO; SEM TEMPO	00:02:11	00:02:15	00:02:52	00:03:48	00:05:34	00:13:06
INTERVALO; TEMPO	00:02:18	00:02:24	00:02:56	00:03:58	00:05:17	00:12:32

Tabela 7.2 – Tabela de tempos de execução para intervalo de 2 horas.

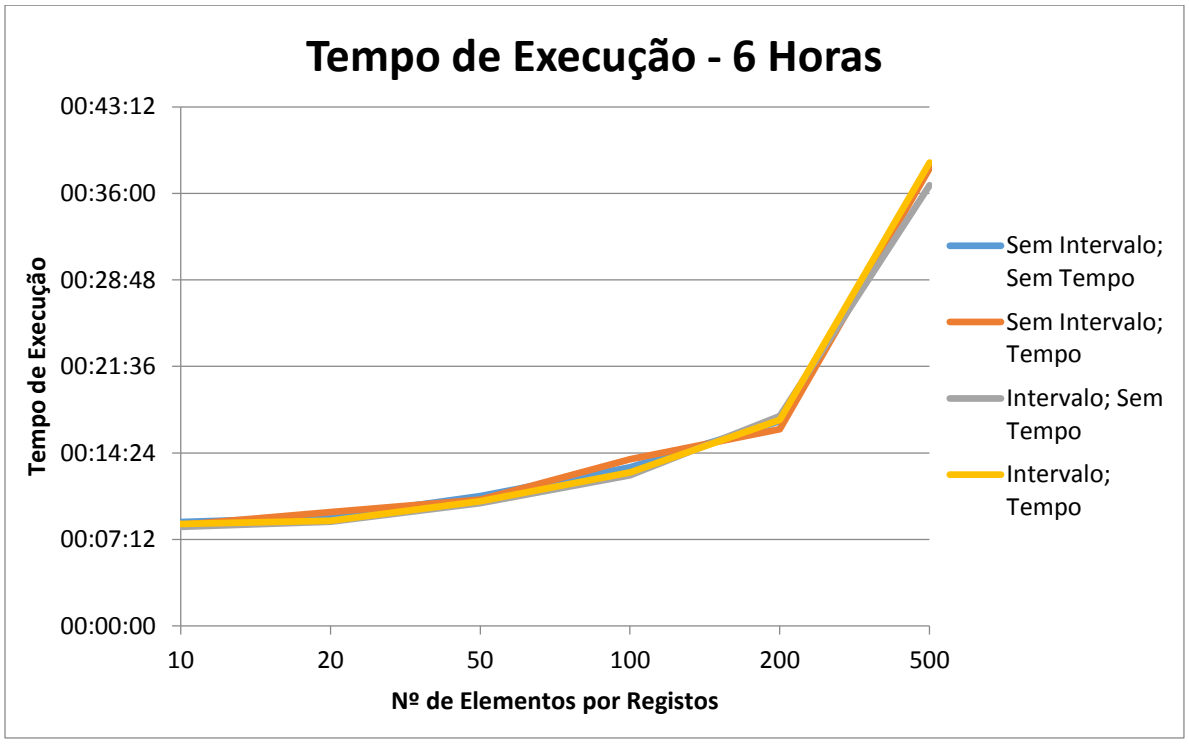


Figura 7.2 – Gráfico de tempos de execução para intervalo de 6 horas.

TEMPO DE EXECUÇÃO - 6 HORAS

	10	20	50	100	200	500
SEM INTERVALO; SEM TEMPO	00:08:39	00:09:03	00:10:48	00:13:13	00:17:02	00:38:24
SEM INTERVALO; TEMPO	00:08:24	00:09:28	00:10:30	00:13:52	00:16:23	00:38:06
INTERVALO; SEM TEMPO	00:08:15	00:08:41	00:10:14	00:12:32	00:17:27	00:36:41
INTERVALO; TEMPO	00:08:30	00:08:45	00:10:23	00:12:47	00:17:09	00:38:35

Tabela 7.3 – Tabela de tempos de execução para intervalo de 6 horas.

Como se pode ver pelas Figuras 12 e 13, a tendência é, tal como esperado, que um maior número de elementos por registo aumente o tempo de execução. No entanto, é possível ver que essa evolução é mais marcada à medida que são adicionados mais elementos aos registos – configurações de 10, 20 e 50 pontos possuem tempos relativamente próximos, enquanto as restantes configurações possuem tempos de execução com maiores diferenças entre si. Embora a escolha do número apropriado de elementos por registo dependa também da qualidade da soluções produzidas por cada configuração, é no entanto possível excluir imediatamente a configuração de 500 elementos por registo, sendo que regista em ambos os casos, durações proibitivamente longas, diferenciando-se grandemente das outras configurações. Do mesmo modo, a configuração referente a 200 elementos é também excluída, pois embora não tendo durações médias tão elevadas, apresenta uma diferença considerável da configuração de 100 elementos (quase equivalente à diferença de tempo entre a configuração de 100 elementos e a de 10).

Através da análise dos gráficos, é também possível verificar que a tendência de aumento do tempo de execução é seguida de maneira idêntica para as quatro configurações relativas a elementos adicionais. Através da análise da Tabela 5 e da Tabela 6, podemos ver que a diferença entre as várias configurações se mantem constante, não sendo as diferenças entre as mesmas afetadas significativamente pelo número de elementos por registo. Como tal, a melhor configuração relativa

ao modelo novo dependerá apenas da qualidade das soluções produzidas pelas diferentes configurações.

Abaixo encontram-se os resultados referentes à qualidade das previsões previstas pelas diferentes configurações (sendo designado como qualidade a distância, em quilómetros, entre a posição prevista e a posição real, sendo considerada melhor a diferença mais pequena). Foi escolhido como valor representativo da qualidade das várias execuções a mediana, visto permitir avaliar melhor a qualidade geral das configurações do que a média (sendo a média mais facilmente influenciada por valores anormais produzidos pelo algoritmo, situação possível dada a natureza não determinística da PG).

