Genetic Programming Based on Novelty Search

Enrique Naredo

Enrique Naredo García

Genetic Programming Based on Novelty Search

Genetic Programming Based on Novelty Search

Enrique Naredo García

SEP Tecnológico Nacional de México

Instituto Tecnológico de Tijuana
Departamento de Ingeniería Eléctrica y Electrónica

Programación Genética Basada en Búsqueda de Novedad

tesis sometida por:
Enrique Naredo García

para obtener el grado de:
Doctor en Ciencias de la Ingeniería

director:
Dr. Leonardo Trujillo Reyes

Tijuana, Baja California, México.
3 de Junio del 2016

SEP Tecnológico Nacional de México

Instituto Tecnológico de Tijuana
Departamento de Ingeniería Eléctrica y Electrónica

Genetic Programming Based on Novelty Search

thesis submitted by:
Enrique Naredo García

to obtain the degree of:
Doctor in Engineering Sciences

advisor:
Dr. Leonardo Trujillo Reyes

Tijuana, Baja California, México.
June 3, 2016

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO
Instituto Tecnológico de Tijuana

"2015, Año del Generalísimo José María Morelos y Pavón"

Tijuana, B.C., 23/Mayo/2016
ASUNTO: Autorización de impresión de Trabajo de Tesis

Lic. Doroteo Luna Castañeda
Jefe del Dpto. de Servicios Escolares
Presente.

En lo referente al trabajo de tesis, **"Programación genética basada en búsqueda de novedad "**, presentado por el **M.C. Enrique Naredo García**, alumno del Doctorado en Ciencias de la Ingeniería, con numero de Control **D81320113**, informamos a usted que después de una minuciosa revisión e intercambio de opiniones, los miembros del comité manifiestan APROBAR LA TESIS, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias por lo que se autoriza al interesado para que procesa de inmediato a la impresión del mismo.

A T E N T A M E N T E
"Por una Juventud Integrada al Desarrollo de México"

DR. PIERRICK LEGRAND
PRESIDENTE

DR. LEONARDO TRUJILLO REYES
SECRETARIO

DR. NOHÉ RAMÓN CÁZAREZ CASTRO
VOCAL

DR. LUIS NÉSTOR CORIA DE LOS RÍOS
VOCAL

DR. VICTOR HUGO DÍAZ RAMÍREZ
VOCAL

DR. PAULO JORGE CUNHA VAZ DIAS
URBANO
SUPLENTE

C.c.p. Oficina de Titulación
C.c.p. División de Estudios de Posgrado e Investigación
C.c.p. Expediente
C.c.p. Interesado
YMR

Resumen

La búsqueda de novedad (Novelty Search –NS) es un enfoque único hacia la búsqueda y optimización, en donde una función objetivo explicita es reemplazada por una medida de novedad de la solución. Sin embargo, NS ha sido mayormente utilizada en robótica evólutiva, su utilidad en problemas clásicos de aprendizaje máquina no ha sido explorado. Este trabajo presenta un algoritmo de programación genética (GP) basado en novedad para clasificación supervisada, con las siguientes contribuciones. Se muestra que NS puede resolver problemas de clasificación del "mundo real", validado en problemas de referencia binarios y multiclase. Estos resultados son posibles al utilizar un descriptor de comportamiento específico del dominio del problema, relacionado con el concepto de semántica en GP. Por otra parte, se proponen dos nuevas versiones del algoritmo de NS; búsqueda de novedad probabílistica (PNS) y una variante del criterio mínimo de NS (MCNS). El primero modela el compartamiento de cada solución como un vector aleatorio elimiando todos los parámetros de NS, reduciendo además el costo computacional del algoritmo. El segundo utiliza una función objetivo para restringir y sesgar la búsqueda hacia soluciones con un alto desempeño. Este trabajo discute los efectos de NS en la dinámica de la búsqueda de GP y en el crecimiento de código. Los resultados muestran que NS puede ser utilizado como una alternativa real para clasificación supervisada. Además muestran que para problemas binarios de clasificación el algoritmo de NS exhibe un implícito control de bloat.

Palabras clave: Programación Genética, Búsqueda de Novedad, Clasificación, Decepción, Bloat.

Abstract

Novelty Search (NS) is a unique approach towards search and optimization, where an explicit objective function is replaced by a measure of solution novelty. However, NS has been mostly used in evolutionary robotics, its usefulness in classic machine learning problems has been unexplored. This thesis presents a NS-based Genetic Programming (GP) algorithms for common machine learning problems, with the following contributions. It is shown that NS can solve real-world classification, clustering and symbolic regression tasks, validated on real-world benchmarks and synthetic problems. These results are made possible by using a domain-specific behavior descriptor, related to the concept of semantics in GP. Moreover, two new versions of the NS algorithm are proposed, Probabilistic NS (PNS) and a variant of Minimal Criteria NS (MCNS). The former models the behavior of each solution as a random vector and eliminates all the NS parameters while reducing the computational overhead of the NS algorithm; the latter uses a standard objective function to constrain and bias the search towards high performance solutions. The thesis also discusses the effects of NS on GP search dynamics and code growth. Results show that NS can be used as a realistic alternative for machine learning, and particularly for GP-based classification.

Keywords: Genetic Programming, Novelty Search, Classification, Deception, Bloat.

To my whole dear family and to all my true friends.

Acknowledgements

This dissertation has been possible because the convergence of many wills that supported my own.

Firstly, I am very grateful to my dissertation committee for their suggestions and constructive remarks, which enabled me to improve greatly the quality of my research work: Dr. Pierrick Legrand, Dr. Leonardo Trujillo Reyes, Dr. Nohé Ramón Cázares Castro, Dr. Luis Néstor Coria de los Rios, Dr. Victor Hugo Díaz Ramírez, and Dr. Paulo Jorge Cunha Vaz Dias. Particularly, I would like to thank my advisor Leonardo Trujillo for his pretty patient guide, and specially for his detailed and constructive feedback to keep my research work on the right path.

Secondly, I would like to thank all my colleagues from the Instituto Tecnológico de Tijuana (ITT) doctoral program in engineering sciences at the Otay campus, particularly to all members of the Tree-Lab, for being open, friendly, and even for creating a pleasant working atmosphere.

Thirdly, thanks to all the researchers involved in the project ACOBSEC, who gave me a great opportunity to visit their corresponding labs at their universities, and even more important to share with me their time and knowledge: Dr. Pierrick Legrand (INRIA and Université de Bordeaux, France), Dr. Francisco Fernández de Vega (Universidad de Extremadura, Spain), Dr. Gustavo Olague (CICESE Ensenada, México), Dr. Sara Silva (Universidade de Lisboa), and of course, to all of their corresponding students.

Special thanks to Dr. Paulo Urbano (Universidade de Lisboa), for all of his invaluable comments to my research work. Even though, we met each other because a very funny causality at the Universidade de Lisboa, we started an interesting and valuable research work in addition to a friendship, which has been increasing over time.

On a personal note, I would like to thank all the friends ('contras') I made from the ITT postgraduate program in computer sciences at the Tomás Aquino campus, for their unconditional support on my bad and sometimes worst days in my PhD journey, and because they always make me feel as a part of their family as well as they are for me.

On a more personal note, I am very thankful to all my family for their lovely support. They are far away in distance, but always close in affection.

Lastly, I would like to thank the funding provided by CONACYT (México) for my scholarship No. 232288, as well as the funding provided by FCT project EXPL/EEISII/1861/2013, and by CONACYT Basic Science Research Project No. 178323, DGEST (Mexico) Research Project 5414.14-P. Particularly I am very thankful for all the support from the project ACOBSEC: FP7-PEOPLE-2013-IRSES financed by the European Commission with contract No. 612689.

Contents

1

Introduction

Many of the problems which concern to human beings to improve their quality of life can be posed as optimization problems. However, in order to do so, we need to be able to describe the system that is involved in the problem. With enough knowledge about the system we can derive a useful abstraction and describe it by a model, identifying the variables and constraints which are relevant to improve some objective. Often, for an engineering problem the objective is to choose the design parameters that maximize the benefits, or minimizes costs without violating the constraints, in other words finding the best feasible solution. Therefore, solutions must be scored according to the quality they show when solving the problem, this typically is measured through to what is known as an objective function or reward function, which precisely rewards those solutions that are closer to the objective. Depending on the problem domain, solutions can be encoded with real-valued or discrete variables, the latter ones are referred to as combinatorial optimization (CO) problems.

Examples of CO problems are the Travellings Salesman Problem (TSP), the Quadratic Assignment problem (QAP), scheduling problems, and many others. Due to the practical importance of CO problems, many algorithms to tackle them have been developed. These algorithms can be classified as either complete or approximate algorithms. Complete algorithms are guaranteed to find for every finite size instance of a CO problem an optimal solution in bounded time *metaheuristics*.

Therefore, complete methods might need exponential computation time in the worst-case. This often leads to computation times too high

for practical purposes. In approximate methods we sacrifice the guarantee of finding optimal solutions for the sake of getting good solutions in a substantially reduced amount of time.

In the last 30 years, a new kind of approximate algorithms has emerged which basically tries to combine basic heuristic methods in higher level frameworks aimed at efficiently and effectively exploring a search space, these methods are nowadays commonly called *metaheuristics*. Figure 1.1 shows the different classifications of metaheuristics.

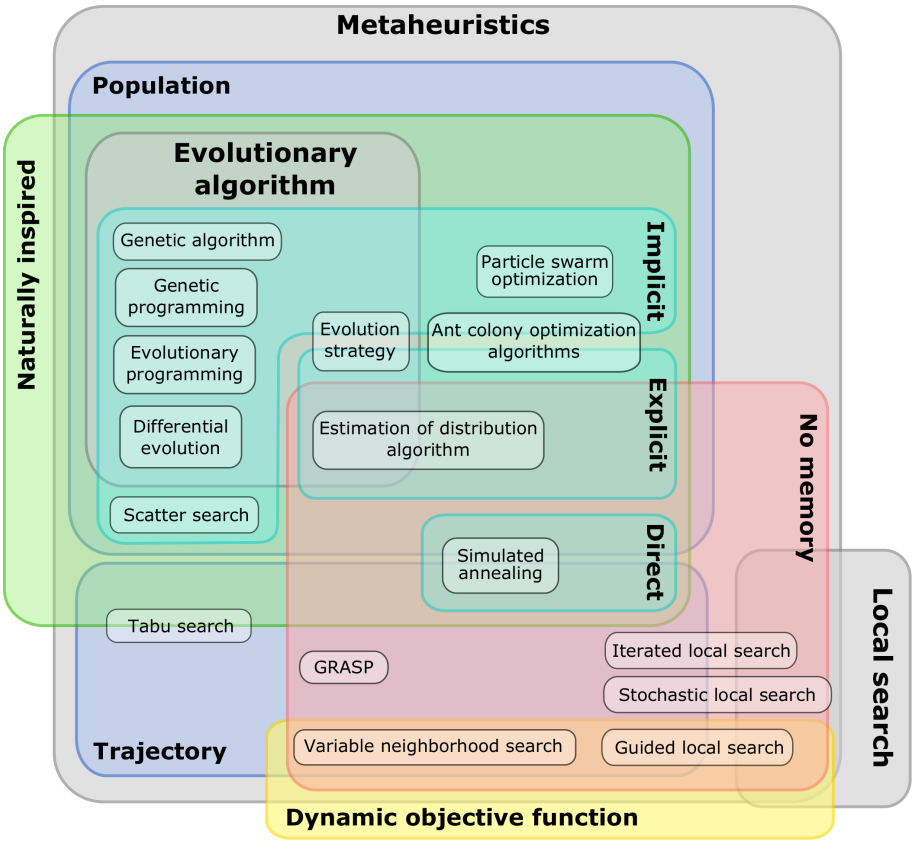


Figure 1.1: Different classifications of metaheuristics. *(Figure author: Johann "nojhan" Dréo, Caner Candan).*

1.1 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are a broad family of search and optimization algorithms that attempt to solve complex problems by mimicking the processes of Darwinian evolution. Among the methods shown in the figure 1.1, genetic programming (GP) is a noteworthy example of EAs.

GP is a branch of genetic algorithms (GAs) and the main difference between GP and GAs is the representation of the solutions. GP evolves computer programs that can have different sizes and shapes, while GAs evolve fixed-length string that represent solution parameters with most of the structure of the final solution already defined by the user. GP has generated a plethora of human-competitive results and applications, including novel scientific discoveries and patentable inventions (Poli et al., 2008a; Koza, 2010). Because its own advantages over other EAs, we focus on this particular method to conduct our research work.

1.2 GP open issues

Even though, the literature on GP shows a progressive increase research devoted to both theory and applications, a series of open issues were clearly stated in the 10th anniversary issue of the main journal in the field (*Genetic Programming and Evolvable Machines*) (O'Neill et al., 2010), among them the following are of particular importance:

OPEN-ENDED EVOLUTION IN GP: *Design an evolutionary system capable of continuously adapting and searching.* It's difficulty ranges from medium to hard. There is difficulty in defining clear criterion for measuring success, and whether or not open-ended evolution is required by GP practitioners.

GP BENCHMARKS: *Is it possible to define a set of test problems that can be rigorously evaluated by the scientific community and then accepted as a more or less agreed upon set of benchmarks for experimental studies in GP?*

It's difficulty ranges from medium to high. Benchmark problems serve as a common language for a community when developing algorithms, and therefore a certain amount of inertia to not adopt better ones exists. Also, since GP is already a fairly complex algorithm, adopting overly simple benchmark problems enables overall lower complexity in the system, albeit at the expense of the significance of results.

FITNESS LANDSCAPES AND PROBLEM DIFFICULTY IN GP: *Identifying how hard a particular problem, or problem instance, will be for some GP system, enabling a practitioner to make informed choices before and during application.* It's a hard issue. Problem difficulty relates not only to the issue of representation, but also to the choice of genetic operators and fitness functions.

GENERALIZATION IN GP: *Defining the nature of generalization in GP to allow the community to deal with generalization more often and more rigorously, as in other ML fields and statistical analysis.* It is weighted as a medium difficulty issue. The hardness of defining generalization in a reliable and rigorous way is common with other ML paradigms. The lack of generalization in a GP system is often due to lack of prior planning by the practitioner.

Even though, these issues are different they are nevertheless closely related. For this reason, the approach taken in this thesis can partially address each of these issues through a mechanism to guide the GP search process that does not focus explicitly on the objective.

1.3 Mechanisms to guide the search

The mechanism to drive the search in an evolutionary algorithm is traditionally through a fitness function which typically is similar to the objective or cost function used to evaluate the solutions based on the problem objective. But there is no technical reason that prevents the fitness function to be different from the objective function.

For instance, a trivial example of using different functions to drive the search can be observed in a random search; where a random function is used to generate solutions and a different function (the objective function) is used to choose the best solution.

In this research, we highlight a conceptual difference between natural and artificial evolution. Nature seems to not follow any objective at all, while artificial evolution is mostly seen as an objective-based process.

According to the Darwinian approach, evolution by natural selection is a process where individuals from current species that are well adapted to their environment better, will have more chances to survive and to pass on their genetic information to their offspring. In this process, it can be seen that natural selection exerts pressure over time to maintain the kind of individuals with the right features to thrive in their (dynamic) environment.

On the other hand, individuals that are not well adapted are considered inferior and probably will not survive. Following this reasoning, it means that these less fit individuals can either go extinct or must look for a different environmental niche in which to survive. Since in nature we can observe a fierce battle between individuals for natural resources, natural evolution clearly can be seen as a process which tends to produce a large diversity of individuals, in other words it is a divergent process.

This thesis explores a recent and alternative approach to guide the search process that is more akin to the divergent nature of biological evolution, where instead of rewarding solutions that seem to be closer to a given objective, it rewards solutions that are different or unique compared to the solutions found before. In this approach, the more different the solution is, the more novel it is considered to be and this is the fitness score assigned to each solution. This new approach is called Novelty Search (Lehman and Stanley, 2011a) where solutions are described by a representative description of their behavior[1] or main features.

[1] The concept of behavior will be formally explained in the following chapters.

NS is best suited for deceptive problems[2], which are closely related with the difficulty of the problem. NS mainly has been used in the field of evolutionary robotics where the behavioral characterization required to implement NS is intuitive. Furthermore, the underlying evolutionary algorithm mostly used with NS has been the NeuroEvolution of Augmenting Topologies (NEAT) (Stanley, 2004). NEAT is a GP-like algorithm (Trujillo et al., 2014), but the use of NS with standard GP systems has been mostly unexplored in related literature. To the best of our knowledge there is just one previous work using a tree-GP system with NS (Lehman and Stanley, 2010a), and more recently another using grammatical evolution (Urbano and Loukas, 2013), both applied on maze navigation tasks. The study of GP and NS, as a combined approach, is explored in depth for the first time in this research.

1.4 Original Contributions

Main motivation in this research work is to contribute with insight to help tackle some of the open issues in GP by incorporating NS in the search process. Our assumption is that integrating NS with GP it can helps to address these issues with a different perspective, particularly on the manner in which a GP search process should proceed. In particular, this thesis makes the following original contributions:

1. This work is the first to use NS outside the evolutionary robotics domain, particularly to solve machine learning and pattern recognition problems, namely: regression (Martínez et al., 2013), clustering (Naredo and Trujillo, 2013) and classification problems (Naredo et al., 2013b).

2. This work is the first to propose a method to design a set of deceptive classification problems as GP benchmarks (Naredo et al., 2015).

[2] More information about NS can be obtained in; The Novelty Search Users Page, `http://eplex.cs.ucf.edu/noveltysearch/userspage/`

3. This work studies generalization in GP based on NS for evolutionary robotics, showing for the first time that NS generalizes better than traditional search and how the size of the training set influences generalization for both approaches (Naredo et al., 2016c).

4. We propose in this work a behavior-based approach to analyze the performance of a GP program for supervised learning (Trujillo et al., 2013b).

5. One of the limitations in the development of real-world GP solutions is the issue of bloat, which is the evolution of unnecessarily large solutions, and according with the fitness-causes-bloat theory this is caused by precisely the fitness function. In this work we found that in some problems, using NS with GP which does not use an objective-based fitness function, the average program size of the evolving population does not increase (Trujillo et al., 2013a).

6. We propose an extension of the progressive minimal criteria NS (PMCNS), named as MCNS_{bsf}, which considers a dynamical threshold based on the best-so-far (bsf) solution (Naredo et al., 2016b).

7. We propose a probabilistic approach to compute novelty named as probabilistic NS (PNS), which eliminates all of the underlying NS parameters, and at the same time reduces the computational overhead from the original NS algorithm (Naredo et al., 2016b).

1.5 Outline of the Dissertation

The dissertation begins with Chapter 2, which introduces the background about GP (Koza, 1992a), explaining two different flavours of GP; tree-based GP and grammatical evolution. Furthermore, we address the main search spaces used in GP, and present a case study to explain how a tree-based GP can be used to solve a complex real-world

problem, in this case disparity map estimation for stereo computer vision.

Chapter 3 introduces the notion of deception, which is frequently present in real-world problems and closely related with the problem difficulty, giving some examples about deceptive functions and deceptive navigation tasks. Furthermore, a method to design synthetic classification problems with deceptive fitness landscape is presented (Naredo et al., 2015).

Chapter 4 presents the NS algorithm (Lehman and Stanley, 2011a), and its unique approach to search inspired by natural evolution's open-ended property. We highlight that traditional evolutionary search is driven by a given objective, but NS on the other hand, drives the search by rewarding the most different solutions. Though, it may sound strange and counter intuitive, NS has shown that in some problems ignoring the objective can outperforms traditional search. The reason for this phenomenon is that sometimes the intermediate steps to the goal do not resemble the goal itself (Stanley and Lehman, 2015).

Furthermore, the related work is presented analysing different previous versions of NS. Two new versions of NS are proposed; MCNS$_{bsf}$ and PNS (Naredo et al., 2016b). The first version is an extension of the progressive minimal criteria NS (PMCNS) (Gomes et al., 2012). The second one is a probabilistic approach to compute novelty, this last proposal has the advantage that it eliminates all of the underlying NS parameters, and at the same time reduces the computational overhead from the original NS algorithm while achieving the same level of performance.

In Chapter 5 a first and original case study of applying NS in a GA-based search is presented to synthesize topologies of current follower (CF) circuits. Experimental results show twelve CFs synthesis generated by GA-NS, and their main attributes are summarized. This work confirms that NS can be used as a promising alternative in the field of automatic circuit synthesis (Naredo et al., 2016a).

Chapter 6 studies the problem of generalization in evolutionary robotics, using a navigation task for an autonomous agent in a 2D environment. The learning system used is a GE system based on NS, which

is a different flavour of GP. In particular, this Chapter studies the impact that the training set has on the performance of the learning process and the generalization abilities of the evolved solutions. Experimental results clearly suggest that NS improves the generalization abilities of the GE system (Urbano et al., 2014a; Naredo et al., 2016c).

Chapter 7 presents a GP system based on NS for the classical GP problem of symbolic regression. In this respect, the paper represents the first attempt to apply NS in this domain, one of the most common engineering and scientific problems. A behavioral descriptor was proposed called ϵ-descriptor in order to properly describe the performance that each GP individual exhibits with respect to the rest of the population. Results are encouraging and consistent with recent proposals that expand the use of the NS algorithm to other mainstream areas (Martínez et al., 2013).

Chapter 8 presents GP system based on NS to search for data clustering functions. A domain-specific behavioral descriptor was proposed named as cluster descriptor (CD). Results show that for simple problems the exploratory capacity of NS is mostly unexploited, but performance is improved in more difficult cases (Naredo and Trujillo, 2013).

Chapter 9 address the supervised classification problem through a GP system based on NS. In this Chapter the proposed new versions of NS presented in Chapter 4, MCNS$_{bsf}$ and PNS, are used to solve binary and multi-class classification problems. Experimental results are evaluated based on two measures, solution quality and average size of all solutions in the population. In terms of performance, results show that all NS variants achieve competitive results relative to the standard OS approach in GP (Naredo et al., 2013b, 2016b).

Lastly, Chapter 10 presents the general conclusions about this research work.

1.6 Summary of publications

conference papers:

1. Naredo, E., Dunn, E., and Trujillo, L. (2013a). Disparity map estimation by combining cost volume measures using genetic programming. In Schütze, O., Coello Coello, C. A., Tantar, A.-A., Tantar, E., Bouvry, P., Del Moral, P., and Legrand, P., editors, *EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation II*, volume 175 of *Advances in Intelligent Systems and Computing*, pages 71–86. Springer Berlin Heidelberg

2. Naredo, E., Trujillo, L., and Martínez, Y. (2013b). Searching for novel classifiers. In *Proceedings from the 16th European Conference on Genetic Programming, EuroGP 2013*, volume 7831 of *LNCS*, pages 145–156. Springer-Verlag.

3. Naredo, E. and Trujillo (2013). Searching for novel clustering programs. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13. ACM.

4. Martínez, Y., Naredo, E., Trujillo, L., and López, E. G. (2013). Searching for novel regression functions. In *IEEE Congress on Evolutionary Computation*, pages 16–23.

5. Trujillo, L., Spector, L., Naredo, E., and Martínez, Y. (2013b). A behavior-based analysis of modal problems. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation Companion*, GECCO Companion '13, pages 1047–1054.

6. Trujillo, L., Naredo, E., and Martínez, Y. (2013a). Preliminary study of bloat in genetic programming with behavior-based search. In *EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation IV*, volume 227 of *Advances in Intelligent Systems and Computing*, pages 293–305. Springer International Publishing.

7. Martínez, Y., Trujillo, L., Naredo, E., and Legrand, P. (2014). A comparison of fitness-case sampling methods for symbolic regression with genetic programming. In *EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation*

V, volume 288 of *Advances in Intelligent Systems and Computing*, pages 201–212. Springer International Publishing.

8. Trujillo, L., Muñoz, L., Naredo, E., and Martínez, Y. (2014). *Genetic Programming: 17th European Conference, EuroGP 2014, Granada, Spain, April 23-25, 2014, Revised Selected Papers*, chapter NEAT, There's No Bloat, pages 174–185. Springer Berlin Heidelberg, Berlin, Heidelberg.

9. Urbano, P., Naredo, E., and Trujillo, L. (2014a). Generalization in maze navigation using grammatical evolution and novelty search. In *Theory and Practice of Natural Computing*, volume 8890 of *Lecture Notes in Computer Science*, pages 35–46. Springer International Publishing.

10. Naredo, E., Trujillo, L., Fernández De Vega, F., Silva, S., and Legrand, P. (2015). Diseñando problemas sintéticos de clasificación con superficie de aptitud deceptiva. In *X Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB 2015)*, Mérida, España.

JOURNAL ARTICLES: PUBLISHED AND SUBMITTED.

1. Naredo, E., Urbano, P., and Trujillo, L. (2016c). The training set and generalization in grammatical evolution for autonomous agent navigation. *Soft Computing*, pages 1–18.

2. Martínez, Y., Naredo, E., Trujillo, L., Pierrick, L., and López, U. (2016). A comparison of fitness-case sampling methods for genetic programming. *Submitted to: Journal of Experimental & Theoretical Artificial Intelligence, currently working with the reviewers' comments*.

3. Naredo, E., Trujillo, L., Legrand, P., Silva, S., and Muño, L. (2016b). Evolving genetic programming classifiers with novelty search. *To appear: Information Siences Journal*.

4. Naredo, E., Duarte-Villaseñor, M. A., García-Ortega, M. d. J., Vázquez-López, C. E., and Trujillo, L. (2016a). Novelty search for the synthesis of current followers. *Submitted to: Information Siences Journal*.