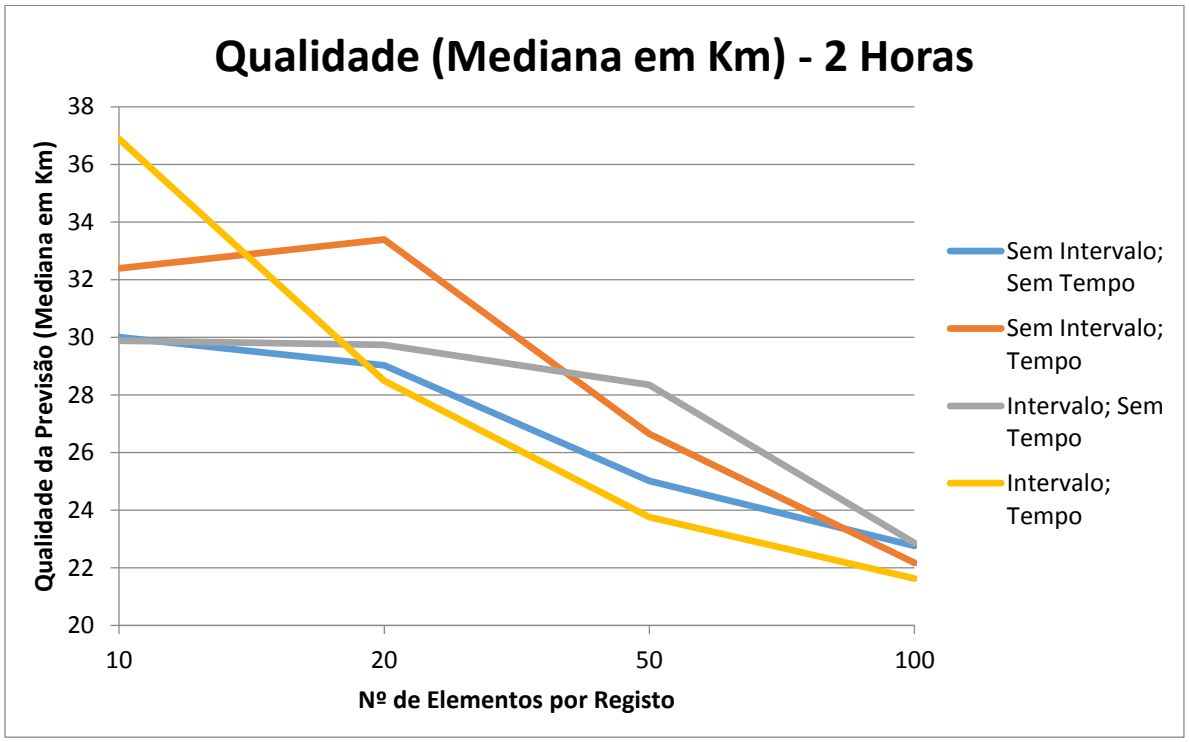


Figura 7.3 – Gráfico de qualidade das previsões geradas para intervalo de 2 horas.

QUALIDADE (MEDIANA, EM KM) - 2 HORAS

	10	20	50	100
SEM INTERVALO; SEM TEMPO	30,001	29,02419	25,02265	22,76695
SEM INTERVALO; TEMPO	32,39503	33,39516	26,64534	22,17022
INTERVALO; SEM TEMPO	29,88647	29,73479	28,34545	22,86318
INTERVALO; TEMPO	36,88312	28,49862	23,76668	21,62396

Tabela 7.4 – Tabela de qualidade das previsões geradas para intervalo de 2 horas.

Pela Figura 14 é possível ver que a tendência geral é o aumento da qualidade das soluções produzidas com o aumento do número de elementos por registo, como espetável, dadas as características da PG. Para o presente teste não foram consideradas as configurações de 200 e 500 pontos – embora altamente provável que continuassem a tendência de melhoria da qualidade devido à existência de mais informação, a sua duração foi considerada excessiva. Tendo em conta a tendência registada, é possível afirmar que a melhor configuração será aquela com o maior número de elementos (tendo apenas em conta apenas o fator qualidade).

Relativamente às configurações do modelo novo, é possível ver que, com exceção dos resultados para a configuração de 10 elementos, a configuração "Intervalo; Tempo" produz os melhores resultados. É visível também que com o aumento do número de elementos, existe uma convergência da qualidade das soluções – é possível supor que, para um número suficientemente elevado de pontos, todas as quatro configurações teriam o mesmo desempenho. No entanto, considerando apenas os dados disponíveis, a configuração "Intervalo; Tempo" apresenta os melhores resultados para configurações com maiores números de pontos.

Após a análise de ambas as vertentes, é então definida a configuração ideal. Do ponto de vista da qualidade, torna-se óbvio que idealmente seria utilizada a configuração com o maior número de elementos, dado que a presença de mais informação tende a produzir melhores resultados. No entanto, devido à importância do tempo de desempenho para o problema particular da presente tese, tornam-se necessárias concessões em ambos os aspetos (de tempo e qualidade). Como tal, é escolhida como configuração ótima para o presente trabalho de tese a configuração de 100 elementos por registos, apresentando uma combinação de qualidade de soluções e tempo de execução considerada como aceitável.

Adicionalmente visto que é escolhida a configuração de 100 elementos por registo, é escolhida como configuração de elementos extra a configuração "Intervalo; Tempo", visto que apresenta os melhores resultados para o número de elementos por registo escolhido, não sendo o tempo de execução um fator relevante para a escolha (dado que todas as quatro configurações possuem diferenças negligenciáveis no tempo de execução, para qualquer número de elementos por registo).

7.2.2. Linear Scaling

Como visto no Capítulo 5.2, a presente tese usa uma nova versão do algoritmo de GS-GP que faz uso de *linear scaling* para calcular o erro dos indivíduos. Em trabalhos anteriores, foi verificado que o uso de *linear scaling*, embora não implique necessariamente uma melhoria da qualidade das soluções, não tem um impacto negativo comparativamente à versão anterior do algoritmo (Vanneschi, 2015). Para a presente tese, dado que inicialmente foi utilizada a versão do algoritmo de GS-GP sem *linear scaling*, o presente capítulo pretende ilustrar as diferenças entre as duas versões para o problema em mãos – não necessariamente apenas relativamente à qualidade das soluções, visto estar já documentado o impacto do uso de *linear scaling* em problemas semelhantes (Vanneschi, 2015), mas também relativamente ao impacto que a nova versão tem sobre o tempo de execução do programa desenvolvido (como já visto anteriormente, um importante fator para a presente tese).

Abaixo na Tabela 7.5 são apresentados os resultados obtidos. Para avaliar a qualidade das soluções (sendo designado como qualidade a distância, em quilómetros, entre a posição prevista e a posição real, sendo considerada melhor a diferença mais pequena) foi considerada a mediana de todas as execuções, visto ser mais representativa da normal execução do algoritmo. São consideradas dois tipos de execuções: sem *linear scaling*, designada como Sem LS, que representa a versão original da aplicação, fazendo uso do algoritmo de GS-GP original; com *linear scaling*, designada como Com LS, que representa a "nova" versão da aplicação, utilizando a nova versão do algoritmo de GS-GP.

	MEDIANA	**TEMPO DE EXECUÇÃO MÉDIO**

COM LS	9,110410398	00:03:47
SEM LS	11,59516331	00:10:35

Tabela 7.5 – Mediana da qualidade (em km) e Tempo Médio de Execução para algoritmos com e sem *linear scaling*.

Pelos resultados acima, é possível confirmar que, tal como esperado pela análise de trabalhos anteriores (Vanneschi, 2015), a qualidade das soluções não diminuí pelo uso de *linear scaling*. Pelo contrário, é até verificada uma melhoria da mediana da qualidade quando é utilizado *linear scaling*. Mas o facto mais relevante é a diminuição do tempo de execução resultante do uso de *linear scaling*, sendo uma melhoria muito considerável face ao uso da versão anterior do algoritmo de GS-GP. A diminuição do tempo de execução, associada à melhoria da qualidade das soluções, confirma que o uso de *linear scaling* representa uma melhoria para a qualidade da aplicação desenvolvida, sendo portanto a primeira versão do algoritmo de GS-GP substituída pela nova versão com *linear scaling*.

7.2.3. Profundidade Inicial das Soluções

Um aspeto levantado durante a implementação do novo modelo foi qual a profundidade inicial ótima da primeira geração de indivíduos. A profundidade inicial máxima é um indicador da complexidade esperada dos indivíduos da população inicial – quanto maior a profundidade de uma árvore, mais complexa a expressão que a mesma representa. Naturalmente, quanto mais complexo um indivíduo, maior o esforço computacional que representa calcular o seu *fitness* – tradicionalmente, soluções de PG tentam sempre limitar a profundidade das soluções pois esta é responsável pelos grandes tempos de execução associadas à mesma; no entanto, nesta implementação de GS-GP, esse problema encontra-se "circundado" graças ao uso de referências (como visto no Capítulo 3.8.2), fazendo com que a profundidade não tenha um impacto tão grande como anteriormente.

A maior profundidade leva também ao uso de mais nós ou funções, o que por sua vez leva a uma maior ramificação da árvore, resultando num maior número de terminais, ou variáveis independentes. Dado que o novo modelo utiliza duas novas variáveis contextualmente diferentes das outras (Tempo e Intervalo, contrastando com 100 posições geográficas), a aprendizagem do modelo pode depender da capacidade que o algoritmo tem das interpretar e, até, distinguir das restantes. Como tal, a redução da profundidade inicial máxima levaria a uma população inicial menos complexa (e diversa) – na verdade, a proposta estudada de reduzir a profundidade inicial máxima para 1 leva a que os indivíduos da população inicial correspondam a um único terminal. Com esta redução, embora se comprometa a qualidade dos indivíduos da população inicial, podem ser criados indivíduos compostos apenas pelas variáveis Tempo e Intervalo. Usando esses indivíduos como base, o algoritmo pode então desenvolver novos indivíduos que deem um maior peso às variáveis escolhidas, eventualmente por isso melhores que soluções provenientes de indivíduos inicialmente mais complexos.

Foi então decidido realizar uma comparação entre o parâmetro tradicional (profundidade inicial máxima de 6) com uma nova configuração – profundidade inicial máxima de 1, a fim de potencialmente dar maior peso aos novos dados.

Abaixo são apresentados os resultados obtidos. Para avaliar a qualidade das soluções (sendo designado como qualidade a distância, em quilómetros, entre a posição prevista e a posição real, sendo considerada melhor a diferença mais pequena) foi considerada a mediana de todas as

execuções, visto ser mais representativa da normal execução do algoritmo. São consideradas duas configurações: a configuração tradicional, com profundidade inicial de 6 (designada por PI6); e a nova configuração, com profundidade inicial de 1 (designada por PI1). Em todos os outros aspetos, as execuções são idênticas.

	MEDIANA	TEMPO DE EXECUÇÃO MÉDIO
PI1	23,53088565	00:03:21
PI6	9,110410398	00:03:47

Tabela 7.6 – Mediana da qualidade (em km) e Tempo Médio de Execução para algoritmos de profundidade máxima da geração inicial 1 e 6.

Como se pode ver pela Tabela 7.6, a configuração PI6 apresenta resultados muito melhores que os da configuração PI1. É possível então concluir que a possibilidade de os novos elementos (Tempo e Intervalo) terem um peso maior na solução final não se traduz em melhores resultados. Tradicionalmente, a configuração PI1 apresentaria melhores tempos de execução que a configuração PI6, visto que indivíduos menores são mais fáceis de avaliar, encurtando o tempo de avaliação dos mesmos. No entanto, dadas as características de GS-GP, os indivíduos são executados apenas uma vez, quando é criada a população inicial. Todas as outras avaliações de *fitness* são realizadas com recurso a referências, o que significa que a duração da avaliação para as gerações que não a primeira é independente da profundidade máxima inicial. Na verdade, é possível observar que a configuração PI1 é ligeiramente mais rápida que a configuração PI6, mas não de um modo significativo – corresponde essa diferença de tempos de execução à criação e avaliação de indivíduos mais complexos me PI6. Caso a complexidade inicial tivesse um impacto significativo sobre o tempo de execução, a diferença entre ambas as configurações, para o presente teste, não seria de apenas alguns segundos.

Considerando os resultados acima descritos, foi decidido manter o valor tradicional de profundidade inicial máxima como 6, visto não existirem vantagens (quer de qualidade ou de tempo de execução) na sua alteração para 1.

7.3. *COMPARAÇÃO COM OUTROS MODELOS DE INTELIGÊNCIA COMPUTACIONAL*

Em paralelo com o presente trabalho de tese foi realizado um estudo destinado a validar a qualidade das soluções produzidas pelo algoritmo de GS-GP utilizado quando comparadas a soluções geradas por outras técnicas de IC. Para o presente estudo, realizado por Castelli et al. (Castelli, 2015), foram consideradas técnicas normalmente utilizadas para o mesmo tipo de problema como base de comparação para o desempenho do algoritmo de GS-GP: *Square Regression* (SQ) (Bates, 1988) e *Radial Basis Function Networks* (RBF) (Broomhead, 1988).

Para os testes realizados, foi realizada uma previsão para 2 horas após a última posição registada. Foram consideradas as posições de 5 embarcações diferentes, de diferentes tipos: navio de passageiros (11.571 registos para treino e teste); navio rebocador (11.390 registos); navio de pesca (8.519 registos); navio tanque (11.896 registos); navio cargueiro (11.964 registos). Dado o diferente tipo das embarcações selecionadas, a amostra representa vários tipos diferentes de rotas (por exemplo, um navio de passageiros possui uma rota muito mais regular que um navio de pesca, cujo a rota é imprevisível). Os conjuntos de treino e de teste utilizados foram criados segundo o modelo

novo definido no Capítulo 7.1, sendo os mesmos conjuntos usados para todas as técnicas de previsão testadas (GS-GP, SQ, RBF). O algoritmo de GS-GP foi configurado como descrito no Capítulo 5.3, com a diferença de o tamanho do torneio considerado ser de 4. Foram executadas 30 execuções para cada técnica, onde a medida de *fitness* escolhida é o RMSE. Abaixo encontram-se os diagramas de caixa apresentando os resultados obtidos para latitude e para longitude (considerando o erro). Para cada diagrama, a marca central de cada caixa representa a mediana, os limites da caixa representam os 25º e 75º percentis e as marcas para lá da caixa representam os resultados mais extremos obtidos.