2

Genetic Programming

ABSTRACT — Genetic programming (GP) is a powerful tool that is widely used to solve interesting and complex real-world problems. This chapter introduces an overview of GP, focusing mainly on the evaluation process, and particularly using the tree-representation. We highlight the difference between fitness and objective function, which is relevant to this research work. Furthermore, we aboard the different search spaces which are concurrently sampled during the search process. On the other hand, we go deeper in the concept of behavior as a form to observe the performance of a GP program, proposing a variable scale containing all different behaviors, from the a low level to a higher description about the performance observed by each solution.

2.1 Introduction

Genetic programming (GP) is a subset of evolutionary computation and a generalization of genetic algorithms (GA) that provides a noteworthy proposal to address the central questions in computer science stated in the 1950s by Arthur Lee Samuel (Koza, 1992a); "how can computers be made to do what needs to be done, without being told exactly how to do it?". GP evolves knowledge and behavior from a high-level problem statement represented by a population of programs, this process is also called program synthesis or automatic program induction

(Koza et al., 2000, 2008; Koza, 2010). For this reason GP has become shorthand for the generation of programs, code, algorithms and structures (O'Neill et al., 2010). GP can genetically breed computer programs capable of solving, or approximately solving, a wide variety of problems from a wide variety of fields (Koza, 1992a).

GP is a domain-independent evolutionary method intended to solve problems without requiring the user to know or specify the form or structure of the solution in advance (Poli et al., 2008b). GP is inspired in natural genetic operations, applying similar operations to computer programs, such as crossover (sexual recombination), mutation and reproduction. Computer programs can be represented in several ways, but since trees can be easily evaluated in a recursive manner this is the traditional representation used in the literature (Koza, 1992a).

GP-trees are composed by two different types of nodes known as functions and terminals. Function are internal tree nodes that represent the primitive operations used to construct more complex programs, such as standard arithmetic operations, programming structures, mathematical functions, logical functions, or domain-specific functions (Koza, 1992a). Terminal nodes are at the leaves of the GP-trees and usually correspond to the independent variables of the problem, zero-arity functions or random constants.

There are other GP versions that use different representations, such as linear genetic programming (Brameier and Banzhaf, 2010), cartesian GP (Peter, 2000), MicroGP[1] or in short μGP (Squillero, 2005), and some authors have even developed special languages for GP-based evolution such as Push, which is used to implement the PushGP system (Spector and Robinson, 2002) .

Moreover, GP is a very flexible technique, a characteristic that has allowed researchers to apply it in various fields and problem domains (Koza et al., 2000, 2008; Koza, 2010). In computer vision, for instance, GP has been used for object recognition (Howard et al., 1999; Ebner, 2009; Hernández et al., 2007), image classification (Krawiec, 2002; Tan

[1] More information about MicroGP can be found on the following web-pages; `http://www.cad.polito.it/research/Evolutionary_Computation/MicroGP.htm`, and `http://ugp3.sourceforge.net/`

et al., 2005), feature synthesis (Krawiec and Bhanu, 2005; Puente et al., 2011), image segmentation (Poli, 1996; Song and Ciesielski, 2008), feature detection (Trujillo et al., 2008c, 2010; Olague and Trujillo, 2011) and local image description (Pérez and Olague, 2008; Perez and Olague, 2009). GP has been successfully applied to computer vision problems since it is easy to define concrete performance criteria and the development or acquisition of data-sets is now trivially done. However, most problems in computer vision remain open and most successful proposals rely on state-of-the-art machine learning and computational intelligence techniques.

2.2 Genetic Programming Overview

As stated previously, GP is a generalization of GAs, where the basic GA process is used to write computer programs (Koza, 1992a), from the artificial evolution perspective GP is a program that is able to write other computer programs. The GA operations of mutation, reproduction (crossover) and cost calculation require only minor modifications. GP is a more complicated procedure because it must work with the variable length structure to represent programs or functions (Haupt and Haupt, 2004).

Like other EAs, GP starts with a population of randomly created individuals, which in this case represent programs or mathematical functions. Inspired in the Darwinian principle of natural selection GP traditionally applies the maxima of "survival of the fittest" to guide the search. Every generation, the individuals in the population are evaluated through an objective function to determine their fitness, usually measuring how close a given computer program is to achieving the desired goal.

The fitness score is then used in the selection process to choose individuals as parents to generate the next generation of candidate solutions. Chosen parents are passed through genetic-like operations to share genetic information (between two parents) or to mutate any of their genetic material.

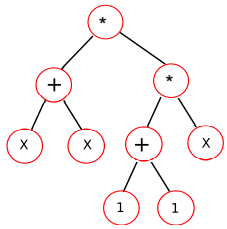
There are three termination criteria traditionally used in GP, first two are related with respect to an upper limit reached: a maximum number of either generations or evaluations of the fitness function. The third criterion is to stop the search when the chance of improvement in the next generations is excessively low. After at least one of the termination criteria is satisfied, the best program produced during the whole run, usually named as the best-so-far, is returned as the result of the run.

2.3 Nuts and Bolts of GP

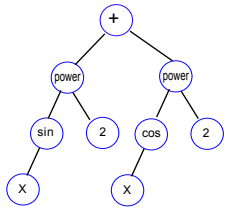
GP evolves a randomly generated initial population of programs into a new population of programs, through genetic operations applied on every where each iteration is referred as a generation using the evolutionary terminology. Hopefully the evolved programs will be better than their predecessors, but since it is a stochastic process similarly as used in nature there is no guarantee of finding the optimal solution. Since randomness is an essential feature of GP, there are some drawbacks that deterministic methods do not exhibit.

Computer programs in GP traditionally are represented as tree structures with nodes and leafs. GP tree nodes are program instructions named as functions and GP tree leafs are the features (dimensions) of the dataset, we call these elements terminals. For instance, a data point can be given by $\mathbf{x}_i = (d_1, ..., d_k, ..., d_n)$ where n data elements d_k are the problem features. The set of functions can be composed by the basic arithmetic, trigonometric, or by any other complex function or even by a combination between them, for instance: $plus, minus, times, divide, sin, cos, log, sqrt, power, abs$.

The overall process of evolving programs with standard GP is shown in the algorithm 1. The main difference that can be highlighted between the algorithms found in the literature is that the fitness f and objective function g traditionally is the same, but there is no restriction to be different as shown in Algorithm 1. The general flow of a GP search is presented in Figure 2.2.



GP program K_1
Prefix notation:
$K_1(x) = \text{times}(\text{plus}(x,x),\text{times}(\text{plus}(1,1),x))$.
Infix notation:
$K_1(x) = (x+x)*((1+1)*x))$.



GP program K_2
Prefix notation:
$K_2(x) = \text{plus}(\text{power}(sin(x),2),\text{power}(cos(x),2))$.
Infix notation:
$K_2(x) = sin^2 x + cos^2 x$.

Figure 2.1: Example of programs (expressions) K_1 and K_2, evolved by GP using the previous set of terminals and functions. On the left is depicted the tree representation, while on the right it can be seen both prefix and infix mathematical notation for each GP program.

2.4 Search Spaces in GP

It is known that many EAs concurrently sample three different spaces during a search: genotypic, phenotypic and objective space. In the case of traditional tree-based GP, the genotypic space corresponds to the syntactic space and selective pressure is given in objective space by the objective function designed for the problem. Typically, the genotype of a computer program K is given by its syntax (GP-tree), as shown in Figure 2.3. Syntactic space contains all possible computer programs which can be composed by the set of functions and terminals used in the search. The syntax receives as input the information from the vector of terminals $\mathbf{x}$. In GP, the phenotype corresponds to the algorithm which is implicitly implemented by the syntax; though, the phenotype is rarely analyzed explicitly by most GP systems (McDermott et al., 2011) since extracting it is not a trivial task.

Algorithm 1 Genetic Programming Algorithm.

Input: elitism rate; e, cross rate; x, mutation rate; m
Set of functions; S, Set of terminals; T
Fitness function; f
Objective function; g
Termination criteria; number of generations; n, threshold; h
Output: The best-so-far solution scored with the objective function g, found at any generation is designated as the result.
Procedure:
Initial generation
$t \leftarrow 0$;
Create initial population
$P(t=0) \leftarrow generate(S,T)$;
Execute and evaluate each initial program
$evaluate(P(t=0))$
Assign fitness according with the fitness function
$F_p(t=0) \leftarrow f(P(t=0))$
Assign quality score according with the objective function
$G_p(t=0) \leftarrow g(P(t=0))$
while not meet termination criteria n,h **do**
  i) Parents to apply genetic operations
  a. Best existing programs taken as parents.
  $P_p(t) \leftarrow best(P(t),e)$
  b. Select parents according the fitness score
  $P_p(t) \leftarrow selectParents(P(t),f)$;
  c. Create new programs as parents if necessary
  $P_p(t) \leftarrow generate(S,T)$
  ii) Create children by mutation.
  $P_{ch}(t) \leftarrow mutate(P_p(t),m)$
  iii) Create children by crossover (sexual reproduction)
  $P_{ch}(t) \leftarrow reproduction(P_p(t),x)$
  Execute and evaluate each evolved (children) program
  $evaluate(P_{ch}(t))$
  Assign fitness according with the fitness function
  $F_p(t) \leftarrow f(P_{ch}(t))$
  Assign quality score according with the objective function
  $G_p(t) \leftarrow g(P_{ch}(t))$
  Assign the new population for the next generation
  $P_p(t+1) \leftarrow P_{ch}(t)$
end while

While these spaces have been the focus of most GP research, other spaces have recently been used to develop new GP-based algorithms. For instance, the computer program K produces an output vector, named recently in GP literature as semantics (Moraglio et al., 2012).

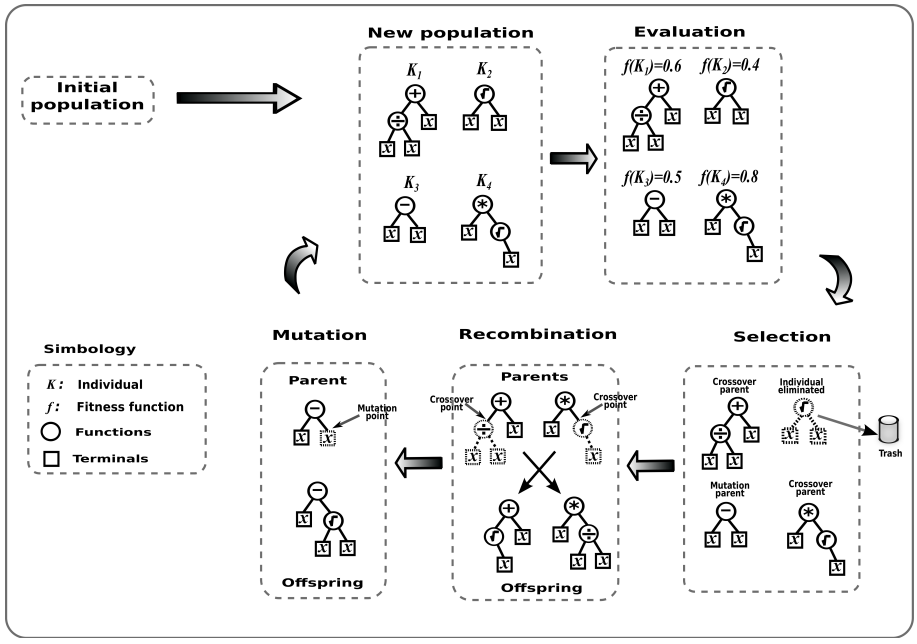


Figure 2.2: Block diagram depicting the main processes involved in a tree-based GP search

Afterwards, the output semantics are transformed into a single quality measure by the objective function O which assigns fitness to K.

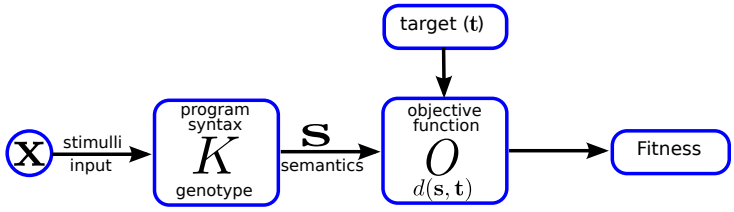


Figure 2.3: The traditional program evaluation process used in GP, where K is the genotype of a computer program, $\mathbf{s}$ is the program output vector called semantics, the objective function O which typically is defined as a distance $d(\mathbf{s},\mathbf{t})$ between semantics ($\mathbf{s}$) and the expected target ($\mathbf{t}$), obtaining then the fitness score assigned to the program K.

Formally, semantics can be defined as in (Moraglio et al., 2012). Given a set of n fitness cases in a training set $T = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$, the semantics of a program K_j with $j = 1, ..., m$ is the corresponding set of outputs it produces, expressed as $\mathbf{s} = (K_j(\mathbf{x}_1), ..., K_j(\mathbf{x}_n))$. If we consider real-valued outputs, then $\mathbf{s} \in \mathbb{R}^n$. In this case, the objective function is usually defined as a distance $d(\mathbf{s}, \mathbf{t})$ between a program's semantics $\mathbf{s}$ and the desired target $\mathbf{t} = (y_1, ..., y_n)$, see Figure 2.3. The most interesting feature about semantic space is that it defines a unimodal fitness landscape. Through semantics, researchers have proposed new search operators that help improve the GP search. For instance, Beadle and Johnson (Beadle and Johnson, 2008) proposed Semantically Driven Crossover (SDC) that promotes semantic diversity. Uy et al. (Uy et al., 2011b) proposed four different variants of semantically aware crossover operators for symbolic regression. Moraglio et al. (Moraglio et al., 2012) proposed Geometric Semantic GP (GSGP), designing special syntactic search operators that generate offspring that are mapped to geometrically bounded regions in semantic space. Recently, GSGP has been enhanced with local search mechanisms that improve performance and convergence (Castelli et al., 2015). However, in some domains the target is known beforehand, (the optimal output -semantics- might not be known), instead what is measured is the effect that the output has on an external system, what we are referring to as the domain context.

For example, consider the static range selection (SRS) GP classifier (Zhang and Smart, 2006) (discussed in Section 9.1), which functions as follows. For a two class $\{\omega_1, \omega_2\}$ problem and real-valued GP outputs, the SRS classifier applies a simple classification rule $\mathcal{R}$: if the program output for input pattern $\mathbf{x}$ is greater than a threshold r, then the pattern is labeled as belonging to class ω_1, otherwise it is labeled as a class ω_2 pattern. In this case, while the semantics of two programs might be different (maybe substantially), they can still achieve the same classification performance.

Indeed, these issues have led some authors to pose that the different mappings found and sometimes used in the GP process are the representation of different phenotypes mappings between them. For

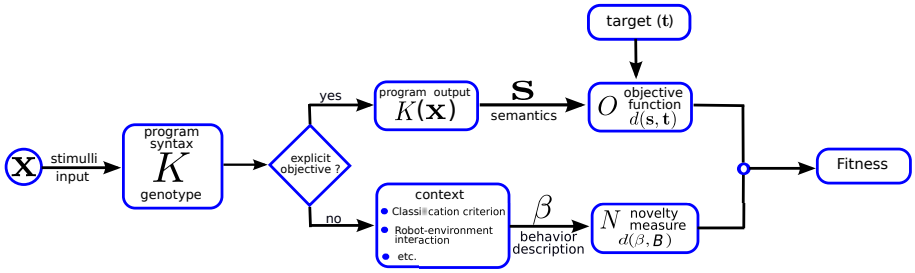


Figure 2.4: The basic program evaluation process used in GP considering the choice of using the objective explicitly or implicitly. First option (*yes*), computes fitness through a distance function between the computer program **s** (semantics) and the expected target **t**, named as objective-based fitness. Second option (*no*), computes fitness through a behavior description β, then applying a distance function between each current behavior β and a set of behaviors B composed by current behaviors and previous behaviors to assign a novelty score as fitness, named as novelty-based fitness.

instance, (McDermott et al., 2011) suggests that the behavior of each component mapping can be studied separately. In this work, however, we take advantage of recent findings gained from the field of evolutionary robotics (ER), to focus on the aggregate behavior of all these mappings and analyze the total effect that a given program has on a specific problem. The concept of behaviors has been widely used in robotics, particularly emphasized in the seminal works by Brooks from the 1980's (Brooks, 1999), making it easy to integrate this concept in ER. In behavior-based robotics, for instance, an individual's behavior is described at a higher level of abstraction than the semantics of a particular controller, accounting not only for program output but the context in which the output was produced. The relation between input and semantic space can be injective or non-injective, while behaviors can allow for multivalued mappings or many-to-many relations between inputs and behavior space.

In ER, where GP can be used to evolve controllers that interface directly with a robot's actuators, the fitness functions used normally

account for the observed high-level interactions between the robot and its environment. In fact, in a broad survey of types of fitness functions used in ER, Nelson et al. (Nelson et al., 2009) found that much of the research introduces *a priori* human knowledge when selecting a fitness function to solve a given problem. Nelson et al. group the fitness functions into seven classes, called training data, behavioral, functional incremental, tailored, environmental, competitive and aggregate fitness functions, ordered based on the amount of *a priori* knowledge incorporated into the function. The first class of fitness functions coincides with the most common approach taken in GP, where training data is used, the highest amount of *a priori* knowledge about the problem, the same as depicted in Figure 2.3. However, as Nelson et al. stated, this type of fitness functions require specific knowledge of what should be the optimal output, something that in many cases is not feasible or might be even unnecessary, as we argued in the example of the SRS classifier. Therefore, all other classes of fitness functions in ER, each to a different extent, incorporate the concept of behavior.

Since we can find some disagreement about the interpretation of the concept of behavior that we can find in different areas, we agree with the definition given in (Levitis et al., 2009) (p. 108) from the behavioral biology perspective, which states that a "behavior is the internally coordinated responses (actions or inactions) of whole living organisms (individuals or groups) to internal and/or external stimuli, excluding responses more easily understood as developmental changes".

Considering the above definition, in this work we understand the concept of behavior as a measurable description about the internally coordinated external responses of a computer program K to internal and/or external stimuli $\mathbf{x}$ within a given environment or context. The behavior produced by a particular solution K is captured by a domain dependent descriptor β. In the ER case, context is given by the robot morphology and parts of the environment that cannot be sensed, while the inputs are the robot sensors and the outputs of the controller interface directly with the actuators. In this case the robot behavior descriptor can include such quantities as the robot position (Lehman and Stanley, 2008), the robot velocity (Nelson et al., 2009) or patterns gen-

erated in the robot's path (Trujillo et al., 2008b). Conversely, for the classification problem described above, the context is provided by the specified classification rule $\mathcal{R}$, while one way to describe a classifier behavior can be related to the accuracy of the classifier on the training instances. Afterward, the observed behavior β can be used to compute fitness by a function that considers the objective either explicitly or implicitly.

The explicit approach is the customary way to compute fitness, using an objective function measuring how close the observed behavior is to a particular goal. The implicit approach can be to compute fitness based on the novelty or uniqueness of each behavior, such as in NS. The concept of behavior is graphically depicted in Figure 2.4, along with the manner in which behavioral information is included in the computation of the objective and fitness functions.

Figure 2.5: Conceptual view of how the performance of a program can be analyzed. At one extreme we have objective-based analysis, a coarse view of performance. Semantics-based analysis lies at another extreme, where a high level of detail is sought. Finally, behavior analysis provides a variable scale based on how the problem context is considered.

We propose that the behavior-based perspective should be seen as part of a scale of different forms of analyzing performance. In essence, the objective function, semantics, and behaviors are different levels of abstraction of a program's performance as shown in the Figure 2.5. At

one extreme form of analysis, an objective function provides a coarse grained look at performance, a single value (for each criterion) that attempts to capture a global description of performance. At the other end of the analysis scale, semantics describe program performance in great detail, considering the raw outputs. On the other hand, behavior descriptors should be considered to be situated between fitness and semantics, providing a finer or coarser level of description depending on how behaviors are meaningfully characterized within a particular domain.

Moreover, the behavior-based approach allows us to modify the fitness computation in other ways. In this work we study the NS algorithm (Lehman and Stanley, 2008; Stanley and Lehman, 2015) that focuses selective pressure based on the concept of solution novelty. In particular, since NS was conceived for ER problems it measures novelty in behavioral space. This work extends previous NS-based contributions, to apply NS in the common machine learning task of supervised classification.

2.5 GP Case Study: Disparity Map Estimation for Stereo Vision

This section intends to provide a real-world example of how GP is used to solve a real-world complex task. In particular, we address the problem of dense stereo correspondence, where the goal is to determine which image pixels in both images are projections of the same 3-D point from the observed scene. The proposal in this work is to build a non-linear operator that combines three well known methods to derive a correspondence measure that allows us to retrieve a better approximation of the ground truth disparity of stereo image pair. To achieve this, the problem is posed as a search and optimization task and solved with genetic programming (GP). Experimental results on well known benchmark problems show that the combined correspondence measure produced by GP outperforms each standard method, based on the mean error and the percentage of bad pixels. In conclusion, this

work shows that GP can be used to build composite correspondence algorithms that exhibit a strong performance on standard tests.

In this work, the hypothesis is that three is better than one when it comes to correspondence algorithms. In other words, we wish to find an operator K that takes as input the cost volume produced by the three correspondence methods discussed above (CV_{SAD}, CV_{NCC} and CV_{BTO}) and generates as output a new cost volume CV_O which is optimal with respect to the accuracy of the respective disparity map. This task is posed as a supervised learning problem, where the ground truth disparity is known for a set of N images, the goal is to minimize the error of the disparity map given by $K(CV_{SAD}, CV_{NCC}, CV_{BTO})$ with respect to the ground truth.

This chapter formulates the above problems in terms of search and optimization, with the goal of automatically generating an optimal operator K. The proposal is to solve this problem using a GP search, where the terminal (input) elements for each individual are the cost volumes CV_{SAD}, CV_{NCC} and CV_{BTO}, and the output is CV_{GP}, the best possible approximation to CV_O found by the evolutionary process.

2.5.1 *Experimental Configuration and Results*

The proposal developed in this work is to evolve an operator that combines three well known correspondence measures, using Genetic Programming.

The images used on this work are taken from the Middlebury Stereo Vision Page[2], which are standard benchmark tests for stereo vision algorithms. The dataset used contains six pairs of stereo images named: Barn1, Barn2, Bull, Poster, Sawtooth, and Venus. The left image from each pair is shown in Figure 2.6 for each data set. Moreover, for each image pair the dataset includes both left-right and right-left pixel accurate disparity maps as ground truth.

[2] Images available on the Middlebury Stereo Vision Page: `http://www.vision.middlebury.edu/stereo`.

(a) barn1 (b) barn2 (c) bull

(d) poster (e) sawtooth (f) venus

Figure 2.6: Stereo images for experiment tests from Middlebury Stereo Vision Page.

The goal of our experimental work is to asses the quality of the raw disparity map generated by GP, compared with respect to the base methods and the ground truth (GT) of each image pair.

In this work a standard Koza style tree-based GP representation is used, with subtree-crossover and subtree-mutation. The general parameters of the search are given in Table 2.1. Of particular importance is the terminal set, which is given by the cost volumes from the correspondence functions used: Sum of Absolute Differences (SAD), Normalized Cross Correlation (NCC) and Birchfield-Tomasi (BTO), therefore, CV_{SAD}, CV_{NCC}, CV_{BTO}, represent cost volumes from each method and are used as terminals for GP.

To compute each cost volume the search region depends on the minimum and maximum disparity given by the ground truth disparity map of the particular image; therefore, for Barn1 the search region is: [28,131], for Barn2 is: [27,132], for Bull: [29,153], for Poster: [27,161], Sawtooth has: [31,143], and Venus: [24,158]. Furthermore, the neighborhood $M \times N$ considered by each method is given by the number of

Table 2.1: GP system parameters for the experimental tests

Parameter	**Description**
Population size	20 individuals.
Generations	100 generations.
Initialization	*Ramped Half-and-Half*, with 6 levels of maximum depth.
Operator probabilities	Crossover $p_c = 0.8$; Mutation $p_\mu = 0.2$.
Function set	$\{+, -, *, /, \sqrt{\cdot}, sin, cos, log, x^2, \lvert\cdot\rvert\}$
Terminal set	$\{CV_{SAD}, CV_{NCC}, CV_{BTO}\}$ Where CV is the Cost Volume from SAD, NCC, and BTO functions respectively
Bloat control	Dynamic depth control.
Initial dynamic depth	6 levels.
Hard maximum depth	20 levels.
Selection	Lexicographic parsimony tournament
Survival	Keep best elitism

rows N and the number of columns M, and particularly for the experiments the neighborhood size used is 3×3.

Fitness is given by the cost function shown in Equation 2.1, that assigns a cost value to every individual K expression as feasible solution given by GP. The goal is to minimize the error computed between the disparity map from every K expression and the ground truth, therefore

$$f^S(K) = \frac{1}{NM} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} |d^S_{(CV_m)}(i,j) - d^S_{(GT)}(i,j)|, \tag{2.1}$$

where s is the image sequence used, N is the total number of image pixels, (i,j) represents the pixel position on the matrix, $d_{CV^s_m}$ is the disparity map computed using the method m (SAD, NCC, or BTO) for the s image sequence.

First three image sequences; Barn1, Barn2, and Bull, were selected as training data and the other three images sequences; Poster, Sawtooth, and Venus, as testing data.