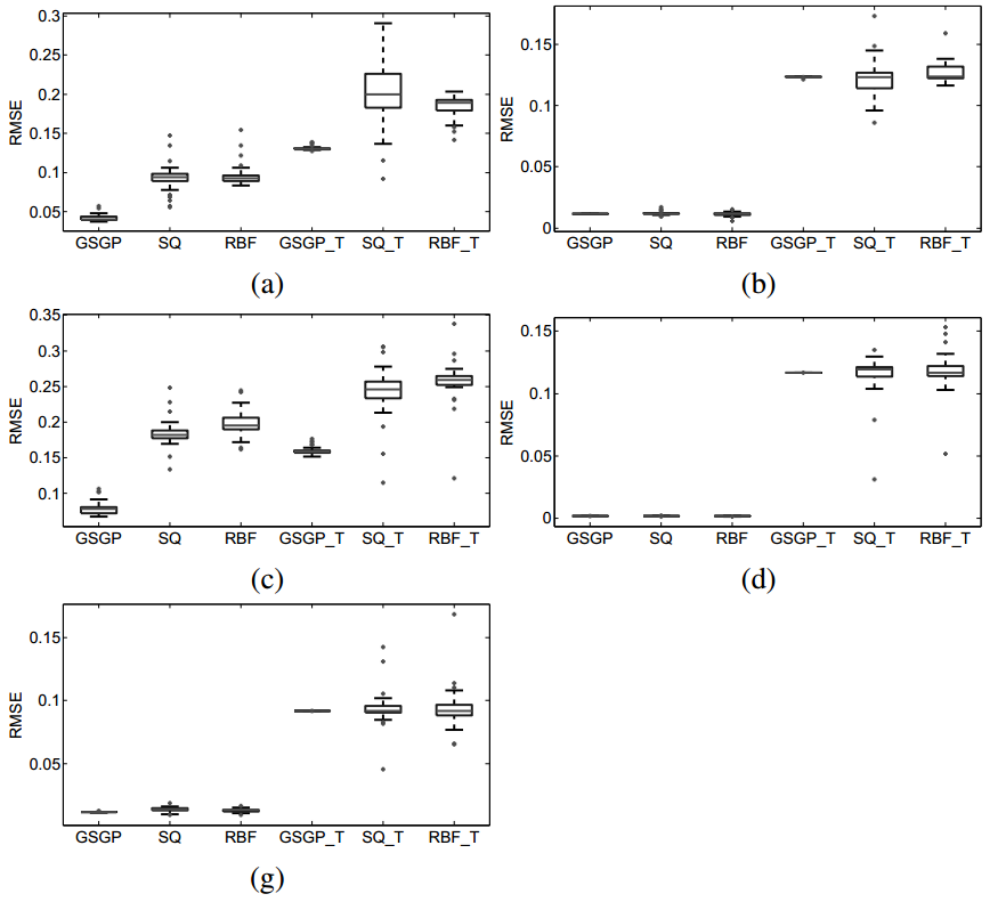


Figura 7.4 – Diagrama de caixa do RMSE entre a posição prevista e a posição real (latitude) para navio de passageiros (a), navio rebocador (b), navio de pesca (c), navio cargueiro (d) e navio tanque (g). GSGP, SQ e RBF representam os resultados obtidos para os conjuntos de treino; GSGP_T, SQ_T e RBF_T representam os resultados obtidos para o conjunto de teste (Castelli, 2015).

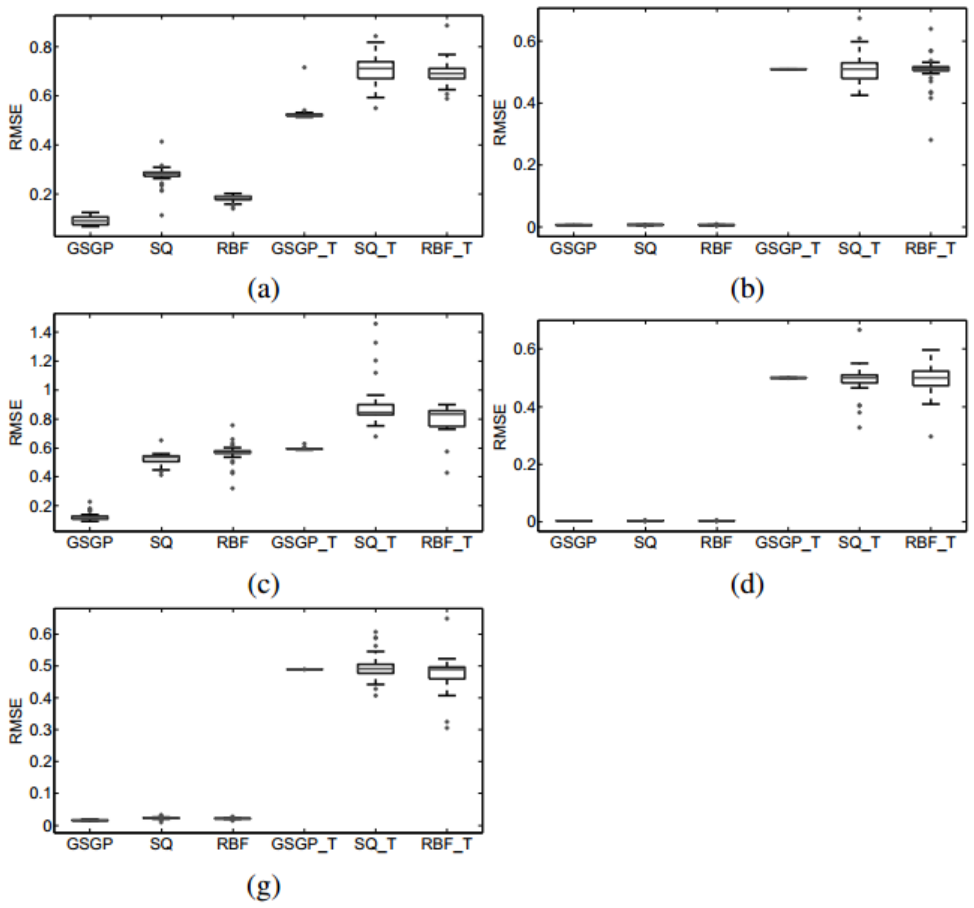


Figura 7.5 – Diagrama de caixa do RMSE entre a posição prevista e a posição real (longitude) para navio de passageiros (a), navio rebocador (b), navio de pesca (c), navio cargueiro (d) e navio tanque (g). GSGP, SQ e RBF representam os resultados obtidos para os conjuntos de treino; GSGP_T, SQ_T e RBF_T representam os resultados obtidos para o conjunto de teste (Castelli, 2015).

Como pode ser observado acima nas figuras 7.4 e 7.5, o algoritmo de GS-GP produz resultados tão bons ou melhores que aqueles produzidos pelas outras técnicas testadas, obtendo um melhor desempenho para todos os casos de treino e, para casos de teses, resultados melhores para 3 das 5 embarcações (sendo o desempenho comparável para as restantes 2 embarcações) (Castelli, 2015).

Verificando o bom desempenho do algoritmo de GS-GP, é necessário acrescentar que existem outras técnicas de CI capazes de produzir resultados semelhantes (redes neuronais, máquinas de vetores de suporte, etc.). No entanto, embora outras técnicas sejam capazes de produzir resultados comparáveis, apenas o fazem após um demorado processo de execução, o que as torna efetivamente inutilizáveis num contexto real (Castelli, 2015). Do mesmo modo, os resultados produzidos pelo algoritmo de GS-GP poderiam ser melhores do que os apresentados caso a execução considerasse mais indivíduos, mais iterações ou um maior número de elementos por registo, etc. – no entanto, tais alterações levariam a um aumento do tempo de execução. Como tal, embora exista um erro inerente ao uso do algoritmo, o mesmo é considerado satisfatório (não é necessário prever a posição exata da embarcação em causa se a previsão estiver perto o suficiente para que a embarcação posso ser avistada – permitindo uma intervenção eficaz da Marinha). Na tabela 7.7 é possível ver o tempo médio de execução do algoritmo de GS-GP (não o tempo de execução da totalidade da solução desenvolvida, apenas o tempo que o algoritmo demora a ser executado para

uma coordenada – latitude ou longitude). Na tabela 7.8 é possível ver qual o error registado para cada embarcação (mínimo e mediana), sendo a distância em quilómetros entre o ponto real e o ponto previsto.

TIPO DE NAVIO	TEMPO DE EXECUÇÃO (SEGUNDOS)
NAVIO DE PASSAGEIROS	75,4
NAVIO REBOCADOR	76,3
NAVIO DE PESCA	67,8
NAVIO TANQUE	74,1
NAVIO CARGUEIRO	75,9

Tabela 7.7 – Tempo médio de execução (em segundos) de cada tipo de embarcação, retirado de 30 execuções independentes.

	DISTÂNCIA (KM)	
TIPO DE NAVIO	*MÍNIMA*	*MEDIANA*
NAVIO DE PASSAGEIROS	0,58	3,626
NAVIO REBOCADOR	0,243	1,72
NAVIO DE PESCA	0,687	8,918
NAVIO TANQUE	0,332	1,294
NAVIO CARGUEIRO	0,093	0,156

Tabela 7.8 – Distância mínima e mediana (em quilómetros) registada para cada tipo de embarcação, retirada de 30 execuções independentes.

Adicionalmente, são apresentados os mapas dos pontos gerados pelo programa para facilitar a interpretação dos dados acima (sendo apresentado os mapas que registaram a menor diferença de distância entre os pontos a prever passadas 2 horas). Nos mapas abaixo, o ponto laranja representa o ponto real a ser previsto pela aplicação e o ponto amarelo representa a previsão criada. Adicionalmente, os pontos vermelhos representam parte da rota real da embarcação 2 horas antes da previsão, e os verdes à correspondente rota prevista.

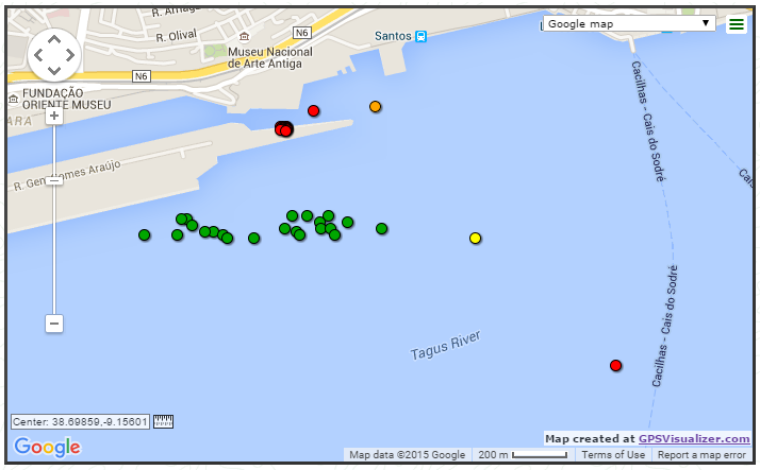


Figura 7.6 – Mapa das posições previstas (amarelo – previsão; rota – verde) e correspondentes posições reais (laranja – ponto a prever; vermelho – rota) do navio de passageiros.

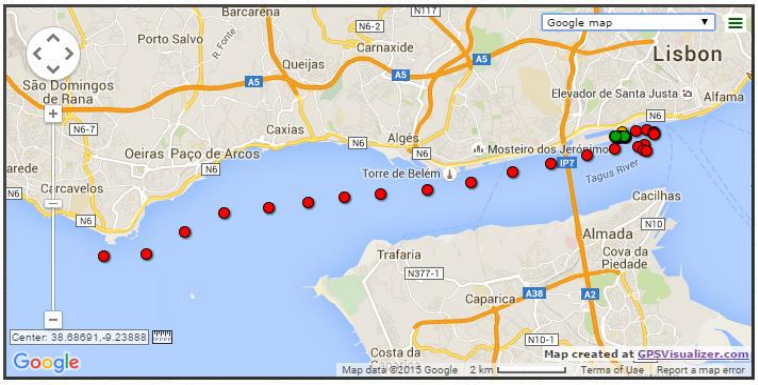


Figura 7.7 – Mapa das posições previstas (amarelo – previsão; rota – verde) e correspondentes posições reais (laranja – ponto a prever; vermelho – rota) do navio rebocador.

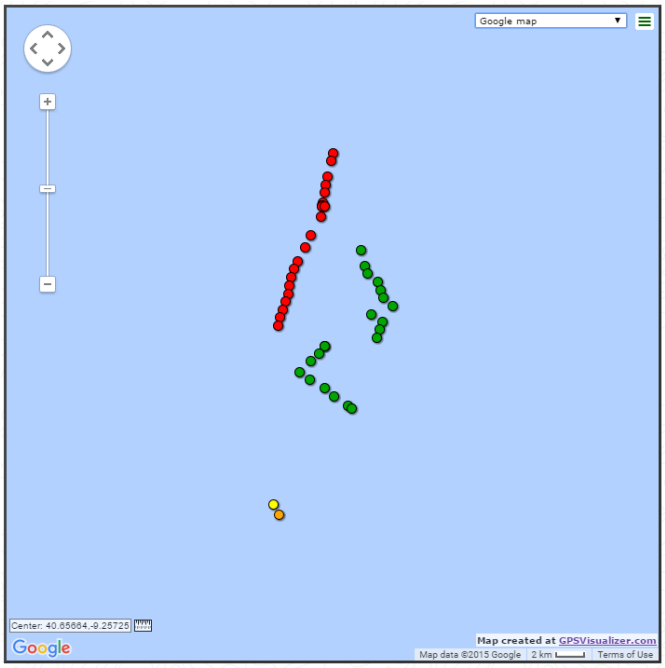


Figura 7.8 – Mapa das posições previstas (amarelo – previsão; rota – verde) e correspondentes posições reais (laranja – ponto a prever; vermelho – rota) do navio de pesca.