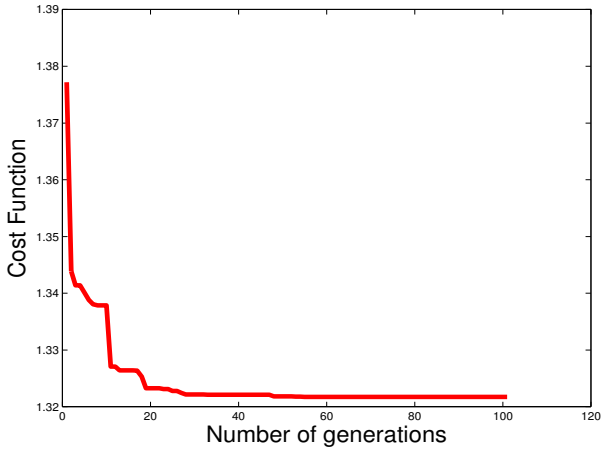


Figure 2.7: Convergence Graphic of the best GP run result

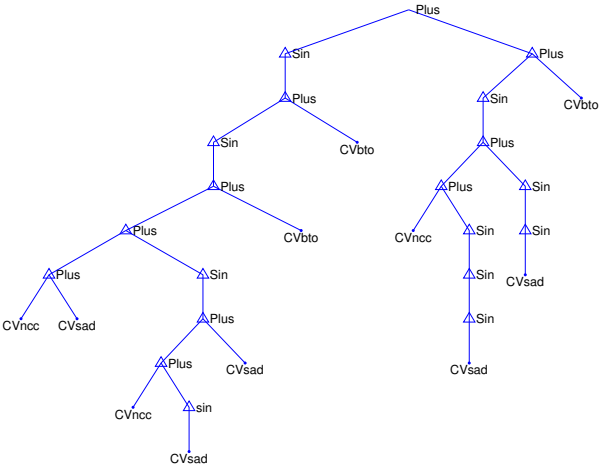


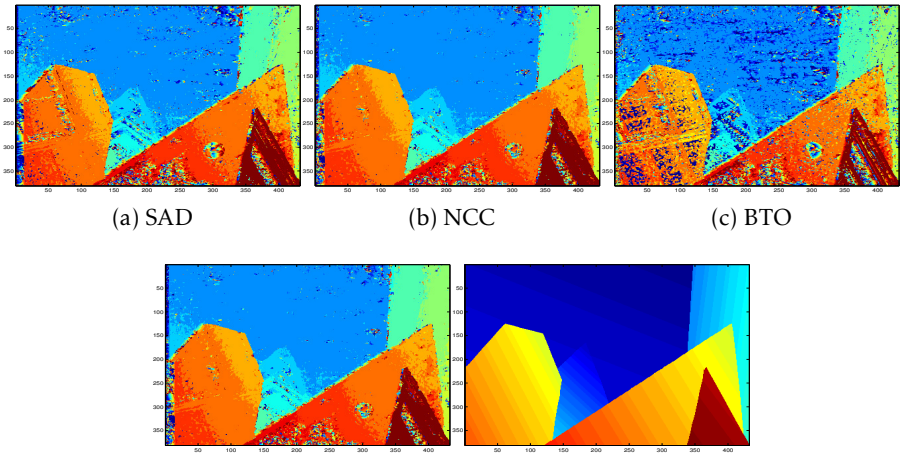
Figure 2.8: Function tree of the best GP run result.

Besides the error evaluation explained in Eq. 2.1, we use to compare the error function showed on Middlebury University page that compute bad pixel percentage, given by

$$BP = \frac{1}{N} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (|d^s_{CV_m}(i,j) - d^s_{GT}(i,j)| > \delta_d), \tag{2.2}$$

Table 2.2: Comparison results for the Barn1 image (bold indicates best result).

Error method	**SAD**	**NCC**	**BTO**	**GP**
Mean of Abs. Differences	1.483379	1.344098	1.834855	**1.311435**
Bad Pixel Percentage	1.021885	0.805163	1.397609	**0.771604**



(a) SAD (b) NCC (c) BTO

(d) GP (e) Ground Truth

Figure 2.9: Disparity maps comparison for the Barn1 image.

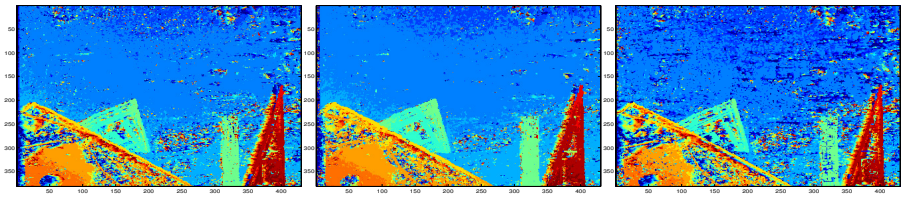
where the threshold for a bad evaluation δ_d is a disparity error tolerance. For the experiments we use $\delta_d = 1.0$, since it is the threshold value used on the reference Middlebury web page.

2.5.1.1 *Experimental Results*

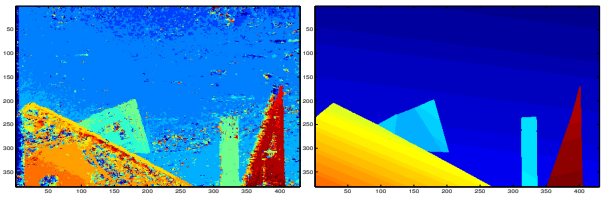
We compare the SAD, NCC, and BTO matching costs against the GP cost volume. The GP search was executed over 20 times, in all cases achieving similar performance. Figure 2.7 presents the convergence plot for the best run, which shows how the fitness of the best individual evolved over each generation. Figure 2.7 presents the best GP so-

Table 2.3: Comparison results for the Barn2 image (bold indicates best result).

Error method	SAD	NCC	BTO	GP
Mean of Abs. Differences	1.959252	1.660405	2.038269	**1.640279**
Bad Pixel Percentage	1.579143	1.248889	1.774773	**1.226271**



(a) SAD (b) NCC (c) BTO



(d) GP (e) Ground Truth

Figure 2.10: Disparity maps comparison for the Barn2 image.

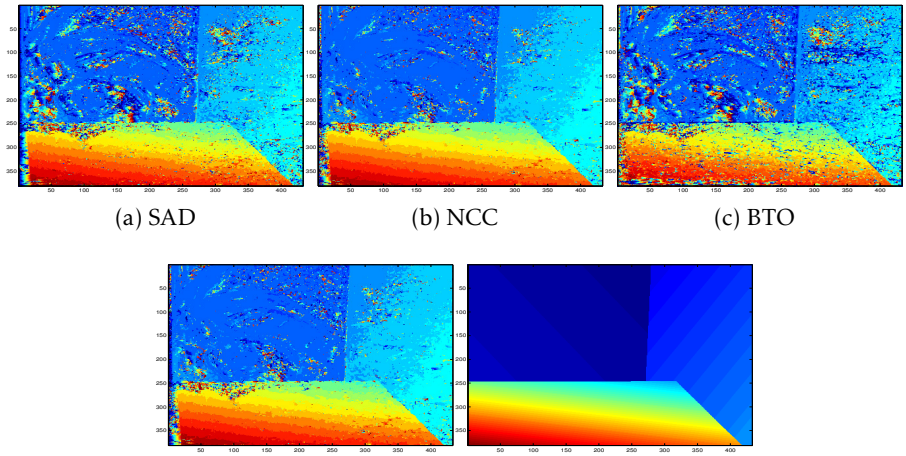
lution found expressed as a program tree. Moreover, the mathematical expression of the best GP individual K is given by

$$f^*_{gp} = sin\{sin[sin(sin\,CV_{SAD} + CV_{SAD} + CV_{NCC}) + CV_{SAD} + CV_{NCC} + CV_{BTO}] + CV_{BTO}\} + sin(sin^3 CV_{SAD} + sin^2 CV_{SAD} + CV_{NCC}) + CV_{BTO}\,. \tag{2.3}$$

Finally, Tables 2.2 to 2.7 present a comparative analysis of the best GP individual and the three standard methods. Each table shows both the mean of the absolute difference, and the bad pixel percentage errors for each pair of stereo images. In all cases we can note that the GP

Table 2.4: Comparison results for the Bull image (bold indicates best result).

Error method	**SAD**	**NCC**	**BTO**	**GP**
Mean of Abs. Differences	1.785469	1.378297	2.030898	**1.330506**
Bad Pixel Percentage	1.186709	0.703539	1.422414	**0.675807**



(a) SAD (b) NCC (c) BTO

(d) GP (e) Ground Truth

Figure 2.11: Disparity maps comparison for the Bull image.

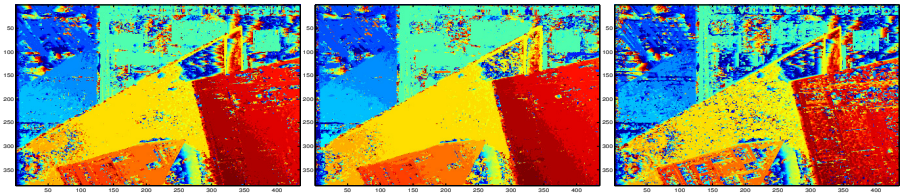
method achieves the best performance. Figures 2-6 present a comparative visual analysis of the best GP individual and the three standard methods, compared with the ground truth. Visually, we can detect the improvement provided by the GP operator K over the standard methods.

2.6 Chapter Conclusions

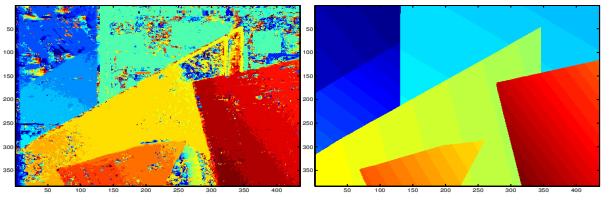
In this chapter a background about GP is presented, focusing on the fitness function as well as in the search spaces. Here, is highlighted that

Table 2.5: Comparison results for the Poster image (bold indicates best result).

Error method	SAD	NCC	BTO	GP
Mean of Abs. Differences	2.432111	1.988050	3.010154	**1.945446**
Bad Pixel Percentage	1.57552	1.218279	1.985391	**1.202187**



(a) SAD (b) NCC (c) BTO



(d) GP (e) Ground Truth

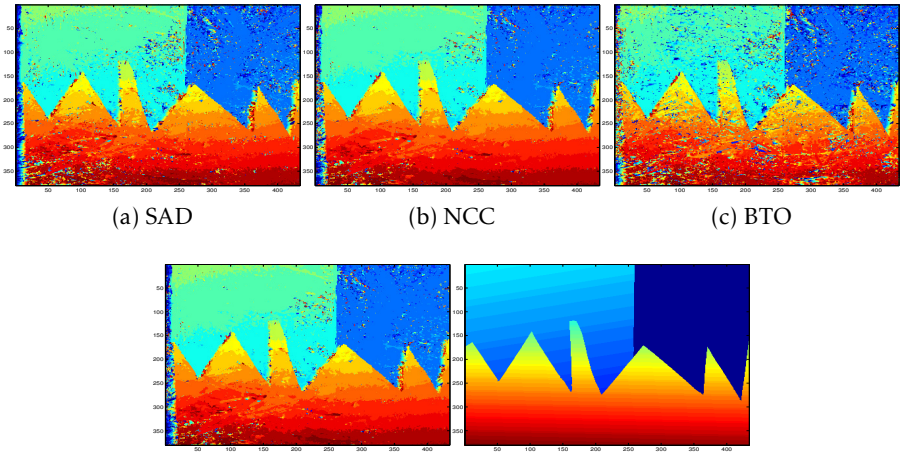
Figure 2.12: Disparity maps comparison for the Poster image.

we can use a different approach to implement a fitness function. The traditional approach is to reward solutions which are closer to the objective. Different approaches to the customary objective-based fitness function has been proposed, one interesting approach will be presented in the Chapter 4.

Furthermore, this chapter studies the problem of dense stereo correspondence using GP. The proposed approach is to combine three well-known similarity measures and derive a composed estimation of the disparity map for a stereo image pair. This task is posed a search and optimization problem and solved with GP. The terminal elements for the GP search is composed by the cost volumes produced by the three

Table 2.6: Comparison results for the Sawtooth image (bold indicates best result).

Error method	**SAD**	**NCC**	**BTO**	**GP**
Mean of Abs. Differences	1.532218	1.433264	1.727435	**1.374315**
Bad Pixel Percentage	1.073662	0.859555	1.169753	1.202187



(a) SAD (b) NCC (c) BTO

(d) GP (e) Ground Truth

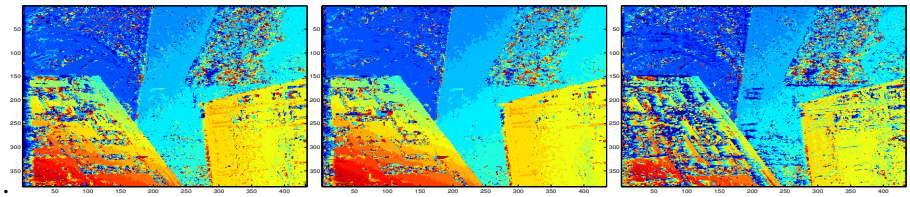
Figure 2.13: Disparity maps comparison for the Sawtooth image.

correspondence methods; SAD, NCC, and BTO. Fitness is based on the error between the estimated disparity map and the ground truth disparity.

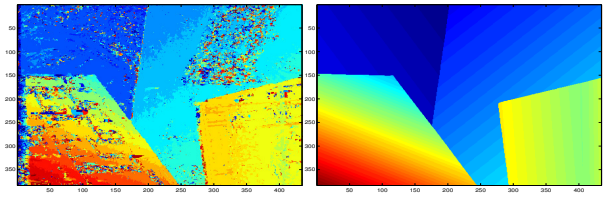
Experimental results for this case study show that the evolved GP operator achieves better performance that the standard methods based on well-known benchmark problems. These results are validated with a set of test cases and an additional performance metric. While these results are an encouraging first step, further work is considering following this topic. For instance, we can add other standard approaches as input to the GP search, such as non-parametric correspondence methods. Moreover, we can use the raw disparity map generated by the GP

Table 2.7: Comparison results for the Venus image (bold indicates best result).

Error method	SAD	NCC	BTO	GP
Mean of Abs. Differences	2.281053	1.916733	2.704778	**1.880739**
Bad Pixel Percentage	1.617716	1.285286	1.804815	**1.267094**



(a) SAD (b) NCC (c) BTO



(d) GP (e) Ground Truth

Figure 2.14: Disparity maps comparison for the Venus image.

operators as raw input for global optimization methods which could allow us to define a higher level fitness evaluation.

3

Deception

abstract — Since the very introduction of genetic algorithms (GAs), it was noted by Goldberg that there is a certain kind of problem which deceives a search process. Rewarding solutions closer to the objective in these deceptive problems, the search will be lead away from the global optimum, getting trapped into regions of local optima. The notion of deception is related with problem complexity. The more complex the problem is, the higher the possibilities that it is a deceptive problem. Most deceptive problems found in the evolutionary computation literature are toy problems, particularly applied on binary representations, and more recently in evolutionary robotics. In this chapter, our intent is to design a set of deceptive classification problems which hopefully could serve as benchmark to test evolutionary or search-based classification strategies.

3.1 Introduction

EAs in general have shown good performance when solving complex problems, but a central issue is the characterization of problems that are difficult to solve, particularly by GAs. A noteworthy attempt was made by (Goldberg, 1989) to explain the ability of GAs in frequently succeeding at finding good solutions; what is called the build-

ing block hypothesis (BBH). In this hypothesis Goldberg states that GAs can identify segments (blocks) of the optimal solution contained in the current solution. Furthermore, Goldberg hypothesises that GAs use these blocks to generate new and better solutions by recombining them or by mutating them, which at the end (mostly) build-up the complete optimal solution.

There are a wide range of benchmark problems to test the performance of EAs, which can be ranked from easy to hard problems. Among the efforts to characterize GA-hard problems some have focused particularly on the notion of *deception*. Since the introduction of the GAs, many deceptive problems have been proposed for binary-coded GAs.

More recently, (Lehman and Stanley, 2008) have proposed several deceptive navigation robotic tasks. But, to the best of our knowledge, there are not deceptive benchmark problems for pattern recognition, particularly deceptive classification problems.

This work introduces a first attempt to design a deceptive classification problem that can be used for benchmarking. The following sections address the notion of deception.

3.2 Deception in the Artificial Evolution

Finding the factors that affect the performance of GAs to solve optimization problems, has been a major interest in the theoretical community (Jones and Forrest, 1995). In general, the performance is measured in terms of their ability to find the closest solution to the global optimum of a given problem. So, according with the BBH, if a GA finds the global optimum, particularly on problems with a binary representation, it is because in fact it correctly identified the correct building blocks for the problem, classifying those problems as easy-GA problems. On the other hand, it is a hard-GA problem if a GA fails to find the global optimum for that particular problem and this means it did not identify the correct building blocks.

Nonetheless, there is another kind of problems, where a GA seems to identify some of the building blocks, apparently pointing to the global optimum, but instead gets trapped into regions of local optima. These are known as deceptive problems since they deceive the search when combining information of the best solutions found so-far, leading the search far away from the global optimum (Rana, 1999). If the GA is not able to find building blocks of low order which are instances of the global solution, then probably it will not be able to find the global optimum, and instead will find a sub-optimal solution (Deb and Goldberg, 1994). The notion of deception was originally introduced by (Goldberg, 1987), in order to better understand what kind of situations are more probable to create difficulties for a GA search.

3.3 Deception in Evolution

The importance of the concept of deception is a controversial topic. In the EA literature we can find a large diversity of arguments, from those that assure that deception is the only important thing that must be considered to classify a problem as a hard-GA problem (Das and Whitley, 1991), to those that claim that deception is neither necessary nor enough to determine the difficulty of a problem (Grefenstette, 1993).

It is clear that some deceptive problems are hard problems, but it is also true that there are other factors that affect the difficulty of a problem, such as: epistasis, multimodality, noise, and spurious correlation among others (Goldberg, 1989; Schaffer et al., 1990; Whitley, 1991; Grefenstette, 1993; Deb and Goldberg, 1993; Jones, 1994; Yang, 2004; Day and Lamont, 2004; Rada-Vilela et al., 2014).

Another well known example that captures the characteristics that make some problems difficult to be solved by GAs focus on the notion of "rugged fitness landscape" (Weinberger, 1990; Horn and Goldberg, 1995). Concerning this (Whitley, 1991) argues that the complexity is related with the difficulty of a problem, then a problem with high ruggedness tends to generate deceptive fitness landscapes. On the other hand,

in the evolutionary robotics, for instance, (Lehman and Stanley, 2008) introduce an intuitive definition of deception related with the difficulty of a problem: *"A deceptive problem is that where an evolutionary algorithm does not find the objective in a reasonable period of time"*.

Unifying both approaches, we can agree that if a problem is so complex that show a very rugged fitness landscape, then an EA will take a long time to find the right path to the global optimum, if possible. So we can conclude as (Rana, 1999) said; deceptive problems correspond with what we normally consider to be difficult problems.

3.4 Deceptive Problems

Typical examples of deceptive problems can be found in the literature, mostly for a binary representation to test GAs. One important deceptive problem for our research work is the trap function, which will help us later to explain the approach used in this chapter to design a deceptive classification problem. On the other hand, another important problem presented here, is the deceptive navigation task, which will allow us to introduce, in the next chapter, a new approach to manage the search to solve deceptive problems.

3.4.1 *Trap Functions*

Trap functions are deceptive functions of unitation. The advantage of trap functions is that they have few parameters but can still be made deceptive (Mengshoel et al., 1998). In order for a function to be deceptive, must have at least two optimums; one being the local optimum, and the other the global optimum. The trap function definition given in (Deb and Goldberg, 1993) is the following: A trap function f for a binary string u, assigns fitness values depending solely on the number of ones, defined as

$$f(u) = \begin{cases} \frac{a}{l-h}(u-h), & h < l \quad \text{If} \quad u \geq h, \\ \\ \frac{b}{h}(h-u), & h > 0 \quad \text{otherwise}, \end{cases} \tag{3.4}$$

where l is the binary string length, a is the global optimum, b is the local (deceptive) optimum, and $h \in (0, l)$ is the location of the slope change which divide the region where the optimum local is located from the region where can be found the global optimum with the binary string of size l containing just ones, an example is shown in Figure **??**. Trap

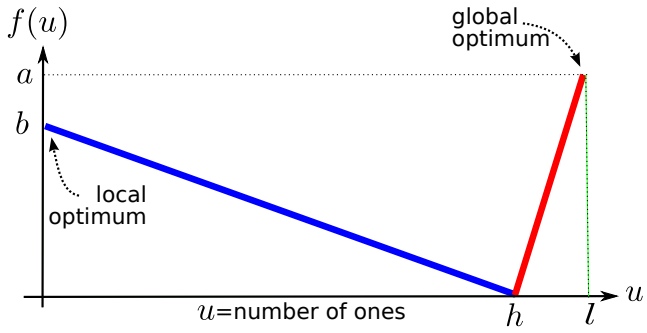


Figure 3.1: Trap function with the global optimum located at point a, and the local optimum is located at point b, u is the variable with the number of ones in the binary string. Varying the parameters of the function the degree of deception can be varied.

functions are highly artificial in two respects: i) they are composed by sub-problems which can be solved independently, and ii) the complement of the deceptive solutions is always the global solution. The trap functions are attractive because they are relatively easy to manipulate and can be difficult to solve by GAs (Mengshoel et al., 1998).

3.4.2 *Scalable Fitness Function*

This function is proposed by (Cuccu et al., 2011a), which is a deceptive function that exhibits plateaus of local minima and multi-modality.

It is used to test the ability of a GA to cross a large region with fitness values that mimic on average a flat region, which is defined as follow

$$f_{l,\omega}(x) = \begin{cases} x^2 l\,, & |x| \leq 1, \\ 1 - \frac{1}{2}\sin^2\left(\frac{\pi\omega(|x|-1)}{l}\right), & 1 \leq |x| \leq l+1\,, \\ (|x|-l)^2\,, & |x| \geq l+1. \end{cases} \tag{3.5}$$

with parameters $l \geq 0$ and $\omega \in \mathbb{N}$. This function is symmetric around the global optimum 0, and is based on the standard parabolic curve, which is cut between $+1$ and -1 constructing a plateau of length l. In

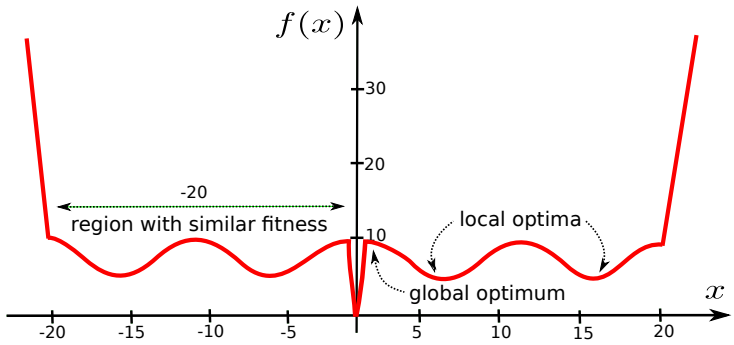


Figure 3.2: Variable deceptive function with parameters: $l = 20$ which defines the plateau length and $\omega = 2$ which defines the amplitude of the sine function.

the region $1 \leq |x| \leq l+1$ where both plateaus are located, the parameter ω controls the amplitude of the sine function, in such a way that if $\omega = 0$ there will be a flat region, while varying ω will introduce a local optima region generated by the sine waves, this controls the deception degree of the function.

3.4.3 *Deceptive Navigation*

In the evolutionary robotics literature (Lehman and Stanley, 2008) proposed a deceptive navigation task similar to the one shown in Figure 3.2. A robot controlled by an artificial neural network (ANN) must

navigate from a starting point to an end point in a fixed time. A reasonable fitness function for such a domain is how close the maze navigator is to the goal at the end of the evaluation. Using this fitness function, the neuro-evolution search was only successful in three out of 40 runs, showing that this is a highly deceptive problem. In the next chapter will be shown how this navigation task is solved in 39 out of 40 runs with a new approach to explore the search space.

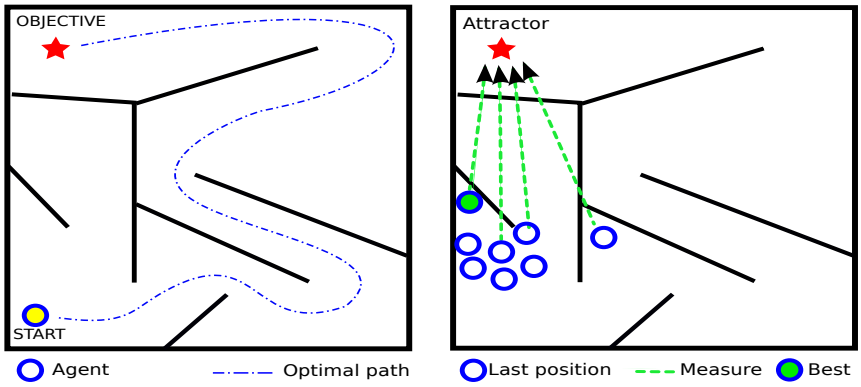


Figure 3.3: Deceptive navigation task. The figure on the left shows the optimal path to the objective. The figure on the right shows how the objective acts as an attractor rewarding the solutions that are closer to the objective, but in fact they are getting trapped into a local optima region.

3.5 Deceptive Classification Problem Design

This section presents the description of a method to design a set of synthetic classification problems, which show deception and hopefully can serve as benchmarks to test classification algorithms (Naredo et al., 2015). The approach to build a deceptive classification problem is through a similar strategy that is used to build a trap function shown in Figure **??**, where we can note two regions, one smaller region where the global optimum is located, and another large region where is found

the local optimum. If we observe the trap function shown in Figure **??**, we can note that the smaller the region where the global optimum is located, the more difficult will be to reach that region and to find the global optimum.

We can see clearly that for a GA it is more likely to fall in the larger region and getting trapped in the local optimum, even if there is a solution that falls in the smaller region the gradient can push it all the way to reach the global optimum, can be a set of counterpart located in the bigger region with higher fitness, therefore having higher probability to be chosen, in such a way that the solution in the right way can be lost. Following this reasoning, we can now attempt to generate a similar fitness landscape considering a special synthetic dataset, linear classifiers and an objective function that measures the deceptive nature of the problem.

3.5.1 *Synthetic Classification Problem*

Let us consider a simple synthetic classification problem with a dataset $X = \{x_1, x_2, ..., x_n\}$ composed by 2-classes $\{\omega_1, \omega_2\}$, and 2-dimensions (d^1, d^2). Each class is contained into the sets C^1 and C^2, each set is composed by two subsets $C^1 = \{C^1_a, C^1_b\}$, $C^2 = \{C^2_a, C^2_b\}$ respectively, these are imbalanced subsets where the sub-index a stands for the majority subset, while sub-index b stands for the minority one. The imbalance factor is controlled by the parameter $p \in [0,1]$. Figure 3.3 shows a configuration for both classes with $p = 0.90$.

To generate the datasets, a Gaussian function has been chosen with parameters $(\mu, \mho)$. The centroid of each subset is given by μ with coordinates (d^1, d^2), while $\mho$ is the data covariance matrix, where $\mho = (\sigma_{d^1}, \sigma_{d^2})$. The standard deviation σ_1 is measured over dimension d^1, while σ_2 corresponds to d^2. To simplify our task, we build the data set such that the clusters of samples follow a regular shape within this 2-dimensional space, then we use the strategy to generate more data than needed with a factor q for each subset, and then trim the shape selecting the required samples that fall within this shape. For instance,

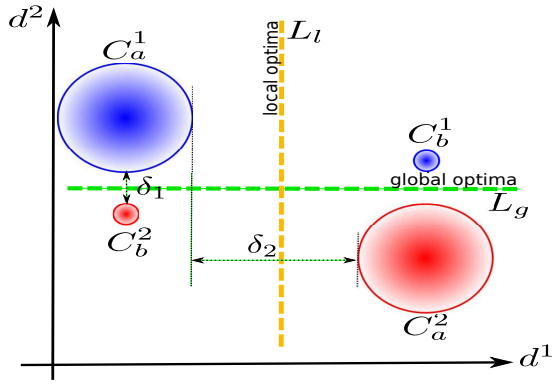


Figure 3.4: Synthetic classification problem with 2-classes $\{\omega_1, \omega_2\}$ and 2-dimensions (d^1, d^2), where each class set is composed by two imbalanced clusters $C^1 = \{C_a^1, C_b^1\}$, $C^2 = \{C_a^2, C_b^2\}$ with circular shapes. The figure shows two linear classifier models, the dotted green line shows the local optimum hyperplane L_l, and the dotted orange line shows the global optima hyperplane L_g. Furthermore, the optimal class separation is given by δ_1, while δ_2 is the separation between both majority clusters from each class.

for Figure 3.3 we generated p data points that fall within circle shaped clusters.

3.5.2 *Linear Classifier*

We consider linear classifiers to solve the previous synthetic classifier problem proposed considering the definitions given in (Theodoridis et al., 2010; Theodoridis and Koutroumbas, 2008). The linear classification problem involves a linear boundary, that is a hyperplane which can be described by $a^T x = b$ for some $a \in \mathbb{R}^n$ and $b \in \mathbb{R}$, where a and b correspond to d^1 and d^2 respectively for the classification problem presented in the previous section. Such a line is said to correctly classify these two sets if all data points with $y_i = +1$ fall on one side (hence $a^T x \geq b$) and all the others on the other side (hence $a^T x < b$).

Hence, the affine inequalities on (a,b): $y_i(a^T x - b) \geq 0, i = 1,...,m$, guarantee correct classification. The above constitute linear (in fact, affine) inequalities on the decision variable, $(a,b) \in \mathbb{R}^{n+1}$. This fact is the basis on which we can build a linear programming solution to a classification problem. Once a classifier (a,b) is found, we can classify a new point $x \in \mathbb{R}^n$ by assigning to it the label by the classification rule: $\hat{y} := \text{sign}(a^T x - b)$.

A classifier is a function c that maps an example $x \in X$ to a binary class $c(x) \in \{-1,+1\}$. A linear classifier uses; feature functions $\mathbf{f}(x) = (f_1(x),...,f_m(x))$ and feature weights $\boldsymbol{w} = (w_1,...,w_m)$ to assign $x \in \mathcal{X}$ to class $c(x) = \text{sign}(\boldsymbol{w} \cdot \mathbf{f}(x))$, where $sign(y) = +1$ if $y > 0$ and -1 if $y < 0$. The main idea is to apply a non-linear transformation to original features $\mathbf{h}(x) = (g_1(\mathbf{f}(x)),...,g_n(\mathbf{f}(x)))$ and learn a linear classifier based on $\mathbf{h}(x_i)$ Linear classifier decision rule: given feature functions $\mathbf{f}$ and weights $\boldsymbol{w}$, assign $x \in X$ to class $c(x) = \text{sign}(\boldsymbol{w} \cdot \mathbf{f}(x))$

The focus is on the direct design of a discriminant function/decision surface that separates the classes in some optimal sense according to an adopted criterion. The technique that is built around the optimal Bayesian classifier relies on the estimation of the probability density function (pdf) describing the data distribution in each class. However, in general this turns out to be a difficult task, especially in high-dimensional spaces.

Alternatively, one may focus on designing a decision surface that separates the classes directly from the training data set, without having to deduce it from the pdfs. We begin with the simple case of designing a linear classifier, described by the equation

$$g(x) = \boldsymbol{w}^T \mathbf{x} + w_0 = 0 \tag{3.6}$$

which can also be written as

$$\boldsymbol{w}'^T \mathbf{x}' \equiv [\boldsymbol{w}^T, w_0] \begin{bmatrix} x \\ 1 \end{bmatrix} = 0 \tag{3.7}$$

where $\boldsymbol{w} = [w_1, w_2, ...w_l]^T$ is known a the *weight vector* and w_0 as the *bias*. That is, instead of working with hyperplanes in the $\mathbb{R}^l$ space, we

work with hyperplanes in the $\mathbb{R}^{l+1}$ space, which pass through the origin. This is only for notational simplification. Once a $\boldsymbol{w}'$ is estimated, an x is classified to class ω_1 if

$$\boldsymbol{w}'^T\mathbf{x}' = \boldsymbol{w}^T\mathbf{x} + w_0 \geq 0 \tag{3.8}$$

or to class ω_2 if

$$\boldsymbol{w}'^T\mathbf{x}' = \boldsymbol{w}^T\mathbf{x} + w_0 < 0 \tag{3.9}$$

for the 2-class classification task. In other words, this classifier generates a hyperplane decision surface; points lying on one side of it are classified to ω_1 and points lying on the other side are classified to ω_2. For notational simplicity, we drop out the prime and adhere to the notation $\boldsymbol{w}$, x; the vectors are assumed to be augmented with w_0 and 1, respectively, and they reside in the $\mathbb{R}^{l+1}$ space.

3.5.3 *Objective Function*

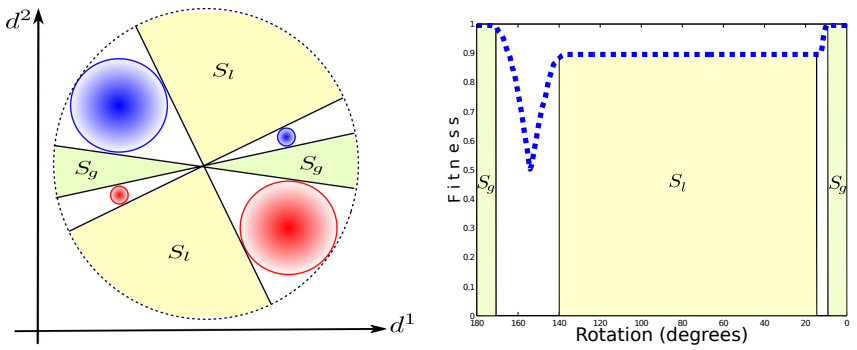
The most important factor in a deceptive problem is the objective function. A typical measure for classification performance, which is frequently used as objective function is the classification accuracy $Acc \in [0,1]$, which computes the proportion of the correctly classified data sample with respect to the dataset size. Accuracy gives information about how close the classifier performance is to the perfect classification, given by

$$Acc = \frac{TP + TN}{TP + FP + TN + FN} \tag{3.10}$$

where TP is the true positive, given by the number of samples correctly classified; FP is the false positive, given by the number of samples incorrectly classified; TN is the true negative, given by the number of samples correctly rejected; and finally FN is the false negative, given by the number of samples incorrectly rejected.

3.5.4 *Non-Deceptive Fitness Landscape*

We are here considering only linear models, and we must analyse how these models shape the fitness landscape. So, first we need to note that linear models can be located at any place around the datasets according to the line equation $d^2 = md^1 + b$, with m the slope, and b the vertical axis (d^2) intersection. These parameters allow 3 degrees of freedom building a $3D$ fitness landscape, but for the sake of clarity and to ease our explanation, we will analyse a $2D$ fitness landscape represented by a blue dotted line shown in Figure 3.4 (b), built by considering b fixed, and centred at the origin, as shown in Figure 3.4 (a).



(a) Search space

(b) Fitness landscape

Figure 3.5: The left figure shows the search space showing the sectors S_g and S_l, where the global optimum and local optima are located respectively. The right figure shows the $2D$ fitness landscape.

We begin our graphical analysis to build a fitness landscape in the search space, by drawing a line starting from the rotation center to the higher point of the minority sub-dataset C_b^2, and extending this line to meet the circle border as shown in the Figure 3.4 (a).

Secondly allowing to increase m to rotate this line clockwise till to the lower point of the majority sub-dataset C_a^1, we already had drawn the left hand sector S_g, approximately from 190° to 170°, where we can find linear models with 100% accuracy as shown in Figure 3.4 (b)

where it is shown with a blue dotted line the performance for the half section approximately from 180° to 170°.

Rotating the drawn line with the same direction approximately from 170° to 155°, will start classifying the data from the majority sub-dataset C_a^1 beginning to decrease the classification rate from 100% to 50% as shown in Figure 3.4 (b).

From approximately 155° to 140°, the linear model start recovering its classification performance reaching in this section a maximum of 90%, given by the imbalance factor $p = 0.90$ for this dataset configuration.

The next circle sector that our rotating linear model meets is the sector S_l, which contains the local optima region generates a plateau region in the fitness landscape from approximately 140° to 15° as shown in Figure 3.4 (b), which can be seen as a region with neutrality.

Finally, the linear model will increase the classification rate when it meets the minority sub-dataset C_b^1 until it reaches the perfect accuracy when it meets the next sector S_g. Since the geometry of the clusters is symmetric, then left half from the circle shaped search space will show similar behavior as described previously, mirroring the fitness landscape from 0°.

3.5.5 *Local Optima and Global Optimum*

The choice of getting two imbalanced partitions for each class is to create at least two regions S_l, S_g, where S_l contains the local optima, and S_g the global optimum, and considering just linear models as classifiers we can find models such as as L_l that correctly classifies a great part of the dataset, but there is a model L_g which is provides the global optimal (perfect accuracy) solution as shown in Figure 3.3. In order to create these regions, one possible strategy is to separate first the classes through a distance δ_1 to create the region S_g, then to separate the majority sub-datasets by a distance δ_2 to create the region S_l.

Figure 3.3 shows a possible configuration for the proposed classification problems. The symmetric construction of the dataset in Fig-

ure 3.3 is to easily transmit the general idea, but there could be other choices, for instance to consider more than two clusters per class with different amounts of imbalance.

3.5.6 *Deceptive Objective Function*

In order to get a deceptive objective function we can take a similar approach used to build a trap function shown in Figure **??**. Considering the same objective function showed in Equation 3.9 and applied to the previous synthetic classification problem, we can build a piecewise function to get two regions; one for the local optima and the other for the global optima. Therefore, we add a class separation criteria given by the minimum Δ_a and maximum Δ_b distance between classes, where Δ_a is measured between the points P_b^1 and P_b^2, and Δ_b is measured between the points P_a^1 and P_a^2 as shown in Figure 3.5.

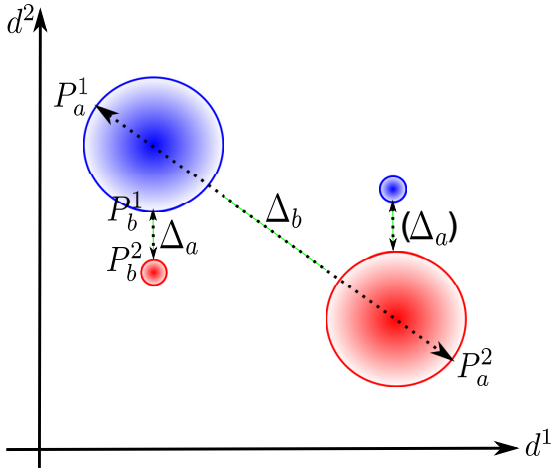


Figure 3.6: Class separation criteria, considering minimum and maximum separation given by the Euclidean distance Δ_a and Δ_b respectively, where $\Delta_a = E(P_b^1, P_b^2)$ and $\Delta_b = E(P_b^a, P_a^2)$.

Furthermore, we consider a minimum distance d_{min} measured from the current linear classifier and the closest point to any of the sub-

datasets. Now, we can implement the piecewise function through a threshold h related with classification accuracy rate, for instance $h = 0.99$, meaning that the rule will be applied when the linear model success classifying 99% of the dataset as follows

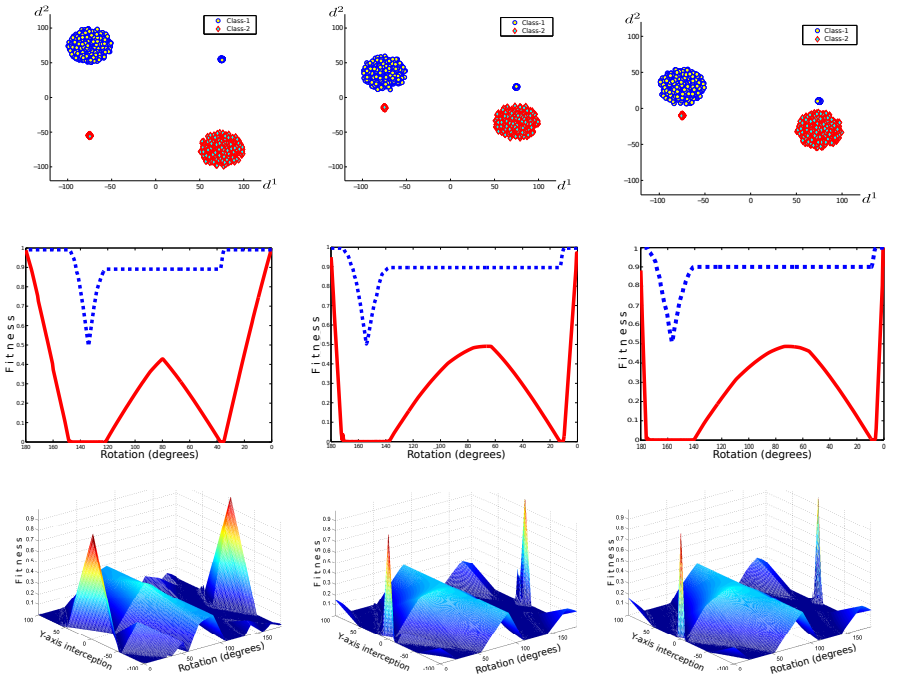
$$Fitness = \begin{cases} Acc \cdot \frac{d_{min}}{\Delta_a}, & \Delta_a > 0, \quad Acc \geq h, \\ Acc \cdot \frac{d_{min}}{\Delta_b}, & \Delta_b > 2\sigma_{d^1} + 2\sigma_{d^2}, \quad \text{otherwise} \end{cases} \tag{3.11}$$

Recall that we are designing a deceptive classification problem, which must show two regions, one for the local and other for the global optima. So, notice that Δ_a must be greater than zero in order to assure a class separation, and since in our design we are considering circle shaped clusters and their radius are $\sigma_{d^1}, \sigma_{d^2}$ for class 1 and 2 respectively, then in order to both majority subsets be separated Δ_b must be greater than $2\sigma_{d^1} + 2\sigma_{d^2}$.

3.5.7 *Deceptive Fitness Landscape*

Using the piecewise function from 3.5.6 applied in the $2D$ search space for linear models shown in 3.4 (a), and following similar explanation done to build the fitness landscape shown in 3.4 (b), we can now generate a deceptive fitness landscape when solving a particular configuration of the synthetic classification problem.

In Figure 3.6, we present three versions of the synthetic classification problem shown on the first raw, with its respective $2D$ fitness landscape on the second row, and the $3D$ fitness landscape on the third raw. The configurations are sorted according to the proportion $\delta_1 : \delta_2$ used to built them, which relates the global optimum and local optima respectively. Since the notion of deception is related with the relation between the regions where both the global optimum and local optima are located, then can be said that this proportion controls the grade of deception for each dataset configuration. A proportion $1 : 1$ means that both regions are equal, while $1 : 10$ means that the region for the local optima is 10 times bigger than the region for the global optimum.



(g) 1 : 1, Low (h) 1 : 5, Medium (i) 1 : 10, High

Figure 3.7: Set of synthetic deceptive classification problems sorted from left to right in increasing order of deception induced in the fitness landscape. First row shows the dataset configuration, vertical and horizontal axes are d^1, d^2 used to locate the subsets, in our experiments were set to be in $[-100, 100]$. Second row shows the 2D fitness landscape. Third row shows the 3D fitness landscape.

The second row of Figure 3.6 presents the corresponding $2D$ fitness landscape, and the third row shows the $3D$ fitness landscape for each dataset configuration version.

On the $2D$ fitness landscape a graphical performance comparison is presented between the non-deceptive fitness function represented by a blue dotted line, against the deceptive fitness function represented by a red solid line. The maximum classification performance value that

can be reached by the deceptive fitness function in the region of local optima is of approximately 50%. The main difference between both fitness functions, is that on the deceptive fitness function there are regions with values of zero performance, as can be noted on the red solid line. This particular behavior separate both regions, isolating one from the other, and clearly it can be observed how deception operates in a similar way to the trap function of Figure **??**.

The $3D$ fitness landscapes on the third row of the Figure 3.6 consider in addition the parameter of the vertical axis (d^2) intersection, meaning that now the linear models can be located around the datasets too. On the last plot with the configuration proportion of $1:10$ it can be seen a high degree of deception involved in this particular configuration.

3.5.8 *Preliminary Results*

The next step after generating the dataset configuration and the deceptive fitness function is testing a set of synthetic classification problems using some standard classification methods. In this case, the methods selected are; naive Bayesian method and the Support Vector Machine (SVM). The first method based on the data distribution leads to the local optimal with 50% performance in all the datasets tested, while the SVM method achieve a perfect score of 100% finding the global optimal, because this method concerns mainly in the optimal data separation.

3.6 Chapter Conclusions

In this chapter the notion of deception was presented as well as a quick tour guide from its origins to our days. We begin our journey from the very origins of the GAs and how deception was already identified as an important issue, we went through describing some well known deceptive functions, and finally arrived to the navigation tasks which are good environments where deception can be easily observed.

Table 3.1: Preliminary results, using a Naive Bayesian method and SVM.

Proportion	Naive Bayes	SVM
1:1	0.5000	1.0000
1:2	0.5000	1.0000
1:3	0.5000	1.0000
1:4	0.5000	1.0000
1:5	0.5000	1.0000
1:10	0.5000	1.0000

This short survey allowed us to observe that machine learning research lacks deceptive classification benchmarks, taking this research opportunity in this chapter was presented a design process to build a synthetic classification problem that together with a deceptive fitness function can be used to test classification methods. In this chapter, we only cover the GP open issue about benchmarks in a very reduced form, but proposing a very novel one.

To the best of our knowledge this is the first attempt to design a deceptive classification problem (Naredo et al., 2015). According with the preliminary results, as expected, linear classifiers will get trapped in the region of local optima. Of course, a non-linear classifier can easily solve this data set. Particularly, a method based on data separation (SVM) has success in finding the global optimum then the conclusion in this case is that the classification problem designed here is not deceptive for this kind of methods. Furthermore, this chapter helps to introduce the importance of the deception concept, which is frequently present in real-world problems and mostly related with the difficulty of the problem.

There are several future research lines to extend this work, one is to take into account the methods based on data separation to design synthetic classification problems that can introduce deception.

4

Novelty Search

ABSTRACT — Novelty search is a unique approach to search inspired by natural evolution's open-ended propensity to perpetually discover novelty. Rather than converging on a single optimal solution, nature discovers a vast variety of different ways to meet the challenges of life. As an abstraction of novelty-discovering in nature, novelty search directly rewards behaving *differently* instead of rewarding *progress* to some ultimate goal, which is the traditional approach to search. That is, in a search for novelty there is no pressure to be *better*.

To be more precise, evolutionary search is usually driven by measuring how close the current candidate solution is to the objective. That measure then determines whether the candidate is rewarded (i.e. whether it will have offspring) or discarded. In contrast, novelty search never measures progress at all. Rather, it simply rewards those individuals that are different.

Instead of aiming for the objective, novelty search looks for novelty; surprisingly, sometimes *not looking for the goal* in this way leads to finding the goal more quickly and consistently. While it may sound strange, in some problems ignoring the goal outperforms looking for it. The reason for this phenomenon is that sometimes the intermediate steps to the goal do not resemble the goal itself. John Stuart Mill termed this source of confusion the "like-causes-like" fallacy. In such situations, rewarding resemblance to the goal does not respect the intermediate steps that lead to the goal, often causing search to fail.

4.1 Introduction

Novelty search (NS) was born from a radical idea about artificial intelligence (AI) proposed by Lehman and Stanley (2008) based on previous evolutionary art experiments using their Picbreeder system (Stanley, 2007). Genetic art was first introduced by Richard Dawkins (Dawkins, 1986), currently known as evolutionary art (Romero and Machado, 2007). Picbreeder[1] is a webpage where visitors using their creativity can breed pictures, which are able to have "children" that are slightly different from their parents, and finally get new pictures with awesome designs.

The radical idea borrowed from Picbreeder to develop NS, is that it can be found solutions without really looking for them. The evolutionary art experiment is just one example of non-objective system of discovery (Stanley and Lehman, 2015). Furthermore, in the experiment was noticed that the webpage visitors frequently pick the available pictures up as parents, according with an interestingness criteria to have children from them. Pictures that were the most different, or more novel, among all other pictures had better chances of being selected for survival and reproduction. In other words, novelty is a rough shortcut for identifying interestingness (Stanley and Lehman, 2015).

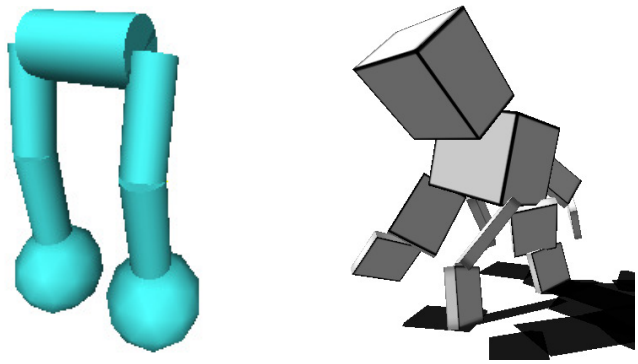
Provided with this insight the following step was to use these ideas into the design of a search algorithm. But a first stone on the road to design this algorithm and then endorse it, is the counter intuitive idea that a computer algorithm without an objective can work properly, since almost each of the algorithm designed do have an objective.

When solving a problem with relative low complexity to find the desired solution, the objective becomes a good idea to drive the search. But when the problem at hand shows an increasing degree in its complexity, then it is not so easy to find the desired (or approximate) solution. Particularly, in this context, following an objective could guide the search to find solutions that seem to be good, but in fact are far away from the desired solution. This is the deceptive phenomenon we

[1] More information about Picbreeder can be obtained in; http://www.picbreeder.org/

discussed in the previous chapter, and NS has shown an ability to find the correct stepping stones to construct the desired solution.

NS has achieved promising results in different areas of evolutionary robotics Woolley and Stanley (2012), such as navigation Lehman and Stanley (2008, 2011a); Urbano et al. (2014a); Urbano and Loukas (2013), morphology design Lehman and Stanley (2011b), and gait control Lehman and Stanley (2011a).



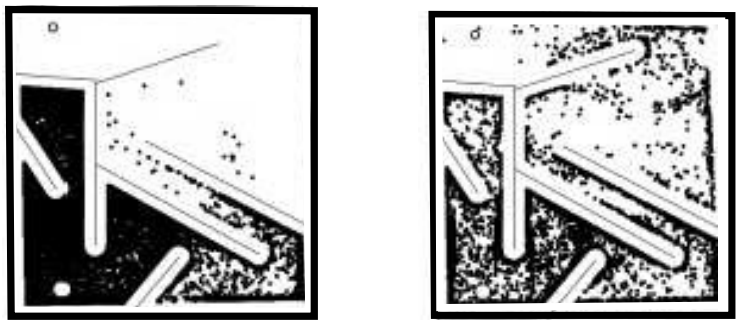
(a) Biped (b) Creature

Figure 4.1: Two examples of evolutionary robotics applying NS.