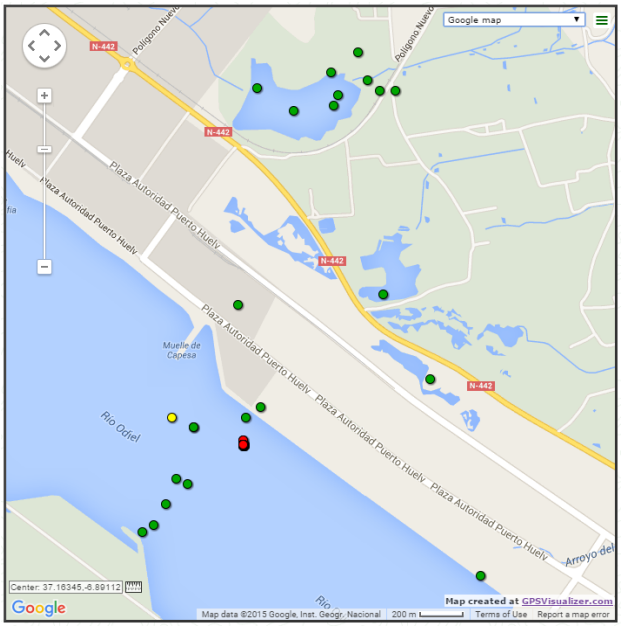


Figura 7.9 – Mapa das posições previstas (amarelo – previsão; rota – verde) e correspondentes posições reais (laranja – ponto a prever; vermelho – rota) do navio tanque.

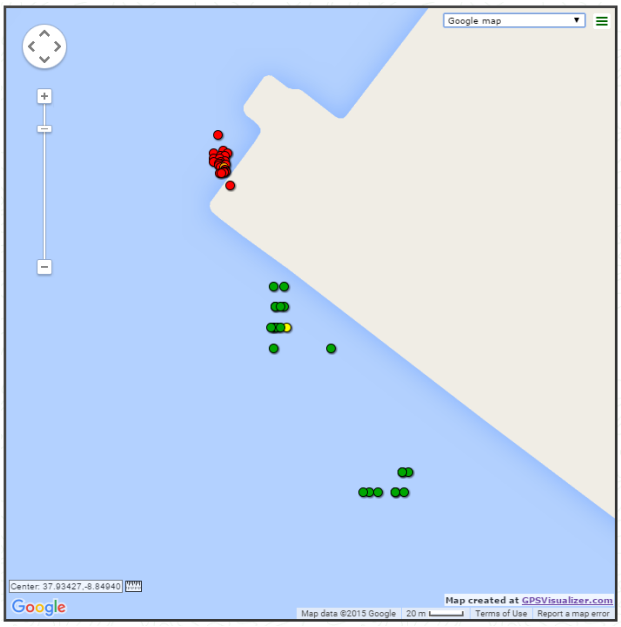


Figura 7.10 – Mapa das posições previstas (amarelo – previsão; rota – verde) e correspondentes posições reais (laranja – ponto a prever; vermelho – rota) do navio cargueiro.

Pela análise dos dados acima, é possível concluir que o algoritmo de GS-GP é adequado para ser utilizado para o problema em questão, produzindo resultados equiparáveis (quando não melhores que) os produzidos por outras técnicas frequentemente utilizadas para o mesmo problema – sendo o tempo de execução e o error de previsão considerados como aceitáveis pela Marinha Portuguesa (Castelli, 2015). Tendo os resultados apresentados sido gerados pelo programa desenvolvido, serve o presente capítulo como uma confirmação da qualidade das soluções produzidas pela solução desenvolvida.

8. CONSIDERAÇÕES FINAIS

Voltando a considerar os objetivos definidos para o presente trabalho, é possível afirmar que os mesmos foram cumpridos na sua totalidade. Considerando o objetivo final definido para o presente trabalho, considerem-se os seguintes dois pontos que o constituem:

- Desenvolver uma solução informática que integre o algoritmo de GS-GP desenvolvido por Vanneschi et al. (Vanneschi, 2013) com o sistema Oversee da Critical Software, permitindo o seu uso pela Marinha Portuguesa;
- Configuração do mesmo sistema a fim de assegurar uma execução eficiente do mesmo;

Ambos os pontos podem ser considerados cumpridos aquando o final da tese. A solução encontra-se pronta a ser entregue à Critical Software, realizando a integração originalmente pedida. Do mesmo modo, os parâmetros do algoritmo de GS-GP que foram considerados com maior probabilidade de impacto no desempenho do mesmo foram estudados, sendo escolhidas as configurações que melhores resultados apresentam para tempo de execução e qualidade de resultados, simultaneamente.

Adicionalmente, foram desenvolvidos dois novos modos de operação para o algoritmo de GS-GP, não definidos no início da tese. Estes desenvolvimentos foram realizados tendo por base o *feedback* dado pela equipa da Critical Software entre iterações de implementação, que os considerou como plausíveis de gerar valor acrescentado à solução a ser desenvolvida. Também para lá do âmbito original da tese foi o trabalho adicional realizado para o artigo de Castelli et al. (Castelli, 2015), criando novas evidências de que o algoritmo de PG escolhido é capaz de produzir soluções competitivas com aquelas geradas por ferramentas atualmente utilizadas – servindo também, no âmbito deste trabalho de tese apenas, de confirmação adicional da qualidade das soluções produzidas.

Com base nos resultados obtidos, é possível afirmar que a PG, especialmente a GSGP, tem um grande potencial para diversas aplicações em problemas semelhantes. Considerando a atual tendência de organizações guardarem grandes volumes de dados, dos quais é difícil extrair informação relevante, a PG apresenta-se como uma ferramenta capaz de gerar grande valor numa grande variedade de áreas. Serve o presente trabalho de tese, dentro do seu âmbito, como mais uma aplicação funcional de PG a um problema real, sendo capaz de gerar soluções utilizáveis para uma resposta atempada e eficaz.

8.1. *LIMITAÇÕES*

Como referido durante o Capítulo 6, as limitações do presente trabalho de tese depreendem muito da natureza no mesmo. Nomeadamente, é possível considerar como limitação presente durante todo a tese o acesso limitado a dados atuais referentes a posições de embarcações.