4.2 Open-ended Search

The bio-inspired origins of EAs suggest a substantial difference with respect to traditional optimization approaches. However, EAs are guided by an objective function and specially designed search operators, just like most optimization algorithms Luke (2013). The use of an objective function in EAs is a key difference with respect to natural evolution, an open-ended process that lacks a predefined purpose.

Open-ended artificial evolution does not use an objective function to drive the search, at least not an explicit one. An important feature of open-ended systems is the continuous emergence of novelty Banzhaf (2014). In fact, some of early EAs were open-ended Dawkins (1996);



(a) objective-based (b) novelty-based

Figure 4.2: Objective-based against novelty-based in a task navigation.
novelty-based

however, they have mostly been used in Artificial Life Ofria and Wilke (2004) and interactive search Kowaliw et al. (2012). Only recently has open-ended search been proposed to solve mainstream problems, one promising algorithm is Novelty Search (NS) proposed by Lehman and Stanley (2008).

4.3 Nuts & Bolts of NS

In NS individuals in an evolving population are selected based exclusively on how different their behaviour is when compared to the other behaviours discovered so far. Through the exploration of the behaviour space, the objective will eventually be achieved, even though it is not being actively pursued. Implementing novelty search requires little change to any evolutionary algorithm aside from replacing the fitness function with a domain dependent novelty metric. This metric quantifies how different an individual is from the other individuals with respect to their behaviour. Like the fitness function, the novelty metric must be adequate to the domain, expressing what behaviour characteristics should be measured and therefore conditioning what behaviours will be explored. The use of a novelty measure creates a

constant pressure to evolve individuals with novel behaviour features, instead of maximising a fitness objective.

A common criticism of NS is that it might appear to be a random or exhaustive search, however NS performs a structured search by accumulating information of the problem (Velez and Clune, 2014). Basically, NS will exhibit strong performance when randomly generated solutions, those from initial generations, exhibit weak performance, thus the search towards novelty will be correlated with the search for better performance. However, unlike objective-based search, NS does not stagnate around local optima, since once they are discovered NS will quickly push the search towards new regions in solutions space (Velez and Clune, 2014).

4.3.1 *Behavioral representation*

The behaviour of each individual is typically characterised by a vector of numbers. The experimenter should design the behaviour characterisation so that each vector contains aspects of the behaviour of the individual that are considered relevant to the problem that is being solved. Once the behaviour characterisation is defined, the novelty distance metric $dist$ can be defined as the distance between the behaviour vectors. A commonly used distance is the Euclidean distance between the vectors.

The behaviour characterisation can be for example the situation of the agent at the end of the trial or some measure that is sampled along the trial. For instance, when originally introduced, novelty search was demonstrated on a maze navigation task (Lehman and Stanley, 2011a), where the behaviour characterisation was a vector containing the final position (x, y) of the robot in the maze. Choosing the aspects of the behaviour that should be put in the behaviour characterisation typically requires domain knowledge, and has direct implications on the diversity of behaviours that will be found by evolution. Excessively detailed behaviour characterisations can open the search space too much, and might cause the evolution to focus on evolving behaviours that are

irrelevant for solving the problem. On the other hand, a too simple behaviour characterisation might be insufficient for accurately estimating the novelty of each individual, and can prevent the evolution of some types of solutions.

It is important to note that the detail of the behaviour characterisation is not necessarily correlated with the length of the behaviour vector. In the maze navigation experiments (Lehman and Stanley, 2011a), the authors expanded the behaviour characterisation to include intermediate points along the path of an individual through the maze, instead of just the final position. The authors experimented with different sampling frequencies, resulting in behaviour characterisations of different lengths, and the results showed that the performance of the evolution was largely unaffected by the length of the behaviour characterisation. Although a longer characterisation increased the dimensionality of the behaviour space, only a small portion of this space was reachable since adjacent points in a given path were highly correlated (i.e. the agent can only move so far in the interval between samples). It was demonstrated that larger behaviour descriptions do not necessarily imply a less effective search, despite having a larger behaviour space.

4.3.2 *NS algorithm*

Instead of designing an objective function that summarizes the performance of each individual, to use NS successfully the concept of uniqueness must be grounded in some way. The uniqueness of a solution must be measured against the rest of the evolved solutions; for instance, solutions can be compared based on their genotype or phenotype. Instead, NS is based on the concept of behavior space, where a behavior is characterized by a vector β that describes the way an agent K (or GP individual) acts in response to a series of stimuli (input) within a particular context. In NS, the proposed measure of sparseness ρ around each individual K described by its behavior descriptor β, is given by

$$\rho(\beta_j) = \frac{1}{k}\sum_{l=1}^{k} \mathrm{d}(\beta_j, \beta_l)\,, \tag{4.12}$$

where β_l is the l–th nearest neighbor of β_j in behavior space with respect to a distance or similarity measure $\mathrm{d}(,)$ (Lehman and Stanley, 2008), and the number of neighbors k is an algorithm parameter. Given this definition, when the average distance is large then the individual is located within a sparse region of behavior space, and it is located in a dense region if the measure is small, see Figure 4.3. The original NS proposal considers the current population and an archive of individuals to compute sparseness, this avoids backtracking. An individual is added to the archive if its sparseness satisfies a certain threshold condition ρ_{th}, which is the second NS parameter. Several papers have suggested implementing the archive as a FIFO queue of size q, this alleviates the cost of computing sparseness but adds another parameter.

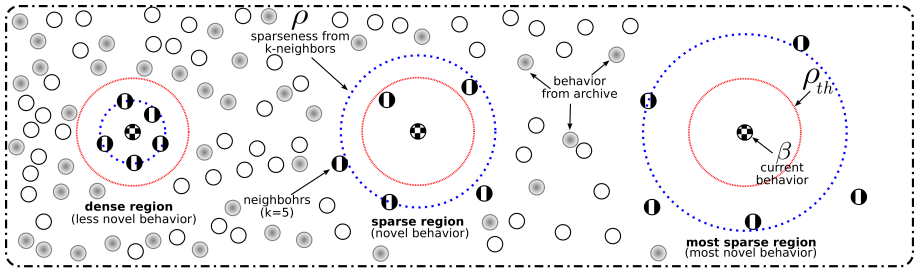


Figure 4.3: The original NS proposed in Lehman and Stanley (2008) uses a measure of local sparseness ρ around each individual behavior β within behavior space to estimate its novelty, considering the current population and novel solutions from previous generations stored in an archive by mean of a threshold which depends on the sparseness measure ρ_{th} of the current individual behavior β. The figure shows three different scenarios for an individual's behavior β, where the cases are sorted from the most dense region (less novel) to the most sparse one (the most novel).

4.3.3 *Underlying evolutionary algorithm*

Once the objective-based fitness is replaced with novelty, the underlying evolutionary algorithm operates as normal, selecting the most novel individuals to reproduce. Over generations, novelty search encourages the population to spread out across the space of possible behaviours, eventually encountering individuals that solve the given problem, even though progress towards the solution is not directly rewarded.

4.3.4 *Minimal Criteria Novelty Search*

Lehman and Stanley proposed an extension to NS that evolves solutions more efficiently (Lehman and Stanley, 2010c), called minimal criteria novelty search (MCNS). The main idea behind MCNS is that novelty should be preserved as long as it satisfies some minimal criteria (MC) for selection. Those individuals that meet the MC preserve their novelty score, and individuals that do not satisfy the MC will receive a penalized score, given by

$$\rho_{MCNS}(\beta_j) = \begin{cases} \rho(\beta_j), & \text{if the MC are satisfied} \\ 0, & \text{otherwise.} \end{cases} \tag{4.13}$$

For instance, for navigation in an unenclosed maze a simple MC is that each individual must stay within the maze. The MC can be used as a tool to manage and discard infeasible solutions during the search, thereby directing the search towards the most promising regions in the search space (Lehman and Stanley, 2010c). Moreover, several works have attempted to combine novelty with an objective function. For instance, (Cuccu et al., 2011b) and (Doucette and Heywood, 2010) proposed an arithmetic combination of both measures, and a novelty-based multiobjectivisation is proposed in (Mouret, 2011).

4.4 Contributions on NS

NS suffers from several shortcomings and are the main topic of the proposal developed in this section. In particular, computing novelty using Equation 4.11 can lead to several problems. First, it is not evident which value for k will provide the best performance. Second, the sparseness computation based on Equation 4.11, has a complexity of $O(m+q)^2$, where m is the size of population, and q is the archive size, which will grow unbounded if it is not implemented as a FIFO queue (Lehman and Stanley, 2008, 2010b,c, 2011a). Third, choosing which individuals should be stored in the archive is also important, several approaches have been proposed and each adds an additional empirical parameter or decision rule.

In this section we present two proposals to compute novelty. The first proposal is an extension of the progressive minimal criteria NS (Gomes et al., 2012), named as MCNS$_{bsf}$, which considers a dynamical threshold based on the best-so-far (bsf) solution. The second proposal is a probabilistic approach to compute novelty named as probabilistic NS (PNS), which eliminates all of the underlying NS parameters, and at the same time reduces the computational overhead of the original NS algorithm.

4.4.1 *MCNS$_{bsf}$*

In this work we propose an extension of the progressive minimal criteria NS, we call it MCNS$_{bsf}$ (best-so-far) and it combines NS with the problem's objective using the above equation. Thus, MCNS$_{bsf}$ evolves the population through novelty, but constrains the search by penalizing individuals that do not meet the MC, which is based on performance. Considering a minimization problem we define that the MC of Equation 4.12 is satisfied if and only if $F(K_j) \leq F(K_{bsf})(\alpha+1)$, where $F(\cdot)$ is the objective function that assigns a quality score to each individual K_j, K_{bsf} is the best solution found so far and $\alpha \in [0,1]$. We choose a dynamic threshold $(\alpha+1)$ as a proportion of $F(K_{bsf})$, because if we

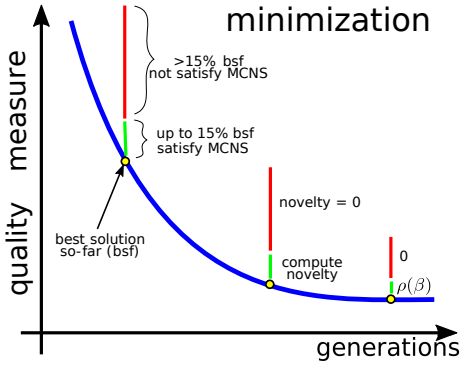


Figure 4.4: Illustration of the effect of the proposed MCNS approach.

use a static threshold we would need to set this threshold differently for each problem. If the value is too high, then none of the individuals will satisfy the MC, which was observed in preliminary tests as shown in chapter 9. Conversely, if it is too low, all individuals will satisfy the MC and it would become useless. The proposed approach can be seen as a broader group of methods, where the MC is set proportionally to some statistic over the current population, in this case it is the best fitness, but the method could set the MC based on the median, the mean or any other statistic. However, in this work we only consider the best solution since it provides the greediest approach to possibly improve convergence speed, the original goal of the MCNS approach, while other variants will be studied in future work.

Some drawbacks of the proposed MCNS variant are the following. MCNS introduces a new parameter α, in addition to those already used by NS. Moreover, if many individuals in a given generation are assigned a novelty measure of 0, the worst case, then the search might lack a sufficient gradient and could proceed randomly. However this last point is applicable to most MCNS algorithms.

4.4.2 *Probabilistic NS*

The second proposal to overcome the drawbacks of NS is to use a probabilistic approach towards computing the novelty of each solution. The behavior descriptor β of a GP classifier is modelled as a binary random vector of size n, with n the number of fitness cases and an estimation of its probability mass function $P(\beta)$ is used to compute the novelty ϕ of each solution behavior β , given by

$$\phi(\beta) = \frac{1}{P(\beta)}\,, \tag{4.14}$$

such that the novelty of each solution is inversely proportional to the probability of producing it during the search. In this way, measuring novelty is accomplished without the need of empirically tuning any additional parameters in the GP search. The time complexity of computing ϕ is negligible once $P(\beta)$ is known and does not require the use of an external archive. To simplify this process further, we make the naive Bayesian assumption that the individual dimensions i in the behavior descriptor $\beta_{j,i}$ are independent; i.e. that the performance of a GP individual K_j on a particular training sample is independent with its performance on any other. Under this assumption, ϕ can be computed as

$$\phi(\beta_j) = \frac{1}{\prod_{i=1}^{n} P_i(\beta_{j,i})}\,, \tag{4.15}$$

where $P(\beta_i)$ represents the pmf of the i-th component β_i of β. Therefore, the problem then becomes estimating each $P(\beta_i)$ during the search, which is accomplished as follows. First, let B^t represent the behavior matrix of generation t, given by the Equation **??**, such that B^t contains all behaviors $\beta_{j,i}^t$ from each individual j corresponding to the

i-th fitness case at a generation t, with m individuals in the population and n fitness cases.

$$B^t = \begin{bmatrix} \boldsymbol{\beta}_1^t \\ \vdots \\ \boldsymbol{\beta}_j^t \\ \vdots \\ \boldsymbol{\beta}_m^t \end{bmatrix} = \begin{bmatrix} \beta_{1,1}^t & \cdots & & \cdots & \beta_{1,n}^t \\ \vdots & \ddots & & & \vdots \\ & & \beta_{j,i}^t & & \\ \vdots & & & \ddots & \vdots \\ \beta_{m,1}^t & \cdots & & \cdots & \beta_{m,n}^t \end{bmatrix}, \tag{4.16}$$

$$\delta^t = \sum_{j=1}^{n} \beta_{j,i}^t = \begin{bmatrix} \delta_1^t & \cdots & & \cdots & \delta_n^t \end{bmatrix}. \tag{4.17}$$

A second step is to compute the frequency of different behaviors in the first generation ($t = 0$) for each fitness case as shown in Equation **??**. The frequency of 1s as behavior description related with a particular fitness case is computed by summing over each column of B^t, given by ${}_1\delta_i^{t=0} = \sum_{j=1}^{n} (\beta_{j,i}^{t=0} = 1)$, and the frequency of 0s is expressed as the complement ${}_0\delta_i^{t=0} = m - ({}_1\delta_i^{t=0})$. The accumulated frequencies of 1's and 0's, are computed iteratively every generation t by; ${}_1\widehat{\delta_i^t} = {}_1\widehat{\delta_i^{t-1}} + {}_1\delta_i^t$, and ${}_0\widehat{\delta_i^t} = (t+1)m - {}_1\widehat{\delta_i}$, respectively.

Knowing the frequency of the different behaviors, the probability $P_i(\beta_{j,i})^t$ can be estimated by

$$P_i(\beta_{j,i}^t = 1) = \frac{{}_1\widehat{\delta_i}}{m(t+1)}, \tag{4.18}$$

for the 1s, and for the 0s it is given by

$$P_i(\beta_{j,i}^t = 0) = 1 - P_i(\beta_{j,i}^t = 1). \tag{4.19}$$

Then, the probabilistic novelty of a new behavior β_j^t that appears at generation t can be computed by

$$\phi_j^t = \frac{1}{\prod_{i=1}^{n} P_i(\beta_{j,i}^t)}, \tag{4.20}$$

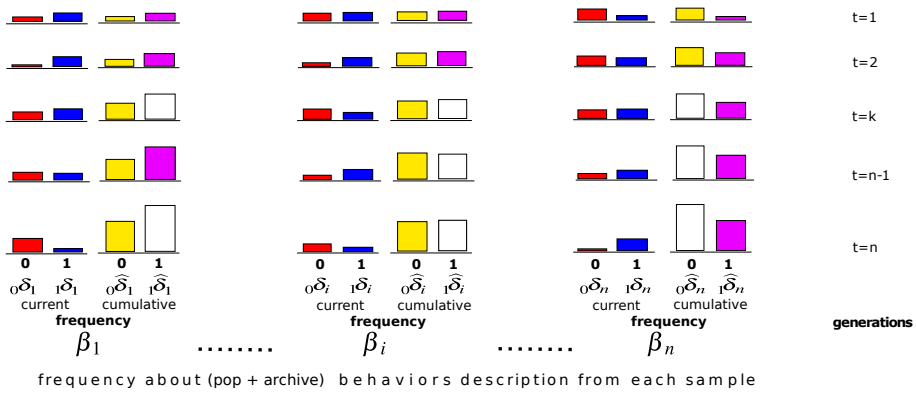


Figure 4.5: Representation of the PNS novelty measure, where each column represents one feature βi of the AD vector and each row is a different generation. In each column, two graphics are presented, on the left is the frequency of individuals with either a 1 or 0 for that particular feature in the current population, and on the right the cumulative frequency over all generations.

taking logarithms on both sides to avoid numerical error, we obtain

$$\log \phi_j^t = \sum_{i=1}^{n} \log\left(\frac{1}{P_i(\beta_{j,i}^t)}\right). \tag{4.21}$$

If we consider a selection method based on ranking (such as tournament), which is the usual method for GP, then the novelty measure used by PNS can be computed by

$$\phi_{PNS}{}_j^t = \sum_{i=1}^{n} \frac{1}{P_i(\beta_{j,i}^t) + \epsilon}, \tag{4.22}$$

where ϵ is a small real value to avoid numerical errors caused by divisions by zero, one way to set it is to $1/population-size$. The general idea of the PNS measure for a behavior descriptor β with a binary string representation, is depicted in Figure 4.5.

PNS seen to be related with the frequency fitness assignment (FFA) method proposed in (Weise et al., 2014), where the fitness is assigned through a lookup table. This table contains the absolute frequencies of the discretized values from the real-valued objective function for each individual. The main difference between FFA and PNS is that PNS model the population behaviors (which can be the objective-based fitness) through a distribution function (i.e. a Gaussian function), and the fitness is assigned according to the corresponding probabilistic value from the currently updated distribution.

Another approach that can be related with PNS is the estimation of distribution algorithms (EDAs) (Larrañaga and Lozano, 2001), sometimes called probabilistic model-building genetic algorithms (PMBGAs) (Pelikan et al., 2002). EDAs diverge from the traditional EAs, particularly in the approach to generate the new population. The population in EAs is generated using genetic operators, such as crossover and mutation, while EDAs generate a new population by sampling a probability distribution, estimated by an explicit probability function encoded by a Bayesian network, a multivariate normal distribution, or another model from selected individuals in previous generations (Larrañaga and Lozano, 2001).

Because the similarities that PNS show respect to the methods FFA and EDAs, could be some that argue that PNS is a different kind of search from NS, and that could be named, for instance as: rarity search (RS).

PNS is a method directly related with NS, but Even though, PNS uses a probability distribution to assign fitness, it can be used on top of most of the EAs, without affecting the selection method or any other mechanism in the search process, but only the approach to assign fitness.

4.5 Chapter Conclusions

This chapter presented the NS algorithm as an option of the traditional approach to guide the search through following objectives. It

was introduced the philosophical ideas behind NS and its relationship with natural evolution. The Open-ended evolution issue in GP stated in (O'Neill et al., 2010) is addressed by the NS approach, which is one of a limited amount of heuristic strategies that all but guarantees that the search can proceed in an open-ended manner.

Furthermore, it was explained the NS algorithm as well the required steps to implement it over most evolutionary algorithms. One important consideration to implement NS is the behavioral representation, which in evolutionary robotics mostly is used the agent last position. Finally, we present two proposals to extend NS, by incorporating the objective function explicitly and dynamically (MCNS) and by proposing a more efficient and parameter free NS variant (PNS). Both of these approaches are fully evaluated on supervised classification in Chapter 9.

Last consideration is that NS as an open-ended search is not a niche strategy, it can be used to solve traditional learning tasks effectively, and based on some measures outperforms standard OS.

5

NS Case Study: Automatic Circuit Synthesis

ABSTRACT — Before applying NS to GP search, this chapter presents a case study of NS applied to a complex real-world task. In this case we use a simple binary GA, to illustrate the manner in which NS can be applied and its results relative to standard objective-based search. In particular, in this chapter a topology synthesis method is introduced using genetic algorithms based on novelty search (NS). The synthesized topologies are current follower (CF) circuits; these topologies are new and designed using integrated circuit CMOS technology of 0.35μm. Topologies are coded using a chromosome divided into four genes: small signal gen (SS), MOSFET synthesis gene (SMos), polarization gene (Bias) and current source synthesis gene (CM). The proposed synthesis method is coded in Matlab and uses SPICE to evaluate the CFs fitness. The GA based on NS (GA-NS) is compared with a standard objective-based GA, showing unique search dynamics and improved performance. Experimental results show twelve CFs synthesis generated by GA-NS, and their main attributes are summarized. This work confirms that NS can be used as a promising alternative in the field of automatic circuit synthesis.

5.1 Introduction

The automated design process of integrated circuits (IC) based on complementary metal-oxide-semiconductor (CMOS) is carried out by the industry of Electronic Design Automation (EDA), to synthesize amplifiers, voltage followers, current mirrors, among others. EDA is a

category of software tools for designing electronic systems, which typically are validated by simulation (Gielen and Rutenbar, 2000; Martens and Gielen, 2008; Mazumder and Rudnick, 1999; Rutenbar et al., 2007; Vellasco et al., 2001).

EDA tools increase productivity in the design of ICs, even for the circuit blocks that are not repetitive. Particularly, the analog design automation is more complex compared to digital design automation, because the relationships among their specifications are more complex. Moreover, analog design requires experience, intuition and creativity, primarily because it works with a large number of parameters that usually exhibit complex interactions among them.

In recent years several researchers have proposed some methods for analog circuit synthesis; for example, (Gielen and Rutenbar, 2000) focuses on the design of passive circuits. Evolutionary algorithms (EA) have proven to be a good choice to generate new circuits, or to optimize previous by sizing their topologies (Gielen and Rutenbar, 2000; Martens and Gielen, 2008; Rutenbar et al., 2007; Vellasco et al., 2001). In this work, we focus on finding new topologies that synthesize a current follower (CF) circuit using a binary encoded genetic algorithm (GA) (Holland, 1975). A binary genetic encoding of unity gain cells (UGC) is implemented through nullator and/or norator elements.

Therefore, in this work we use a GA (Holland, 1975) based on NS to synthesize new topologies for a current follower circuit. The hypothesis in this work is that NS will be able to find new circuit designs, designs that are left unexplored by a standard search process that relies on a fixed objective function. The experimental results show that NS allows the GA to find different topologies with respect to those found in previous works where a standard objective-based GA was used. This work and its results give us insights about the usefulness of NS on electronic circuit synthesis, the first such work in related literature.

5.2 GA Synthesis and CFs Representation

GAs in particular have been used to design electronic circuits in several works. For instance, a GA was used to generate the topology, and/or the component values of electronic circuits in (Gielen and Rutenbar, 2000).

Another alternative was proposed by Koza et al. in (Koza et al., 2005), that use genetic programming (GP) with a variable-length representation containing topology modifying operators, component-creating operators and arithmetic-performing operators. In Koza's approach the operators that modify the circuit topology and select component values are inseparable, and all are under the control of the evolutionary processes operating on the expression.

Circuit synthesis involves both the selection of a suitable topology and the choice of component values, and as Koza has shown, these may be optimized simultaneously by the evolutionary process. However, there is no reason why these operations should not be performed separately, with different optimization methods being used, specifically tailored for each problem. Clearly the circuit topology must be chosen first, and the appropriate algorithm for this task can be a GA. For each circuit topology generated the component values can then be optimized, and the performance of the circuit used as the fitness function in a GA. In such cases, circuit fitness can be evaluated using SPICE, that is a simulation program with IC emphasis. The code for which can bee incorporated into the synthesis program.

The component values could also be optimized using a GA, but this is not the best choice for problems involving well-behaved objective functions dependent on a fixed number of variables. It is well established that numerical optimization methods converge much faster and involve fewer objective function evaluations (Flores Becerra et al., 2014). No optimization method is guaranteed to find the global optimum, but it has been found that numerical optimization of component values achieves a high proportion of results close to the global optimum. This hybrid approach using a GA to select a suitable topology and numerical optimization to choose component values is likely to be

more efficient than allowing evolution to perform both tasks concurrently. In this work, however, we will only focus on the first part of the problem, that of finding novel circuit topologies, leaving any further optimization as future work for real-world implementations.

In particular, this work focuses on the synthesis of current followers (CFs). These circuits copy the value of a current to other parts of a circuit with higher impedance. In other words, the CF can maintain a current higher impedance loads without altering the original source. CFs can be used in various analog circuits, such as filters, oscillators, data transmission, and current conveyors (CC) (Duarte-Villaseñor et al., 2011; Flores Becerra et al., 2014; Tlelo-Cuatle and Duarte-Villaseñor, 2008; Tlelo-Cuautle et al., 2007, 2010).

5.2.1 *Objective Function*

The objective function for a synthesized CF circuit K is adapted from (Duarte-Villaseñor et al., 2011), and is given by

$$F(K) = |1-g| + 0.25 \cdot f(BW) + 0.5 \cdot (f(Z_{in}) + f(Z_{out})) \tag{5.23}$$

with $f(BW)$, $f(Z_{in})$, and $f(Z_{out})$ defined as

$$f(BW) = \begin{cases} \frac{|BW-10^7|}{10^7}, & BW \leq 10^7 \\ \frac{\alpha_1|BW-10^7|}{10^7}, & 10^7 < BW \leq 10^8 \\ 0.1 + \frac{\alpha_2|BW-10^7|}{10^7}, & 10^8 < BW \leq 10^9 \\ 0.2 + \frac{\alpha_3|BW-10^7|}{10^7}, & 10^9 < BW \end{cases} \tag{5.24}$$

$$f(Z_{in}) = \begin{cases} \alpha_2 |100 - Z_{in}|, & Z_{in} < 100 \\ \alpha_1 |100 - Z_{in}|, & Z_{in} \geq 100 \end{cases} \tag{5.25}$$

$$f(Z_{out}) = \begin{cases} \alpha_2 \left|10^4 - Z_{out}\right|, & Z_{out} < 10^4 \\ \alpha_3 \left|10^4 - Z_{out}\right|, & Z_{out} \geq 10^4 \end{cases} \tag{5.26}$$

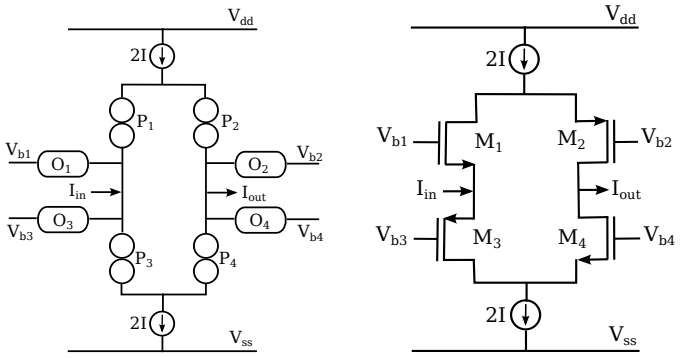
where g is the circuit gain, BW is the circuit bandwith, Z_{in} is the input impedance, Z_{out} the output impedance and the α_is are function

parameters set to $\alpha_1 = 0.01$, $\alpha_2 = 0.001$ and $\alpha_3 = 0.0001$. As can be seen, Equation 5.20 defines a minimization problem, where the goal is to increase the gain g of the circuit, subject to the penalizing terms of equations 5.21, 5.22 and 5.23. In particular, the parametrization given in the above equations pushes the search to find circuits with $g = 1$, $BW = 10MHz$, $Z_{in} = 100\Omega$ and $Z_{out} = 10K\Omega$, which are good values for CF circuits (Duarte-Villaseñor et al., 2011). As stated before, for a standard GA Equation 5.20 can be used to define fitness directly, but this is not necessarily the case as discussed in Chapter 4.

5.2.2 *CFs Representation for a GA*

Using nullators and norators, it is possible to describe the behavior of current followers, where for synthesis purposes, a nullator (O) and a norator (P) always must form a joined-pair (Tlelo-Cuatle and Duarte-Villaseñor, 2008; Tlelo-Cuautle et al., 2007). The node in the joined terminals of the O-P pair is associated to the source (S) of a MOSFET, the other terminal of the O element is associated to the gate (G), and the other terminal of the P element to the drain (D). Figure 5.1 (a) shows the nullor-based description of a CF, it consists of four P elements (P1-P4), each one joined with an O element. In this work we use the binary genetic encoding introduced by (Tlelo-Cuautle et al., 2007), for automatic synthesis of analog circuits, particularly to find CF topologies. The O-P pairs can be described by a small-signal gene called SS; the synthesis of each O-P pair can be codified by the gene called $SMos$. The addition of current and voltage biases is codified by the gene $Bias$; and the synthesis of the current biases for the current mirror CMOS by the gene CM.

The SS gene is defined as joined terminals of the O-P pairs, using two bits for each O-P pair. The $SMos$ gene is defined as PMOS or NMOS, using one bit for each O-P pair. The $Bias$ gene is three bits for each CMOS, and the CM gene is defined for 4 CMs: simple, cascode, Wilson, and modified Wilson; it uses two bits for each CMOS. As



(a) Nullors (b) MOSFETs

Figure 5.1: CF representation using (a) nullors and (b) MOSFETs.

a result, the genetic representation of the CF consists of a chromosome Ch_{CF} of four ordered genes, given by

$$Ch_{CF} = (SS, SMos, Bias, CM) \,. \tag{5.27}$$

An example of the CF synthesis for the chromosome proposal shown in Figure 5.1(a) is depicted in Figure 5.1(b) using MOSFETs, where the CF obtained is known. This description has been reported in (Duarte-Villaseñor et al., 2011), where CFs were synthesized using a standard GA. By using n as the number of O-P pairs, the length of Ch_{CF} is given by

$$\begin{aligned} Length(Ch_{CF}) &= 2n + n + 3n + 2 \\ &= 6n + 2 \,. \end{aligned}$$

In this work we consider three different topology sizes related with the number of MOSFETs used by the CFs, from 1 to 3. For the topology using just one MOSFET, the chromosome length is 8 bit, when using two MOSFETs 13 bits are used, and for three MOSFETs 18 bits.

5.2.3 *CF synthesis with GA-NS*

In this scenario, we hypothesize that the very nature of circuit design, requires an exploratory and unorthodox approach, that promotes uniqueness and creativity. It is precisely in these scenarios, Lehman and Stanley argue, that NS should be able to produce strong results relative to standard objective-based search (Stanley and Lehman, 2015). To apply NS, we choose to make $\beta = Ch_{CF}$, and thus sparseness is computed based on the Hamming distance.

As stated before, our hypothesis is that NS will be able to find designs that are left unexplored by a standard search process. Therefore, in the following experimental section we will focus our work on comparing the performance of the GA-NS with the standard GA-OS approach, highlighting the unique solutions found by GA-NS. It must be stressed that Equation 4.11 is used to determine fitness in NS, and thus establish the selection pressure for the GA search. Nonetheless, the final solutions returned by the search are still chosen based on the objective function, in this case $F(K)$ which is given Equation 5.20. It mus be understood that NS changes the manner in which the search progresses, but the underlying goal of the task is still be evaluated by a domain specific objective. The difference with a standard GA, however, is that the objective function is not used to directly guide the search, it is only used offline to choose the final solution returned by the algorithm.

5.3 Results and Analysis

Hereafter, we will refer to the standard GA as objective or objective-based search (GA-OS), and to GA based on NS as novelty or novelty-based search (GA-NS). The parameters used for both algorithms in all experiments are shown in Table 5.1, in accordance with previous works (Duarte-Villaseñor et al., 2011; Flores Becerra et al., 2014; Tlelo-Cuatle and Duarte-Villaseñor, 2008; Tlelo-Cuautle et al., 2007, 2010). The GA-NS parameters are summarized in Table 7.3. All algorithms are ex-

Table 5.1: Shared parameters for the GA and GA-NS.

Parameter	Description
population size	20
Max generations	200
Stop criteria	10 generations without the best-fitness changing
Selection	Tournament (size= 3)
Crossover	one point
Crossover rate	0.9
mutation	one point
Mutation rate	0.1

Table 5.2: GA-NS parameters.

Parameter	Description
k–neighbors	half of pop size
ρ_{th}	half of chromosome size
archive control	FIFO
archive size	double of pop size

ecuted 10 times and performance is analyzed based on the objective function value of the best solutions found.

The quality measured by the objective function in Equation 5.20 is used by GA-NS to choose the best solution at the end of the run, and by GA-OS as the fitness value to guide the search and to choose the best solution. Note that the objective function defines a minimization problem. As stated before, fitness in GA-NS is given by the sparseness measure in Equation 4.11, using a binary representation for β and the Hamming distance. All experiments were carried out on a Workstation with a Xeon 3.50GHz processor with 16GB of RAM.

Table 5.3: Synthesized CF topologies with one MOSFET.

method	generation	decimal	score
GA-OS	6	143	5.4956
GA-NS	6	122	5.4522
GA-NS	4	219	5.2143

Table 5.4: Synthesized CF topologies with two MOSFET.

method	generation	decimal	score
GA-NS	8	4567	5.0003
GA-NS	6	4979	5.2450
GA-NS	95	5059	5.0003
GA-NS	27	6147	4.8482
GA-NS	91	3638	4.7058

5.3.1 *CF Topologies*

Tables 5.3 to 5.5 summarize the most notable topologies found by GA-OS and GA-NS, respectively for the three considered topology sizes (1, 2 and 3 MOSFETs). In these tables, the second column shows the generation where the best CF topology was found, and the third column shows the decimal conversion of the binary chromosome, while the last column shows its performance as given by the objective function. Moreover, figures 5.2 - 5.4 present the corresponding circuit topologies; i.e., the phenotype of the solutions. For convenience, the legend of each topology in figures 5.2 - 5.4 provides the corresponding decimal conversion of the genotype (chromosome) of the solution.

For the simplest case, considering a single pair O-P, Table 5.3 and Figure 5.2 show two topologies found by GA-NS and one by GA-OS. It is important to note that the GA-OS consistently found the same topologies, one and again. Furthermore, the GA-NS found different results. All topologies found have been previously reported and studied in the literature (Razavi, 2001); this was expected due to the small sizes of

Table 5.5: Synthesized CF topologies with three MOSFET.

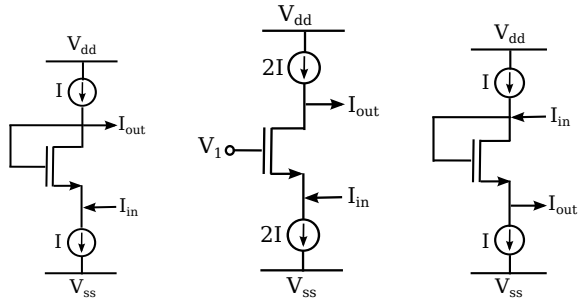
method	generation	decimal	score
GA-NS	90	152195	5.2188
GA-NS	11	62399	5.2468
GA-NS	49	116851	5.2465
GA-NS	193	252746	5.3453

the CFs, and because we are using the same representation and fitness function as previous works.

For the second series of experiments considering two pairs of O-P (two MOSFETs), Table 5.4 and Figure 5.3 only the best topologies found by GA-NS are presented. Topologies No. 4979 and 5059 can be considered trivial because you can find it manually quite easily. Moreover, the topology No. 36387 shows a known circuit (Razavi, 2001), which confirms that the GA-NS is guiding the search to good solutions in the search space. Finally, topologies No. 4567 and 6147 are novel, and should be considered as new designs of CF. In both topologies gain values are near to one, but Z_{in} and Z_{out} are not ideal; however these impedances can be occupied for filter designs made to measure.

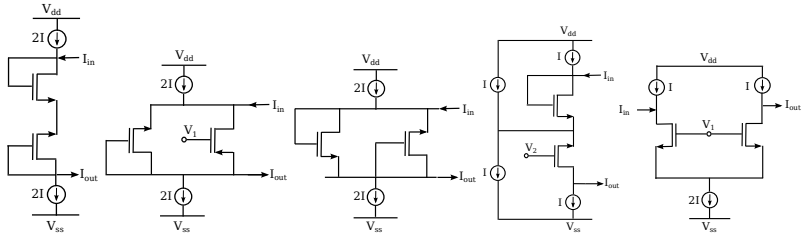
The third set of experiments used three O-P pairs to build circuits with 3 MOSFETs, these results are summarized in Table 5.5 and Figure 5.4. The topology No. 152195 (Figure 5.4(a)) shows three serial Cfsof ine CMOS, this a clear example of a synthesized CF built from smaller Cfs. Figures 5.4 (b) and (c) show more elaborate constructions of CFs found by GA-NS. Finally, the topology no. 252746 in Figure 5.4 (d) shows a novel structure that has not been presented in any related literature. Such a design confirms the ability of the NS paradigm to explore the search space and find unorthodox solutions, even to long-standing and well-known problems.

It is noteworthy, that the current sources as ideal presented in Figures 5.2, 5.3 and 5.4, the GAs are generated by mirrors current type Wilson modified. All topologies have a performance as a current fol-



(a) No. 143 (b) No. 122 (c) No. 219

Figure 5.2: Three CF topologies for the synthesis of a CF with the topology size of one MOSFET.



(a) No. 4567 (b) No. 4979 (c) No. 5059 (d) No. 6147 (e) No. 3638

Figure 5.3: Five CF topologies for the synthesis of a CF with the topology size of two MOSFETs found by GA-NS.

lower with dimensions $Wn = 6\mu$, $Wp = 4.4$, and $Ln = Lp = 1.2\mu$; with $Vdd = 1.6V$, $Vss = 1.6V$, and $I = 20uA$.

5.3.2 *Comparison between GA-OS and GA-NS*

Let us now analyze the effect that the NS algorithm has on the search process for circuit synthesis, relative to objective-based search. Figure 5.5 shows convergence plots of the best solution found by each algorithm, showing the average behavior over all runs. The figure plots

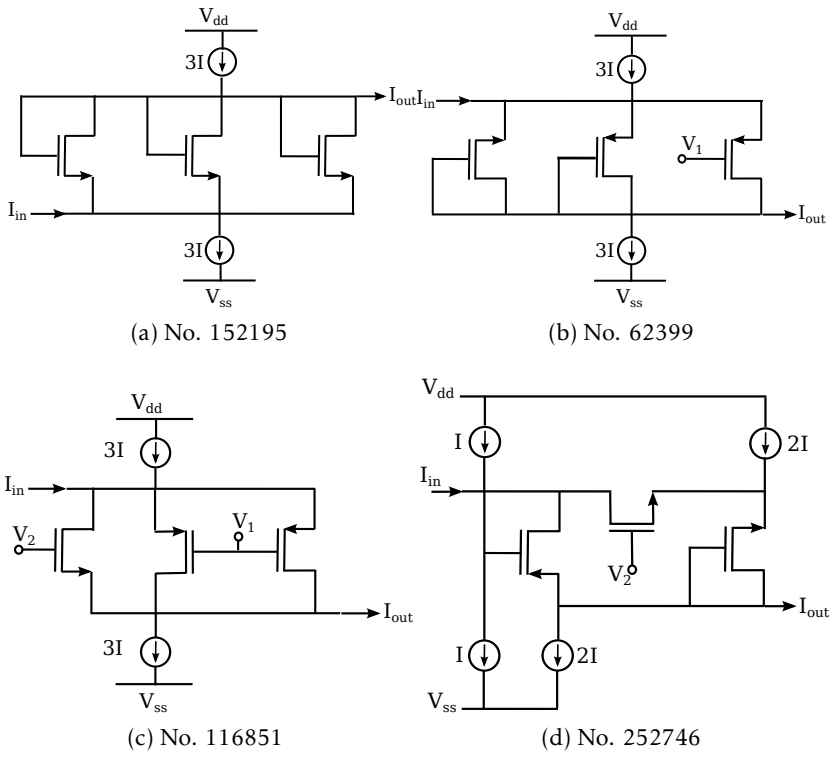


Figure 5.4: Four CF topologies for the synthesis of a CF with the topology size of three MOSFETs found by GA-NS.

the objective function value of the best solution with respect to the number of generations. In particular, we focus on the experiments using two O-P pairs, for circuits with two MOSFETs. Based on this plot, we can see no significant difference between both algorithms, the quality of the solutions found is comparable and the convergence is similar for both algorithms, even though GA-NS reaches a better average performance. However, as stated before, the topologies found by GA-NS do not match those found by GA-OS. Therefore, in terms of the performance of the synthesized circuits both algorithms are more or less equivalent, the difference lies on the actual topologies found by each

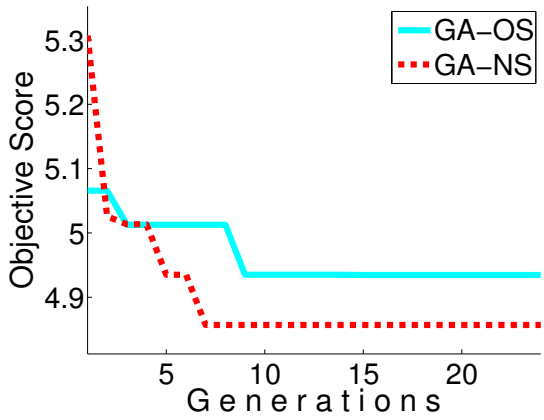


Figure 5.5: Convergence plots for GA-NS and GA-OS for the two MOSFET CFs, showing the performance (objective function value) of the best solution found over the initial generations of the search. The lines represent the average over 10 runs of the algorithms.

method. This results should be highlighted, since some have argued that the search carried out by the NS algorithm seems to be random, given its exclusion of the objective function. However, our results support findings in other domains (Urbano et al., 2014a; Velez and Clune, 2014), that show that NS is not a random process and that it can at the very least achieve the same level of performance as objective-based search.

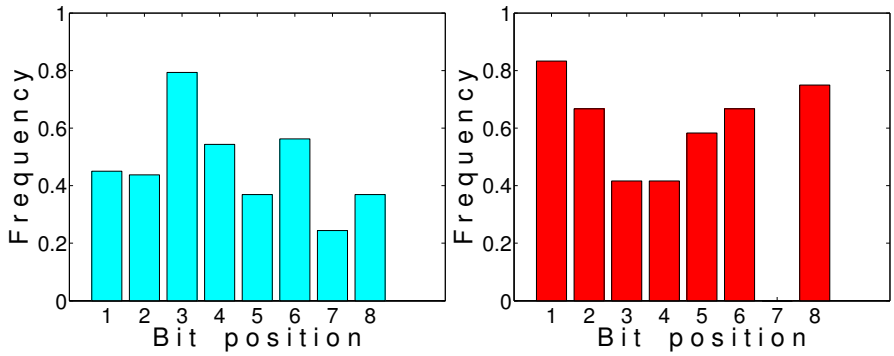
Figure 5.5 only shows the convergence over the first generations according with the number of generations that GA-OS takes to converge. Table 5.6 presents the average number of generations required by each algorithm to find the best solution for each experiment over 10 runs. It is evident from these results that the GA-OS converges much earlier than GA-NS, much earlier than the maximum number of generations allowed (after the fitness stagnates for 10 generations). Since GA-NS fitness is based on novelty, it requires more generations for the search to converge. This behavior highlights the fact that when the search is driven directly by the objective it can become stagnated around local

Table 5.6: Number of generations required by GA-OS and GA-NS to converge for each of the CF circuits: one MOSFET (M1), two MOSFETS (M2) and three MOSFETs (M3). Each row is a different run and the final row shows the average.

	GA-OS			GA-NS		
Run	M1	M2	M3	M1	M2	M3
R1	11	24	20	108	102	140
R2	11	18	11	109	191	201
R3	16	12	13	201	201	97
R4	15	10	21	36	201	97
R5	10	10	22	50	201	113
R6	6	6	6	193	106	176
R7	11	18	11	126	192	70
R8	16	12	13	135	201	201
R9	15	10	21	136	135	201
R10	10	10	22	72	74	201
Aver.	12.1	13.0	16.0	116.6	160.4	149.7

optima. GA-NS certainly finds similar local optima, but the search does not stagnate given its ability to promote diversity and explore other regions of the search space, allowing it to find solutions that might become inaccessible to GA-OS.

For simplicity, and given the quick conversion of GA-OS, we can take a snapshot of the type of solutions found by each algorithm after the first 10 generations. Figures 5.6 - 5.8 compare the composition of the best solutions found by both GA-OS and GA-NS. The figures present frequency histograms, where the height of each bar represents the percentage of runs for which a particular bit in the chromosome was set to a 1 value in the best solution found so far. For instance, if a bar reaches a value 0.5 this it means that a 50% of the best solutions found after 10 generations have a 1 at that particular position within the chromosome. The plots are divided for each experimental configuration, with Figure 5.6 showing the results for the single MOSFET CFs, Figure 5.7 for two MOSFETs and Figure 5.8 for three MOSFETs.



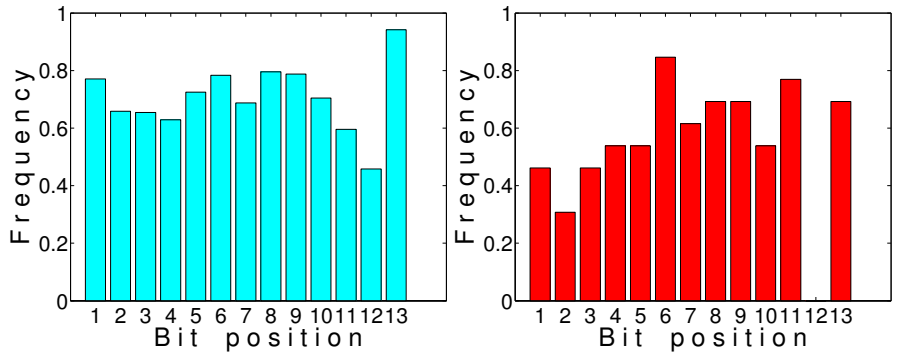
(a) GA-OS (b) GA-NS

Figure 5.6: Histograms showing the average composition of the chromosome of the best solution found by each algorithm after 10 generations, for the single MOSFET circuits.

These figures nicely illustrate our claim, that GA-NS finds solutions that are different from those found by GA-OS. Moreover, that the solutions found, while being different, achieve the same performance based on the objective function. This means that GA-NS explores other areas of the search space, some of which might contain local (or even global) optima that are not accessible to the standard GA-OS.

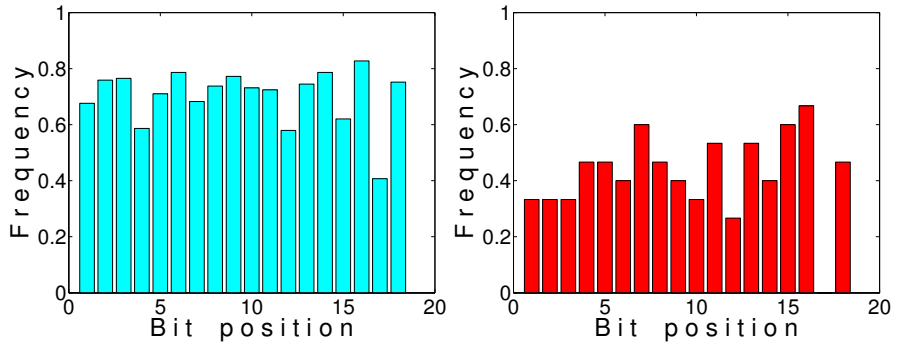
5.4 Conclusions

In this work we use an automatic synthesis approach for analog circuit topologies using a GA, particularly to find topologies for CF circuits. In particular, this work proposes the use of the NS algorithm for circuit synthesis, the first such work in this field. While standard objective-based search assigns fitness and selective pressure based on the domain-specific objective function, NS guides the search by determining fitness based on the uniqueness, or novelty, of each individual solution. In the proposed GA-NS method, the objective function is only used to select the final solution returned by the algorithm, but the



(a) GA-OS (b) GA-NS

Figure 5.7: Histograms showing the average composition of the chromosome of the best solution found by each algorithm after 10 generations, for the two MOSFET circuits.



(a) GA-OS (b) GA-NS

Figure 5.8: Histograms showing the average composition of the chromosome of the best solution found by each algorithm after 10 generations, for the three MOSFET circuits.

search is carried out based on the concept of solution novelty. The experimental results showed that NS allows the algorithm to explore the search space in a different way than a standard GA-OS does. While the

standard approach converged to expected results, the NS approach was able to discover unique solutions for even this well-known and widely studied circuit design problem. Moreover, while the GA-NS algorithm found unique circuits, its performance was equivalent to GA-OS, showing that the NS approach can also produce high-performance solutions even when it omits the objective function from the search process.

Future work derived from this research will focus on the following. First, to optimize the evolved topologies of the CF circuits and to subject them to real-world experimental validation. Second, apply the NS paradigm to synthesis other specialized circuits of interest in the field of electronic design automation. Third, we can enhance the NS approach by attempting to force the search away from specific areas of the search space. For example, it should be possible to seed the population with previously known designs that should be avoided by the search, since they are not as interesting. In this way, the NS algorithm could be used to explicitly search for circuits that are unique in electronic literature.

6

Generalization of NS-based GP Controllers for Evolutionary Robotics

ABSTRACT — Over recent years, evolutionary computation research has begun to emphasize the issue of generalization. Instead of evolving solutions that are optimized for a particular problem instance, the goal is to evolve solutions that can generalize to various different scenarios. This chapter compares objective-based search and novelty search on a set of generalization oriented experiments for a navigation task using grammatical evolution (GE). In particular, this chapter studies the impact that the training set has on the generalization of evolved solutions, considering: (1) the training set size; (2) the manner in which the training set is chosen (random or manual); and (3) if the training set is fixed throughout the run or dynamically changed every generation. Experimental results suggest that novelty search outperforms objective-based search in terms of evolving navigation behaviors that are able to cope with different initial conditions. The traditional objective-based search requires larger training sets and its performance degrades when the training set is not fixed. On the other hand, novelty search seems to be robust to different training sets, finding general solutions in almost all of the studied conditions with almost perfect generalization in many scenarios.

GENERALIZATION OF NS-BASED GP CONTROLLERS FOR EVOLUTIONARY ROBOTICS

6.1 Introduction

Generalization is considered to be a cornerstone concept of mainstream machine learning research (ML) (Kaelbling et al., 1996). In general, one of the most important issues in supervised and semi-supervised ML is the ability of an algorithm to generate general models or solutions, starting from a finite set T of training instances (Kushchu, 2002b). Particularly in robotics, generalization is extremely desirable when addressing reinforcement learning (RL) tasks (Kaelbling et al., 1996). RL is the problem faced by an agent that must learn a behavior through trial-and-error interactions with an environment (Kaelbling et al., 1996). Another research area where generalization plays an important role is evolutionary robotics (ER), a form of RL that uses an evolutionary algorithm (EA) to carry out the learning process.