Relativamente ao acesso aos dados, é de referir que grande parte da investigação realizada na presente tese deriva do uso de dados de teste. Esta limitação existe devido à natureza sensível dos dados a ser utilizados – visto tratarem-se de dados referentes a embarcações, gerados e geridos pela Marinha Portuguesa, o acesso aos mesmos é naturalmente condicionado. Como tal, embora fosse ideal poder aceder aos dados presentes nos servidores em tempo real, o trabalho realizado apenas

pode contar com acesso aos mesmos durante curtos períodos de tempo. Embora os dados utilizados sejam reais e por isso demonstrativos do desempenho do algoritmo, o acesso a um maior número de dados (preferencialmente em tempo real) permitiria realizar um maior número de testes sobre o desempenho da solução desenvolvida.

8.2. *PROPOSTA DE TRABALHO FUTURO*

De maneira a responder à limitação levantada na secção anterior, é recomendado um estudo dos resultados produzidos pela solução num ambiente *live*, a fim de gerar uma amostra de resultados que derive de um maior número de casos. Do mesmo modo, num ambiente *live* é possível analisar de um modo mais operacional os resultados produzidos pela solução. Após a entrega da solução, uma análise dos resultados produzidos levaria a uma melhor compreensão do desempenho global (ou seja, para qualquer caso do problema proposto) da solução desenvolvida.

O novo modelo de *input* utilizado no presente trabalho de tese pode representar um benefício para trabalhos futuros devido à sua capacidade de produzir vários resultados úteis numa única execução de GSGP. Como trabalho futuro, é pretendida uma análise do mesmo a fim de definir se o mesmo poderá produzir resultados melhores através de uma diferente configuração, nomeadamente:

- A premiação do uso das variáveis independentes relativas a tempo, aumentando a sua importância no algoritmo. Tal alteração poderá levar a uma aprendizagem mais eficaz do parâmetro temporal por parte do algoritmo, o que poderá levar a um aumento da qualidade das soluções;
- A redução do número de dados de *input*. Visto que o novo modelo junta os vários intervalos temporais num único ficheiro, é plausível que alguma informação seja redundante, ou que traga pouco benefício para a análise. Como o maior número de dados se traduz em maiores tempos de execução, a remoção de informação redundante poderia levar a uma redução do tempo de execução do algoritmo sem comprometer a qualidade das soluções geradas pelo mesmo.

9. BIBLIOGRAFIA

(2015). *E-Navigation.* Available at http://www.imo.org/en/OurWork/Safety/Navigation/Pages/eNavigation.aspx consulted 06/08/2015.

(1998). *Safety Study of the Operational Relationship between Ship Master/Watchkeeping Officers and Marine Pilots.*

Archetti, F.; Lanzeni, S.; Messina, E.; Vanneschi, L. (2007) Genetic Programming for Computational Pharmacokinetics in Drug Discovery and Development. In *Genetic Programming and Evolvable Machines*, volume 8, pages 413-432.

Banzhaf, W. (1993) *Genetic Programming for Pedestrians. In Proceedings of the 5th International Conference on Genetic Algorithms, Icga-93.* Morgan Kaufmann*,* Mitsubishi Electrical Research Laboratories, Cambridge Research Center.

Bates, D. M.; Watts, D. G. (1988) *Nonlinear Regression Analysis and Its Applications.* Wiley.

Beadle, L.; Johnson, C. G. (2008) Semantically Driven Crossover in Genetic Programming. In *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*pages 111-116.

Beadle, L.; Johnson, C. G. (2009) Semantic Analysis of Program Initialisation in Genetic Programming. In *Genetic Programming and Evolvable Machines*, volume 10, no. 3, pages 307-337.

Beadle, L.; Johnson, C. G. (2009) Semantically Driven Mutation in Genetic Programming. In *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*pages 1336-1342.

Bensassi, S.; Martinez-Zarzoso, I. (2012) How Costly Is Modern Maritime Piracy to the International Community? In *Review of International Economics*, volume 20, no. 5, pages 869-883.

Bensassi, S.; Martinez-Zarzoso, I. (2013) The Price of Modern Piracy. In *Defense and Peace Economics*, volume 24, no. 5, pages 397-418.

Blickle, T.; Thiele, L. (1994) Genetic Programming and Redundancy" In *Proc. Genetic Algorithms within the Framework of Evolutionary Computation (Workshop at KI-94)*.

Broomhead, D. S.; Lowe, D. (1988) Multivariable Functional Interpolation and Adaptive Networks. In *Complex Systems*, volume 2, no. 3, pages 321-355.

Bukhari, A. C.; Tusseyeva, I.; Iee, B.; Kim, Y. (2013) An Intelligent Real-Time Multi-Vessel Collision Risk Assessment System from Vts View Point Based on Fuzzy Inference System. In *Expert Systems with Applications*, volume 40, no. 4, pages 1220-1230.

Burke, E. K.; Kendal, G.; Newall, J.; Hart, E.; Ross, P.; Schulenburg, S. (2003) Hyper-Heuristics: An Engineering Direction in Modern Search Technology. In *Handbook of Metaheuristics*pages 457-474.

Castelli, M.; Vanneschi, L.; Fonseca, P. (2015) Improving Maritime Safety by Analyzing Ais Data: An Artificial Intelligence Approach. In *International Journal of Shipping and Transport Logistics (To Be Published)*.

Chen, C.-H.; Khoo, L. P.; Chong, Y. T.; Tin, X. F. (2014) Knowledge Discovery Using Genetic Algorithm for Maritime Situational Awareness. In *Expert Systems with Applications*, volume 41, no. 6, pages 2742-2753.

Daida, J. M.; Hommes, J. D.; Bersano-Begey, T. F.; Ross, S. J.; Vesecky, J. F. (1996) Algorithm Discovery Using the Genetic Programming Paradigm: Extracting Low-Contrast Curvilinear Features from Sar Images of Arctic Ice. In *Advances in Genetic Programming 2*pages 417-442.

Darwin, C. (1859) *On the Origin of Species by Means of Natural Selection.* John Murray, London.

Dianati, M.; Song, I.; Treiber, M. (2002) An Introduction to Genetic Algorithms and Evolution Strategies. In *Technical report, University of Waterloo, Ontario, N2L 3G1, Canada*.

Estrutura de Missão para a Extensão da Plataforma Continental (EMEPC) (2009). *Continental Shelf Submission of Portugal, Pt-Es/05-05-2009.* Available at http://www.un.org/Depts/los/clcs_new/submissions_files/prt44_09/prt2009executivesummary.pdf, consulted 05/05/2015.

Fogel, L. J. (1962) Autonomous Automata. In *Industrial Research Magazine*, volume 4, no. 2, pages 14-19.

Forrest, S. (1996) Genetic Algorithms. In *ACM Computing Surveys (CSUR)*, volume 28, no. 1, pages 77-80.