The present chapter studies generalization in ER where the learning is carried out by grammatical evolution (GE) (O'Neill and Ryan, 2001; Dempsey et al., 2009; Georgiou, 2012) a form of EA based on the general principles of the genetic programming (GP) paradigm (Koza, 1992b). Although generalization is closely related to robustness, they refer to separate concepts. Robustness can be defined as resiliency to slight changes in the inputs, due to noise or other random processes, while generalization should be understood as the ability of a solution to solve unseen cases (data or conditions not used during the learning process) with a satisfactory degree of success. Therefore, a solution is considered to be general if it is able to solve a large number of unseen cases. The difficulty of generating general solutions stems from the fact that good performance on the training set T cannot guarantee equal performance on unseen inputs, what is also referred to as the bias-variance dilemma (Duda et al., 2000; Bishop, 2006).

The generalization issue has begun to receive much attention in ER over recent years (Francone et al., 1996; Banzhaf et al., 1996; Mahler et al., 2005; Kushchu, 2002a,b; Uy et al., 2010; Langdon and Poli, 2001; Gonçalves and Silva, 2011a,b; Naik and Dabhi, 2013). On the other hand, most works using GP study generalization in standard supervised learning problems, which are usually defined by a training set

$T = \{(x_i, y_i)\}$ with $i = 1,...,n$, where each pair (x_i, y_i) represents an expected input and desired output (also referred to as a fitness-case in GP literature). The goal is to find a model K that minimizes an objective function based on the difference between $K(x_i)$ and y_i $\forall i$. However, ER relies on RL (Nolfi and Floreano, 2000; Nelson et al., 2009), where an autonomous agent is placed within an unknown environment and it must learn to carry out a specific task, such as navigating towards a desired target position. Similar problems have also been studied in GP literature, since the original proposal of the Santa-Fe trail (SFT) by Koza (Koza, 1992b). In this domain, the objective function can be defined in different ways (Kushchu, 2002a,b; Robilliard et al., 2006; Doucette and Heywood, 2010; Georgiou and Teahan, 2010; Urbano and Loukas, 2013; Nelson et al., 2009), but basically it must measure the degree by which the desired high-level task is fulfilled. The training and testing sets are defined by the set of initial conditions specified for the agent within the environment (such as location and orientation), as well as the characteristics of the environment and the amount of time it has to complete the task. Generalization in this domain still merits further research.

This chapter presents an extensive study of generalization in a GE-based ER system for an autonomous agent that must navigate within a 2D environment and reach a predefined and fixed target. The goal is to study the effect that the size of the training set has on the generalization ability of a GE-based learning system. Moreover, this chapter presents an extensive comparison between two approaches towards defining fitness and applying selective pressure during evolution.

First, the standard approach in GE, as in most other EAs, is to define fitness proportionally to the performance of the solutions as expressed by the objective function. This measure is used to quantify fitness and to guide the search, we will refer to the traditional approach as classical or objective-based search. For instance, for a navigation problem the objective function could be defined as the Euclidean distance between the final position of the agent and the target. In this chapter, two objective functions are tested to measure performance: (1) f_1 computes the Euclidean distance between the agents final position and the target

(Lehman and Stanley, 2008), and (2) f_2 considers both the Euclidean distance and the length of the agents trajectory without considering any repeated position (Georgiou, 2012).

Second, the novelty search (NS) algorithm proposes a different perspective to define fitness (Lehman and Stanley, 2008). NS was motivated by the goal of dealing with deceptive fitness-landscapes (Goldberg, 1987) in ER, where the objective function tends to guide the search away from the global optimum. NS replaces the objective function to compute fitness with a measure that quantifies how unique, or novel, an evolved solution is with respect to all previously found solutions during the search. However, instead of using genotypic diversity, a common tool in EAs (Nicoară, 2009; Burke et al., 2004), NS uses a description of what each solution does within its environment, what can be referred to as a behavior descriptor (Lehman and Stanley, 2008; Trujillo et al., 2011b, 2008a; Mouret and Doncieux, 2012). In this way, selective pressure in NS pushes the search towards novel behaviors, allowing the search to avoid local optima. In this chapter we experimentally compare two different behavior descriptors including the one proposed in (Lehman and Stanley, 2010a). NS has been widely used in navigation and other ER problems with strong results (Lehman and Stanley, 2008, 2010a, 2011a; Urbano and Loukas, 2013; Gomes et al., 2013), and has recently been extended to more traditional ML problems with GP (Naredo and Trujillo, 2013; Martínez et al., 2013). However, most works in ER have not studied the issue of generalization in NS, or how the size of the training set affects it.

This chapter also studies the impact of how the training set is determined, considering several different variants. First, the training set is constructed randomly, using a different number of instances, to evaluate how the size of the training set impacts generalization. This random strategy is compared with manually selected initial conditions, to evaluate the bias that a human designer introduces into the learning process and to compare the newly found results with our previous work (Urbano et al., 2014b). Moreover, recent works in GP suggest that varying the training set during the evolutionary process can improve generalization (Gathercole and Ross, 1994; Gonçalves et al., 2012; Gonçalves

and Silva, 2013), but this has only been validated in traditional ML problems, not in ER. Therefore, this chapter also studies the effect of varying the training set during evolution, instead of keeping the training set fixed during the search. The training set is either set statically for the entirety of the evolutionary process or it is randomly varied at the beginning of each generation, or at at the beginning of every run and fixed then. It is assumed that by varying the training set the system might be able to cope with a larger set of different scenarios. In

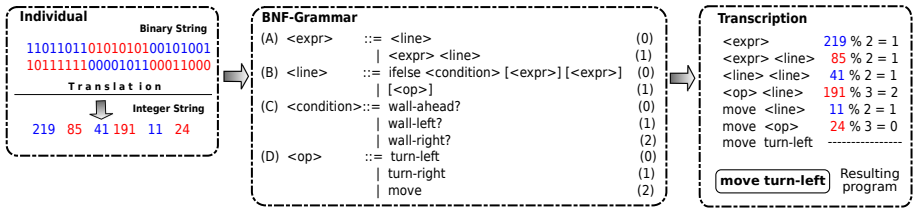


Figure 6.1: Example of a GE genotype-phenotype mapping process, where the binary genotype is translated into an integer string, which is then used to select production rules from a predefined grammar. The derivation sequence of the program is shown on the right, all codons (typically groups of 8 bits) were used but wrapping was unnecessary in this example.

summary, the main contributions of this chapter can be summarized as follows:

i. Experimental evaluation of the effects that the training set has on the generalization ability of two approaches towards determining fitness for a navigation problem within a 2D environment solved with GE: (1) novelty-based, and (2) objective based; compared with random search as a baseline. Moreover, two different objective functions are considered for objective-based search, and two different behavior descriptors are evaluated to measure solution novelty.

ii. The size of the training set is varied, from the simplest case with a single fitness-case, to a relatively large set with 60 fitness-cases. Each training instance defines different initial conditions for the

agent within the environment, specifying its position and orientation. Generalization is then evaluated based on the performance achieved on a test set of 100 randomly generated instances, considering both the quality given by the objective functions and the percentage of cases in which the agent reaches the target (referred to as hits).

iii. The manner in which the training set is determined is also evaluated, considering three different approaches: (1) randomly setting the training set at the beginning of each run; (2) randomly changing the training set at the beginning of each generation; or (3) manually setting the training set based on expert knowledge, which is fixed for all runs.

The remainder of this chapter is organized as follows. Section 6.2 reviews the main background concepts. Afterwards, Section 6.3 presents the experimental set-up of our work, outlining all of the considered variants and performance measures. Section 6.4 presents and discusses the main experimental results. Finally, Section 5.4 presents the conclusions and future research lines that can be derived from this work.

6.2 Background

This section reviews background topics related to the present work. First, we quickly describe GE, which is the basis of our ER system. Second, we discuss the issue of generalization in GP literature in general. Third, we present the NS algorithm and review previous works that study generalization.

6.2.1 *Grammatical Evolution*

As stated before, GE is a variant of GP which is capable of evolving computer programs in an arbitrary language defined by a Backus-Naur Form (BNF) grammar (O'Neill and Ryan, 2001). Programs are indirectly represented by variable length binary genomes, and built by

a developmental process. The linear representation of the genome allows for the application of genetic operators such as crossover and mutation in the manner of a typical genetic algorithm, unlike tree-based GP (Koza, 1992b). Starting with the start symbol of the grammar, each individual's chromosome contains in its codons (typically groups of 8 bits) the information necessary to select and apply the grammar production rules to construct the final program.

Production rules for each non-terminal are indexed starting from 0. To select a production rule for the left-most non-terminal of the developing program, from left to right, the next codon value in the genome is read and interpreted using the formula $I = c\%r$, where c represents the current codon value, % represents the modulus operator, and r is the number of production rules for the left-most non-terminal. The corresponding production rule in the I-th index will be used to replace the left-most non-terminal. If, while reading codons, the algorithm reaches the end of the genome, a wrapping operator is invoked and the process continues reading from the beginning of the genome. The process stops when all of the non-terminal symbols have been replaced, resulting in a valid program. In the wrapping process, if an individual fails with the condition of replacing all of the non-terminal symbols after a maximum number of iterations, then it is considered as an invalid individual and it is penalized with the lowest possible fitness. The mapping process is illustrated with an example in Figure 6.1, where we use a grammar to describe maze navigation programs written in Netlogo (Wilensky, 1999).

6.2.2 *Generalization in Genetic Programming*

Generalization in evolutionary learning has not received the same amount of attention as in traditional ML, but has been addressed by some works. In (Kushchu, 2002a), Kushchu studies the generalization ability of GP, viewed as forming a hypothesis that goes beyond the observed training instances. Training is the process of finding a solution during the learning process, and generalization occurs when a learner

takes a decision and prefers the correct hypotheses over others. This preference can be due to a bias in the learning algorithm or prior knowledge regarding the problem domain. According to Kushchu, there are two major types of bias; representational and procedural. Representational bias is given by the language used to describe the hypotheses, and procedural bias is given by the search process.

Regarding the latter, to promote generalization in ML it is necessary to gather more knowledge about the problem, in such a way that the learning system increases its likelihood of success. Early research in generalization with GP includes (Francone et al., 1996), which provides an evaluation of a compiled GP system on supervised classification problems, showing that GP can achieve comparable generalization to other ML paradigms. Some researchers have correlated the problem of excessive code growth in GP, normally referred to as bloat (Vanneschi et al., 2010), with a lack of generalization. Suggesting that when a program grows too large it might overfit the training data and thus fail to generalize (Rosca, 1996). However, (Mahler et al., 2005) and (Vanneschi et al., 2010) showed that controlling bloat and limiting program growth does not necessarily improve generalization.

More recently, (Uy et al., 2010) examines the impact of semantically aware search operators, showing that this new class of genetic operators can help improve overall performance and generalization of a GP system for symbolic regression problems. Similarly, (Gonçalves et al., 2015) has shown that other semantic operators might help improve generalization in GP for classical ML problems. Other approaches towards improving generalization can be found in (Gonçalves and Silva, 2013; Martínez et al., 2013; Spector, 2012). These works propose fitness case sampling methods for GP, where the set of fitness-cases used to determined fitness at each generation is constructed by sub-sampling the original training set. Results indicate that these algorithms can help improve generalization, but further research is still required. In (Castelli et al., 2010) the authors compare several flavors of GP based on their generalization abilities, determinining that multi-objective optimization seems to improve generalization. Finally, (Trujillo et al., 2011c)

and (Castelli et al., 2011) have attempted to derive measures of functional complexity to be used as predictors of generalization in GP.

What is important to note from this high-level overview, is that almost no work regarding generalization in GP has studied GE or considered RL that does not use a static training set. Moreover, all of these works have focused on traditional objective-based search and none have addressed these issues with NS, an alternative that is discussed next.

6.2.2.1 *Generalization with Novelty Search*

Most recent works that have applied NS to ER (Lehman and Stanley, 2010a, 2011a; Georgiou and Teahan, 2010) have studied the issue of deception, but have not provided in depth results regarding generalization. These works did not test if the evolved behaviors were able to generalize to different initial conditions, target points or environments. Velez and Clune in (Velez and Clune, 2014), on the other hand, did transfer maze navigation controllers evolved with NS to new scenarios. Their experiments in neuro-evolution have confirmed that agents using NS can learn general exploration skills. The transferred robots were able to perform much better than randomly generated agents, but did not outperform transferred robots evolved by a standard objective-based EA. More recently in (Shorten and Nitschke, 2015) the authors present a comparation of an objective-based search, NS and a combination of both approaches to evolve robot neurocontrollers that solve a test set of 1,000 mazes using a static training set of 100 mazes. The results in (Shorten and Nitschke, 2015) show that NS and the NS-objective combination approaches yield comparable generalized maze navigation behaviors, and that both outperform the objective-based approach, but they did not analyse the effect of the training set size.

Kushchu (Kushchu, 2002b) has identified the SFT problem as an example were objective-based search will lead towards brittle solutions with GP. His experimental results suggest that a successful agent won't perform well on small variations of the problem. Therefore, Kushchu proposed to train an agent on a set composed of variations of the SFT

using an objective-based approach, sharing similar characteristics, and tested the learned behaviors evolved with different set of similar trails. He was able to successfully evolve general trail following agents. In (Lehman and Stanley, 2010a) using standard GP, and in (Urbano and Loukas, 2013) using GE, NS was applied successfully to the SFT problem, a known deceptive problem in GP.

Doucette and Heywood (Doucette and Heywood, 2010) have empirically evaluated the impact of NS on generalization performance for the SFT, using the SFT as a single training set and a test set of similar trails. They compared different ways of assigning selective pressure, combining novelty and the objective into a single measure, or using each one independently; their results suggest that no method could produce individuals that solved both the training and testing sets. However, results showed that the classical objective-based GP achieved the best train and test performances, but programs evolved by NS alone had better generalization abilities. In contrast, in two other works (Lehman and Stanley, 2010a; Urbano and Loukas, 2013) NS outperformed objective-based search. Therefore, it is reasonable to state the results so far are inconclusive.

Important open issues that have not been considered include the effect that the training set size has on generalization, as well as the manner in which the training set is constructed, or how different objectives and descriptors might impact the search. However, (Trujillo et al., 2011b) showed that by promoting behavioral diversity during the search for navigation behaviors, an EA could find several different solutions to the same problem, and that such solutions exhibited better generalization abilities when placed in an unknown environment. Those results correlate with the hypothesis that the search for novel and unique behaviors can help an evolutionary process identify general solutions.

6.3 Experimental Setup

This section summarizes our proposed experimental work, to study the impact of the training set on the generalization ability of a GE-based system using standard objective-based search and NS. A high level view of our experimental work is depicted in Figure **??**, consisting of: (1) a navigation task; (2) different training set sizes; (3) different training set selection approaches; (4) two different objective functions; (5) three different search strategies (objective, novelty and random); and (6) two different behavior descriptors for NS. Each aspect is described next.

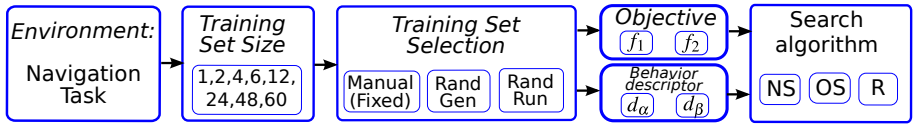
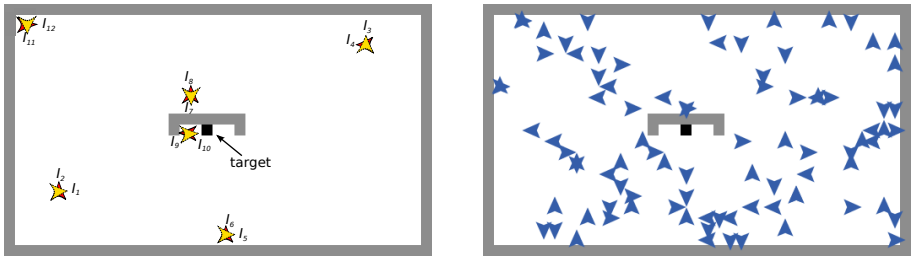


Figure 6.2: High level description of the proposed experimental work, where the environment is a navigation task and the experiments consider several sizes of the training set from 1 to 60 instances. The choices for the training set are: fixed, randomly chosen each generation (Rand-Gen) or every run (Rand-Run). Two objective functions f_1 and f_2 are implemented for different experiments, at the same time two different quality measures are used for both objective functions. The algorithms used are novelty search (NS), objective-based search (OS), and random search (R). NS uses two descriptors of the behaviors: d_α and d_β, which are implicitly related to objective functions f_1 and f_2 respectively.

6.3.1 *Navigation Task and Environment*

The learning task is a navigation problem for an autonomous agent situated in a 2D discrete environment that must reach a static target. The environment is shown in Figure 6.2, it is a rectangular grid of 39×23 cells considering the outer walls, the target is represented by a black square and it is surrounded by an U-like wall composed by 10



(a) **Manual Training Set** (b) **Test Set**

Figure 6.3: Figure (a) shows the learning environment, where the target is depicted by a black square and 12 manually chosen initial conditions for each artificial agent (labeled from I_1 to I_{12}). These instances are grouped into 6 pairs represented as two triangles with different color, and each pair share the same location (x, y) but different orientations (North, East, South, or West). Figure (b) shows 100 initial conditions, which were randomly generated to be used as the test set for all experiments.

cells, so the total cells that an agent can visit is $(37 \times 21) - 11 = 766$. This environment is similar to the one used in (Lehman and Stanley, 2010a), where it is named as the 'medium' (difficulty) maze. Similar to the present work, (Lehman and Stanley, 2010a) compared the performance of GP using three different approaches to guide the search: objective, novelty, and a random fitness. However, in (Lehman and Stanley, 2010a) the training set always contained a single starting point for the agent (the top-left corner), only allowed 100 moves for the agent and did not consider a test set.

In this scenario, each training (or testing) instance is defined by the pair $I_i = (\mathbf{x}_i, \theta_i)$, that defines the initial position $\mathbf{x}_i = (x, y)$ of the agent within the grid environment specified by the row x and column y, and the initial orientation θ_i which can be four possible values: North (N), South (S), West (W) and East (E).

The BNF grammar that defines the space of possible programs is shown in Figure 6.1, which is used to perform the genotype to pheno-

type mapping. The agent has 3 boolean functions as sensors, which each of them is formulated as a question related with the location of a near wall respect to the agent; *wall-ahead?*, *wall-left?*, and *wall-right?*. It can also perform three actions: *move* (moves 1 unit forward, if not blocked by a wall), *turn-left* (90 degrees counter clockwise) and *turn-right* (90 degrees clockwise). The program will be repeatedly executed until the agent hits the target or reaches the maximum number of moves. The agent succeeds if it reaches the target, this is referred to as a hit.

All experiments in this study are performed using the jGE library (Georgiou and Teahan, 2006), a Java implementation of GE, and jGE Netlogo (Georgiou and Teahan, 2010) which is a Netlogo extension of the jGE library. NetLogo[1]is a multi-agent programmable modeling environment, and it was extended with an implementation of the NS algorithm.

6.3.2 *Training and Testing Set Size*

To the best of our knowledge, previous works have not studied the effect that the training set size has on generalization for a navigation problem in ER. Here, we consider seven different training set sizes, of: 1, 2, 6, 12, 24, 48, and 60 instances. Moreover, to evaluate generalization a test set is needed, thus here the testing set is composed by 100 randomly chosen initial conditions, shown in Figure 6.2(b). We believe that given the size of the environment, 100 instances is sufficient to evaluate the generalization of the evolved solutions.

6.3.3 *Training Set Selection*

We evaluate three different approaches to choose the instances that compose the training set: (1) a set of randomly chosen instances de-

[1] Netlogo is authored by Uri Wilensky and developed at the Northwestern's Center for Connected Learning and Computer-Based Modeling (CCL). You can download it free of charge in: https://ccl.northwestern.edu/netlogo/.

termined at the beginning of each run; (2) a set of randomly chosen instances chosen at the beginning of each generation; and (3) a set of manually chosen instances used in all runs.

The manual approach undoubtedly introduces a human bias into the learning process, which might compromise any conclusions drawn from the experimental results. If the problem requires specific initial conditions then this is not a issue, but determining how to construct the training set for an arbitrarily complex environment is in no way a trivial task. Moreover, as the results of this chapter will show, using random training sets not only simplifies the problem formulation but it actually improves the quality of the results.

The first two approaches, (1) and (2), remove this bias and allow us to evaluate the effect of training set size irrespective of the initial positions of the robot. One drawback about randomizing the selection of the training set in our experimental setup is that they may coincide with those used for testing. But even in the worst case with a training set size of 60 and using 100 testing instances, there will be less than 2% of overlap between both sets, and will be mostly particularly for the smaller training sets. The second approach (2) induces a dynamic learning process with a non-static fitness landscape, a scenario where solution diversity and generalization would seem to be necessary. For instance, recent work (Gonçalves and Silva, 2011b) has suggested that changing the fitness cases used at each generation can help improve generalization and reduce overfitting in GP.

6.3.4 *Objective Functions*

For the evaluation of the evolved programs two objective functions are considered: f_1 and f_2. First, f_1 computes the inverse of the sum of Euclidean distance plus one between the final position of the agent α and the target t (Lehman and Stanley, 2008), given by

$$f_1 = \frac{1}{1 + \text{dist}(\alpha, t)} \tag{6.28}$$

where $dist()$ represents the Euclidean distance in this work. Conversely, f_2 considers both the Euclidean distance and the length of the agent's trajectory. In this case, for a discrete environment represented by a 2D grid the trajectory is represented by the number of visited (non-repeated) cells β before reaching the target, or if the target is not reached when the total amount of allowed moves is reached; it is computed by

$$f_2 = \frac{1}{1 + \frac{\text{dist}(\alpha,t)}{\beta}} \,. \tag{6.29}$$

For both objective functions and a training set with n instances, each agent is evaluated n times, once for each instance, and the final value is the average score across all the evaluations. Similarly, the quality score on the test set will be the average score across all 100 instances. It is important to note that f_2 scale higher values than f_1, for this reason a fair quality measure for both functions is the binary verification that an agent reaches the target, this is named as a hit.

6.3.5 *Search Algorithms*

The proposed generalization experiments compare the performance of GE using objective (OS), novelty (NS), and random-based search (R). The goal is to determine which form of selective pressure can increase the generalization abilities of the GE-system in the proposed problem. Moreover, it is important to determine if improving generalization has the negative effect of reducing overall training performance, which will also be evaluated.

To implement NS in this work we use two behavior descriptors d_α and d_β, which are implicitly based on f_1 and f_2 respectively. The first descriptor builds a behavior vector $d_\alpha = (\alpha_1,...,\alpha_n)$, where each α_i is the final position of the agent for training instance I_i. The second descriptor $d_\beta = (\beta_1,...,\beta_n)$ is composed by values β_i that represent the number of visited (non-repeated) cells for training instance I_i.

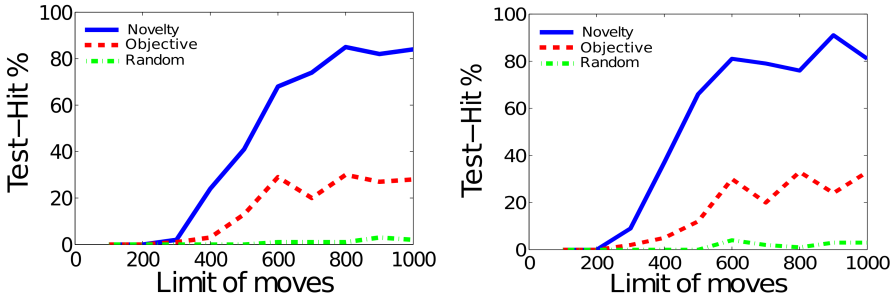
Table 6.1: A general description of the algorithms and measures used in the experimental set-up.

	Name	Description
Training set selection	g	Randomly chosen at the beginning of each generation
	r	Randomly chosen every run.
	Manual	Manual selection of training cases.
Algorithms	Ng	Novelty-based search with a training set of randomly chosen instances chosen at the beginning of each generation.
	Nr	Novelty-based search with a training set of randomly chosen instances determined at the beginning of a run.
	Og	Objective-based search search with a training set of randomly chosen instances chosen at the beginning of each generation.
	Or	Objective-based search with a training set of randomly chosen instances determined at the beginning of a run.
	R	Random-based search instances determined at the beginning of a run.
Measures	f_1	Fitness function based on the Euclidean distance.
	f_2	Fitness function based on the Euclidean distance & visited cells.
	d_α	Descriptor using the final position of the agent, related with f_1.
	d_β	Descriptor using the number of of visited (non-repeated) cells, related with f_2.

In summary, Table 6.1 presents all of the algorithmic variants tested in this work and the different comparative measures. Finally, it is important to point out that R does not use the training set, so only a single version of R is required.

6.3.6 *Limit for Allowed Moves*

The limit of allowable moves simulates the energy (i.e. battery life) that an agent has to solve the navigation task. In this work, we chose to determine a fixed limit for all the experimental conditions. To do



(a) Hit-Test for f_1 (OS) and d_α (NS)

(b) Hit-Test for f_2 (OS) and d_β (NS)

Figure 6.4: Performance comparison of objective, novelty and random-based search using a fixed training set of 12 manually chosen instances and different number of allowed moves, from 100 to 1000. The plots show the average hit percentage from 100 runs using the test set for both fitness functions f_1 (a) and f_2 (b) and both behavior descriptors d_α (a) and d_β (b).

so, we used twelve manually selected initial conditions as a training set for consistency with our previous work (Urbano et al., 2014b), and performed experiments varying the total number of allowable moves, starting from 100 through 1000 moves, with steps of 100. The algorithms were evaluated based on the percentage of hits achieved on the testing sets. The results are shown in Figure 6.3, which is the average over 100 independent runs. In these plots, it is possible to see that for objective-based search hit percentage reaches a maximum value at 600 moves and levels off after that. For NS, performance still increases after 600 moves, but it still reaches almost maximum performance at 600. Random search, on the other hand, performs poorly in all cases. Therefore, for all of the following experiments we set the limit for allowed moves at 600.