Forrest, S; Mitchell, M. (1993) What Makes a Problem Hard for a Genetic Algorithm? Some Anomalous Results and Their Explanation. In *Machine Learning*, volume 13, no. 2-3, pages 285-319.

Hauptman, A.; Sipper, M. (2007) Evolution of an Efficient Search Algorithm for the Mate-in-N Problem in Chess. In *Proceedings of the 10th European Conference on Genetic Programming*, volume 4445, pages 78-89.

Holland, J. H. (1975) *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence.* MIT Press*,* Cambridge, MA, USA.

Howard, D.; Roberts, S. C.; Ryan, C. (2006) Pragmatic Genetic Programming Strategy for the Problem of Vehicle Detection in Airborne Reconnaissance. In *Pattern Recognition Letters*, volume 27, no. 11, pages 1275-1288.

International Association of Maritime Aids to Navigation and Lighthouse Authorities (IALA) (2002). *Iala Guidelines on the Universal Automatic Identification System (Ais).*

Jackson, D. (2010) Phenotypic Diversity in Initial Genetic Programming Populations. In *Lecture Notes in Computer Science*, volume 6021, pages 98-109.

Kazemi, S.; Abghari, S.; Lavesson, N.; Johnson, H.; Ryman, P. (2013) Open Data for Anomaly Detection in Maritime Surveillance. In *Expert Systems with Applications*, volume 40, no. 14, pages 5719-5729.

Keijzer, M. (2003) Improving Symbolic Regression with Interval Arithmetic and Linear Scaling. In *Genetic Programming, Proceedings of EuroGP'2003*, volume 2610, pages 70-82.

Koza, John R. (1992) *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Bradford Book, MIT Press.

Koza, John R.; Forrest, H.; Stiffelman, O. (1999) Genetic Programming as a Darwinian Invention Machine. In *Genetic Programming, Proceedings of EuroGP'99*, volume 1598, pages 93-108.

Laxhammer, R. (2008) Anomaly Detection for Sea Surveillance. In *2008 11th International Conference on Information Fusion*pages 1-8.

Lohn, J.; Hornby, G.; Linden, D. (2004) Evolutionary Antenna Design for a Nasa Spacecraft. In *Genetic Programming Theory and Practice II*pages 301-315.

Martin, M. C. (2006) Evolving Visual Sonar: Depth from Monocular Images. In *Pattern Recognition Letters*, volume 27, no. 11, pages 1174-1180.

McPhee, N. F.; Ohs, B.; Hutchinson, T. (2007) Semantic Building Blocks in Genetic Programming. In *Working Paper Series*, volume 3, no. 2, pages 134-145.

Miller, J. F. (1999) An Empirical Study of the Efficiency of Learning Boolean Functions Using a Cartesian Genetic Programming Approach. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1135-1142.

Moraglio, A.; Krawiec, K. & Johnson, C. G. (2012) Geometric Semantic Genetic Programming. In *Parallel Problem Solving from Nature – PPSN XII, Lecture Notes in Computer Science*, volume 7419, pages 21-31.

Poli, R.; Langdon, W. B. & McPhee, N. F. (2008) A Field Guide to Genetic Programming. Journal, volume.

Riveiro, M; Falkman, G.; Ziemke, T. (2008) Visual Analytics for the Detection of Anomalous Maritime Behaviour. In *2009 12th International Conference on Information Visualization*pages 273-279.

Schwefel, H. P. (1975) *Evolutionsstrategie Und Numerische Optimierung (Ph.D. Thesis).* Technical University of Berlin, Department of Process Engineering.

Silva, S.; Almeida, J. (2003) Gplab - a Genetic Programming Toolbox for Matlab. In *Proceedings of the Nordic MATLAB Conference*pages 273-278.

Tetreault, B. J. (2005) Use of the Automatic Identification System (Ais) for Maritime Domain Awareness (Mda). In *OCEANS, 2005. Proceedings of MTS/IEEE*, volume 2, pages 1590-1594.

Theiler, J. P.; Harvey, N. R.; Brumbly, S. P.; Szymanski, J. J.; Alferink, S.; Perkins, S. J.; Porter, R. B.; Bloch, J. J. (1999) Evolving Retrieval Algorithms with a Genetic Programming Scheme. In *Proceedings of SPIE 3753 Imaging Spectrometry V*pages 416-425.

Trucco, P.; Cagno, E.; Ruggeri, F.; Grande, O. (2008) A Bayesian Belief Network Modelling of Organizational Factors in Risk Analysis: A Case Study in Maritime Transportation. In *Reliability Engineering & System Safety*, volume 93, no. 6, pages 845-856.

Trujilo, L.; Olague, G. (2006) Synthesis of Interest Point Detectors through Genetic Programming. In *GECCO 2006: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, volume 1, no. 887-894.

Turing, A. M. (1950) Computing Machinery and Intelligence. In *Mind*, volume 59, pages 433-460.

Vanneschi, L.; Castelli, M.; Costa, E.; Re, A.; Vaz, H.; Lobo, V.; Urbano, P. (2015) Improving Maritime Awareness with Semantic Genetic Programming and Linear Scaling: Prediction of Vessels Position Based on Ais Data. In *Lecture Notes in Computer Science*, volume 9028, pages 732-744.

Vanneschi, L.; Castelli, M.; Manzoni, L.; Silva, S. (2013) A New Implementation of Geometric Semantic Gp and Its Application to Problems in Pharmacokinetics. In *Genetic Programming, Lecture Notes in Computer Science*, volume 7831, pages 205-216.

Vanneschi, L.; Castelli, M.; Manzoni, L.; Silva, S. (2014) Geometric Semantic Genetic Programming for Real Life Applications. In *To be Published*.

Vanneschi, L.; Castelli, M.; Silva, S. (2015) A C++ Framework for Geometric Semantic Genetic Programming. In *Genetic Programming and Evolvable Machines*, volume 16, no. 1, pages 73-81.

Wright, S. (1932) The Roles of Mutation, Inbreeding, Crossbreeding, and Selection in Evolution. In *Proceedings of the Sixth International Congress on Genetics*pages 355-366.

Yang, S.; Li, L.; Suo, Y.; Chen, G. (2007) Study on Construction of Simulation Platform for Vessel Automatic Anti-Collision and Its Test Method. In *Automation and Logistics, 2007 IEEE International Conference on*pages 2414-2419.

Zhang, M.; Smart, W. (2006) Using Gaussian Distribution to Construct Fitness Functions in Genetic Programming for Multiclass Object Classification. In *Pattern Recognition Letters*, volume 27, no. 11, pages 1266-1274.

Zhao, Y.; Li, W.; Feng, S.; Washington, Y.; Schuster, W. (2014) An Improved Differential Evolution Algorithm for Maritime Collision Avoidance Route Planning. In *Abstract and Applied Analysis*, volume 2014.