The parameters used in the experiments are summarized in Table 6.2, which are standard values for GE systems (Urbano and Loukas, 2013; Robilliard et al., 2006). The experiments were executed in 100 runs with a population of 250 individuals for 50 generations. For in-

Table 6.2: Parameters used for the experimental work. Codons-min and Codons-max are the minimal and maximal number of codons in the initial random population of the GE search.

Parameter	Value	Parameter	Value
Codon-size	8	Generational	YES
Codons-min	15	Mutation prob	0.01
Codons-max	25	Elitism	10%
Number of runs	100	NS archive	NO
Wraps	10	Crossover	codon-crossover
Number of individuals	250	NS k–neighbors	3
Crossover prob.	0.9	Selection	Roulette Wheel

valid individuals, a value of 0 is given for both fitness and novelty. After some preliminary exploration, we set the number of neighbors used to compute the novelty score to $k = 3$, invalid individuals were not considered, and we did not use an archive as suggested in (Lehman and Stanley, 2008), since preliminary experiments showed that it did not improve performance, as well the success of NS without using an archive combined with a short value for k is reported in (Gomes et al., 2015).

6.4 Results and Analysis

This section presents the experimental results divided into three subsections. The first one considers the randomly chosen training sets, the second considers the manually selected training set, while the final one presents an analysis of statistical significance.

6.4.1 *Results for Randomly Chosen Training Sets*

Table 6.3 summarizes the results for all of the experiments that used randomly chosen training sets, organized by the number of training instances (columns 1), the manner in which the training set was selected

(column 2) and the type of selection pressure (column 3). Moreover, results are presented for the average training and testing performance over 100 independent runs. First, for training we show the average fitness of the best solution found, remembering that for NS when f_1 is reported learning was performed with the d_α descriptor and when f_2 is reported d_β was used. For testing, the fitness of the best solution found is computed with the test set, and the percentage of test hits is also presented: $H1$ for function f_1 (objective) and descriptor d_α (novelty); and $H2$ for function f_2 (objective) and descriptor d_β (novelty).

Table 6.3: Summary of the experimental results for all of the variants showing the average over a 100 runs, for the configurations that used a random training set: the training set size is from 1 to 60; Sel is the manner in which the training set is set, for every generation (Gen) or for every run (Run); and three search strategies are considered, Objective, Novelty, and Random. Results are given for training and testing performance, for training the performance of the best solution found is evaluated on each objective function (f_1 and f_2) as well as for testing. The percentage of hits, $H1$ and $H2$, is shown only for the test set. In all cases Bold indicates the best performance.

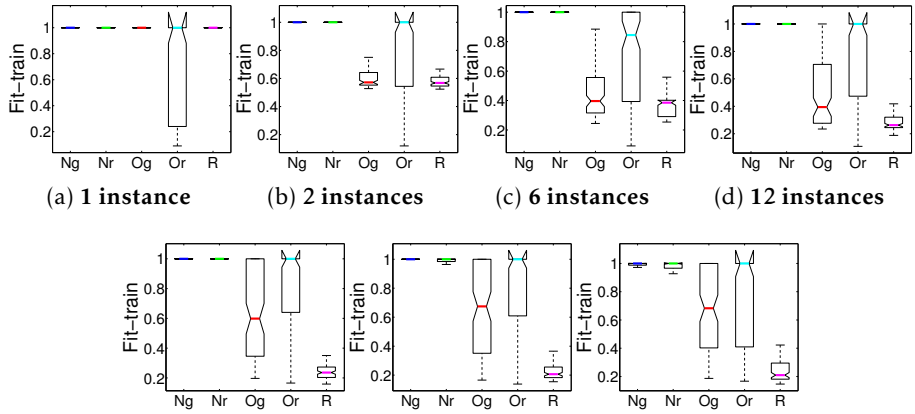
			Training		**Test**			
Size	**Sel**	**Fitness**	f_1	f_2	$H1$	$H2$	f_1	f_2
1	Rand-Gen	Novelty	**1.0000**	**1.0000**	**12%**	**16%**	**0.1966**	**0.7655**
		Objective	0.9691	0.9989	4%	7%	0.1227	0.7246
	Rand-Run	Novelty	**1.0000**	**1.0000**	**27%**	**29%**	**0.3315**	**0.8559**
		Objective	0.7068	0.9945	12%	22%	0.1954	0.8326
		Random	0.5386	0.9689	8%	6%	0.1610	0.7848
2	Rand-Gen	Novelty	**1.0000**	**1.0000**	**45%**	**43%**	**0.5073**	**0.9206**
		Objective	0.6576	0.9870	10%	18%	0.1848	0.8939
	Rand-Run	Novelty	**0.9960**	**1.0000**	**56%**	**59%**	**0.6080**	**0.9504**
		Objective	0.7296	0.9824	39%	38%	0.4439	0.9272
		Random	0.4485	0.9460	10%	8%	0.1826	0.8771
6	Rand-Gen	Novelty	**0.9854**	**1.0000**	**92%**	**88%**	**0.9289**	**0.9914**
		Objective	0.4824	0.9655	15%	35%	0.2254	0.9435
	Rand-Run	Novelty	**0.9733**	**0.9998**	**92%**	**91%**	**0.9280**	**0.9918**

Continued on next page...

Table 6.3 – continued from previous page

			Training		Test			
Size	**Sel**	**Fitness**	f_1	f_2	$H1$	$H2$	f_1	f_2
		Objective	0.7209	0.9744	56%	48%	0.6057	0.9611
		Random	0.2908	0.9335	7%	13%	0.1524	0.9190
12	Rand-Gen	Novelty	**0.9730**	**1.0000**	**94%**	**98%**	**0.9485**	**0.9979**
		Objective	0.5131	0.9664	29%	39%	0.3609	0.9524
	Rand-Run	Novelty	**0.9731**	**0.9999**	**94%**	**97%**	**0.9501**	**0.9978**
		Objective	0.7520	0.9784	66%	62%	0.6964	0.9735
		Random	0.2524	0.9274	8%	12%	0.1673	0.9223
24	Rand-Gen	Novelty	**0.9548**	**0.9999**	**92%**	**99%**	**0.9306**	**0.9990**
		Objective	0.6375	0.9643	49%	40%	0.5419	0.9540
	Rand-Run	Novelty	**0.9650**	**0.9999**	**94%**	**99%**	**0.9509**	**0.9989**
		Objective	0.8099	0.9740	75%	56%	0.7749	0.9721
		Random	0.2682	0.9253	13%	8%	0.2126	0.9188
48	Rand-Gen	Novelty	**0.9739**	**1.0000**	**96%**	**100%**	**0.9642**	**0.9997**
		Objective	0.6727	0.9660	58%	48%	0.6199	0.9597
	Rand-Run	Novelty	**0.9614**	**0.9999**	**95%**	**99%**	**0.9521**	**0.9996**
		Objective	0.8139	0.9742	78%	58%	0.8068	0.9727
		Random	0.2379	0.9267	13%	12%	0.2102	0.9244
60	Rand-Gen	Novelty	**0.9648**	**0.9998**	**95%**	**99%**	**0.9581**	**0.9993**
		Objective	0.6842	0.9640	58%	41%	0.6223	0.9576
	Rand-Run	Novelty	**0.9530**	**0.9996**	**95%**	**99%**	**0.9530**	**0.9993**
		Objective	0.7390	0.9748	70%	58%	0.7336	0.9737
		Random	0.2550	0.9261	15%	10%	0.2313	0.9237

To gain a better appreciation of the differences of each method shown in Table 6.3, Figures 6.4-6.9 present box plots that show the median and spread over the first and third quartiles of the 100 runs. First, for objective function f_1 and its related behavior descriptor d_α for NS: Figure 6.4 shows a comparison of the best training fitness from each run; Figure 6.5 compares the test fitness of the best solution found; and Figure 6.6 shows the percentage of hits on the test set. Similar results are summarized for objective function f_2 and descriptor d_β in Figures 6.7, 6.8, and 6.9, respectively for training fitness, test fitness and test



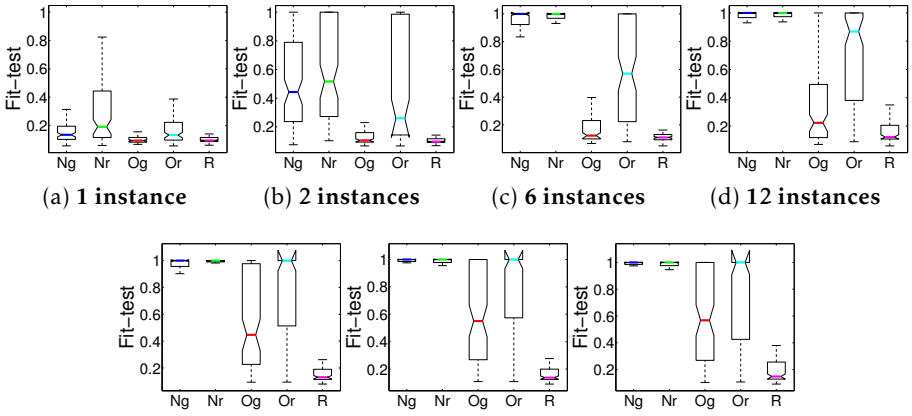
(a) **1 instance** (b) **2 instances** (c) **6 instances** (d) **12 instances**

(e) **24 instances** (f) **48 instances** (g) **60 instances**

Figure 6.5: Box plot comparison of the best solution found using training fitness with function f_1 and descriptor d_α. 'N' stands for novelty-based search, 'O' for objective-based search, and 'R' stands for random-based search. Sub-index 'g' and 'r' stand for the method used to choose randomly the training set, either every generation or every run, respectively. Figures are sorted in ascending order according to the number of instances used in the training sets, from 1 to 60.

hits. All of these figures present results for five of the algorithms summarized in Table 6.1, these are: Ng, Nr, Og, Or and R.

These results exhibit some clear trends. First, let's consider training performance for both objective functions, shown in Figures 6.4 and 6.7. NS achieves good performance irrespective of the size of the training set or the manner in which the training set is chosen (per run or per generation) for both objectives. On the other hand, objective-based search shows worse performance than NS, and Or is relatively better than Og in both figures, suggesting that it is better to keep the training set static for all the generations of the run when using this form of selective search. Random search clearly shows worse performance, but R is basically equivalent to Og for small training sets.

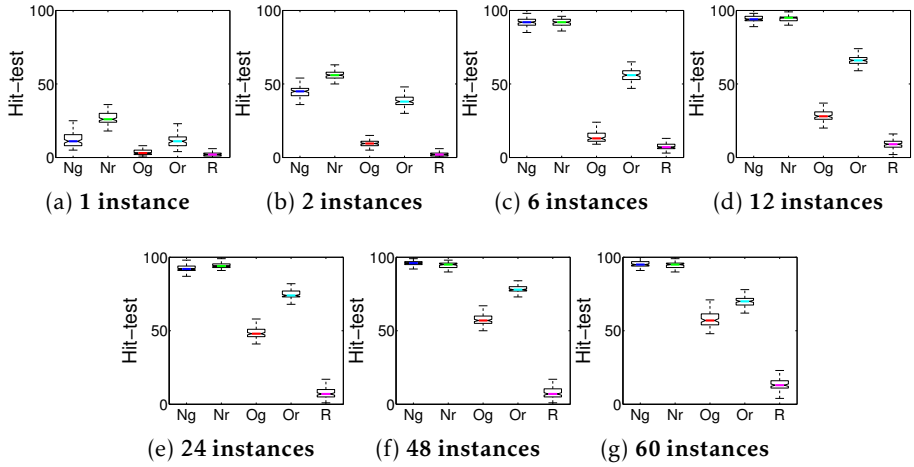


(a) **1 instance** (b) **2 instances** (c) **6 instances** (d) **12 instances**

(e) **24 instances** (f) **48 instances** (g) **60 instances**

Figure 6.6: Box plot comparison of the best solution found using test fitness with function f_1 and descriptor d_α. 'N' stands for novelty-based search, 'O' for objective-based search, and 'R' stands for random-based search. Sub-index 'g' and 'r' stand for the method used to choose randomly the training set, either every generation or every run, respectively. Figures are sorted in ascending order according to the number of instances used in the training sets, from 1 to 60.

When we consider testing performance, the following is suggested by Figures 6.5 and 6.8. First, NS clearly improves on the test set when the training set size increases. In fact, for training sets larger than 6 instances performance is almost perfect for both Ng and Nr. Similarly, objective-based search also improves proportionally to the training set size, but it does not reach the same level of performance than NS. Moreover, Or always outperforms Og, confirming that objective-based search struggles with a dynamic objective function. All of these observations are valid for both objective functions, this means that these trends seem to not depend on the manner in which performance is measured, at least for the two methods considered here. Finally, it is clear that random search cannot reach the same level of performance. This result is of interest, since many researchers erroneously conflate



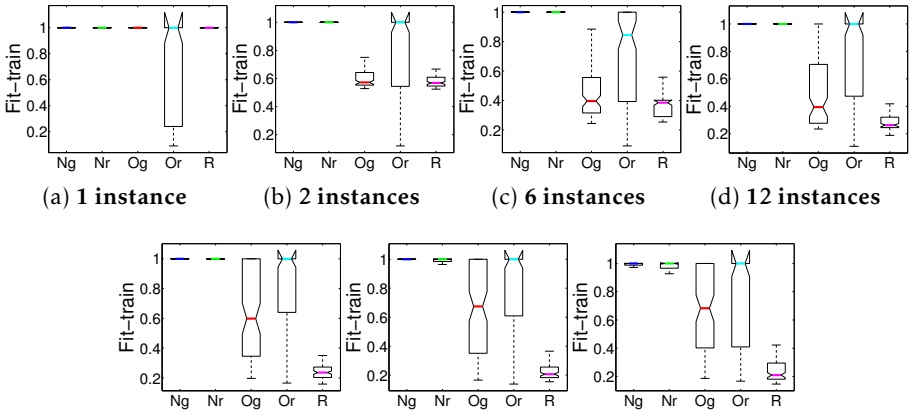
(a) **1 instance** (b) **2 instances** (c) **6 instances** (d) **12 instances**

(e) **24 instances** (f) **48 instances** (g) **60 instances**

Figure 6.7: Box plot comparison of the percentage of testing hits by the best solution found using function f_1 and descriptor d_α. 'N' stands for novelty-based search, 'O' for objective-based search, and 'R' stands for random-based search. Sub-index 'g' and 'r' stand for the method used to choose randomly the training set, either every generation or every run, respectively. Figures are sorted in ascending order according to the number of instances used in the training sets, from 1 to 60.

NS with a random search process, while our results in agreement with (Velez and Clune, 2014) confirm that NS is not a random search.

Finally, let's consider the quality measures given by both the objective functions and hits. The percentage of hits on the test set are the proportion of test instances where the agent reached the target; results are summarized in Figures 6.6 and 6.9. These results are very similar to those observed for the objective functions, basically the same trends are apparent. For both NS and objective-based search performance increases proportionally to the number of training instances. However, NS clearly outperforms all other methods, and only requires 6 training instances to reach an almost perfect performance on the test set. To clearly show the differences between all methods, a bar plot of these results is presented in Figure 6.10. This figure shows the percentage



(a) **1 instance** (b) **2 instances** (c) **6 instances** (d) **12 instances**

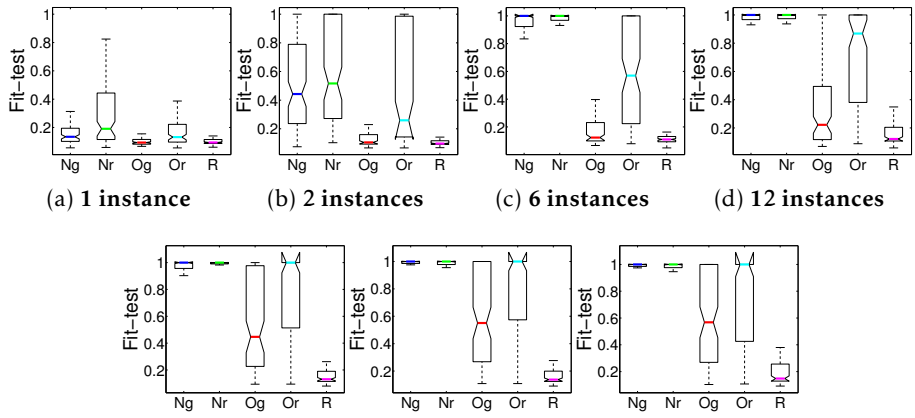
(e) **24 instances** (f) **48 instances** (g) **60 instances**

Figure 6.8: Box plot comparison of the best solution found using training fitness with function f_2 and descriptor d_β. 'N' stands for novelty-based search, 'O' for objective-based search, and 'R' stands for random-based search. Sub-index 'g' and 'r' stand for the method used to choose randomly the training set, either every generation or every run, respectively. Figures are sorted in ascending order according to the number of instances used in the training sets, from 1 to 60.

of hits on the test set relative to the number of training instances (x-axis), for both objective functions (columns) and for both manners in which the training set is selected (rows). These plots clearly suggest that NS achieves substantially better generalization performance than traditional objective-based search and random search. Indeed, NS can achieve almost perfect generalization for sufficiently large training sets, above 6 instances for the present task.

6.4.2 *Comparison with a Manually Selected Training Set*

In this section, the goal is to compare the random construction of training sets with manually chosen initial conditions. Here, we consider 12 different initial conditions for the agent, where it is easier to

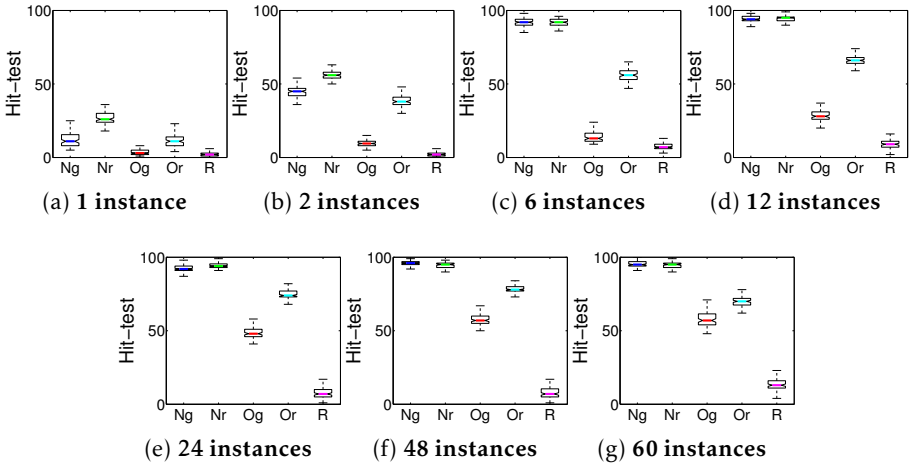


(a) **1 instance** (b) **2 instances** (c) **6 instances** (d) **12 instances**

(e) **24 instances** (f) **48 instances** (g) **60 instances**

Figure 6.9: Box plot comparison of the best solution found using test fitness with function f_2 and descriptor d_β. 'N' stands for novelty-based search, 'O' for objective-based search, and 'R' stands for random-based search. Sub-index 'g' and 'r' stand for the method used to choose randomly the training set, either every generation or every run, respectively. Figures are sorted in ascending order according to the number of instances used in the training sets, from 1 to 60.

reach the target from some positions than others. The chosen instances are depicted in Figure 6.2(a), specifying the location and orientation of each instance; these instances were used in our preliminary work (Urbano et al., 2014b). Table 6.4 summarizes the performance of the learning process when each of the single instances is used for training (each row), and the average over all (final row). It is evident that most of the chosen instances cannot lead the search towards a general solution for none of the search strategies. Moreover, if we consider the averages reported for the test set, they are worse than when compared with a single randomly chosen training case (first row of Table 6.4). It is important to note that these training instances were carefully chosen to provide different scenarios for the learning process, with the hope that they might allow the learning process to find general solutions



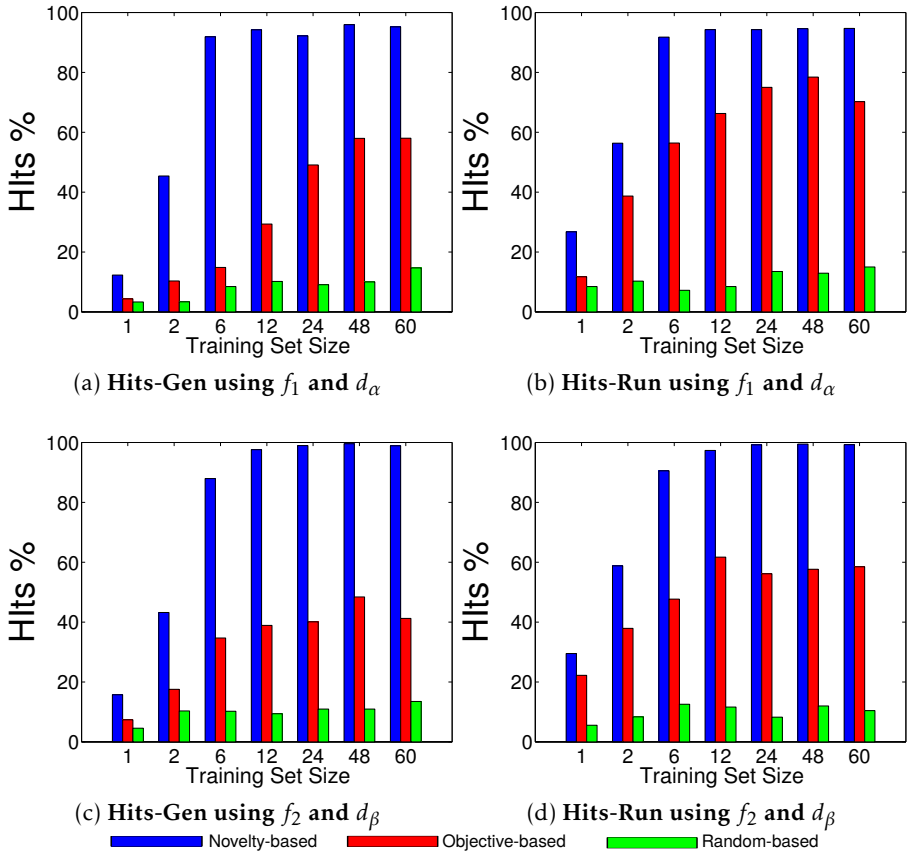
(a) **1 instance** (b) **2 instances** (c) **6 instances** (d) **12 instances**

(e) **24 instances** (f) **48 instances** (g) **60 instances**

Figure 6.10: Box plot comparison of the percentage of testing hits by the best solution found using function f_2 and descriptor d_β. 'N' stands for novelty-based search, 'O' for objective-based search, and 'R' stands for random-based search. Sub-index 'g' and 'r' stand for the method used to choose randomly the training set, either every generation or every run, respectively. Figures are sorted in ascending order according to the number of instances used in the training sets, from 1 to 60.

(Kushchu, 2002a,b). However, the bias introduced by the human expert did not help, and actually performs worse than randomly chosen instances.

It is also true, as the results from the previous subsection showed, that using a single training instance is too few. NS for instance reaches strong performance only when the training set has 6 or more instances. Therefore, we group all of the manually chosen instances into a single training set and compare it with the results of using 12 randomly chosen instances. These results are summarized in Table 6.5. First, it is clear that NS outperforms objective and random search, for all approaches towards training set construction. However, the performance of randomly chosen instances (per run or per generation) outperform



(a) **Hits-Gen using f_1 and d_α**

(b) **Hits-Run using f_1 and d_α**

(c) **Hits-Gen using f_2 and d_β**

(d) **Hits-Run using f_2 and d_β**

Figure 6.11: Comparison of the performance on the test set based on percentage of hits, considering both fitness functions (f_1 and f_2), both behavior descriptors (d_α and d_β) and both strategies to set the training set (every run (Run) or every generation (Gen)).

the Manual approach. Again, it seems that human bias does not improve search performance, and in fact generates a negative impact.

Table 6.4: Results from each of the twelve instances; I_1,...I_{12}, where the sub-index stands for the number of the training instance (in the heading as 'Size'). Three different approaches of fitness metrics are used to evolve solutions; novelty, objective and random. Two fitness function are used to score the best program from each run; f_1 considers last position, and f_2 and considers both last position and number of non-repeated visited cells. The average number of best programs that hit the target is shown in the column 'H' where the subindex indicates the function used (f_1 or f_2). Limit of moves for all experiments is 600. Bold indicates the best performance.

		Training		**Test**			
Size	**Fitness**	f_1	f_2	$H1$	$H2$	f_1	f_2
I_1	Novelty	**1.0000**	**1.0000**	**3**%	**3**%	0.2631	**0.8735**
	Objective	0.7445	0.9847	**3**%	0%	**0.2862**	0.8588
	Random	0.4403	0.9649	1%	0%	0.1496	0.8572
I_2	Novelty	**1.0000**	**1.0000**	**6**%	1%	**0.3652**	0.8889
	Objective	0.9524	0.9889	2%	**4**%	0.2017	**0.9098**
	Random	0.2801	0.9567	2%	2%	0.1594	0.8715
I_3	Novelty	**1.0000**	**1.0000**	**12**%	3%	**0.4768**	**0.9297**
	Objective	0.6977	0.9890	1%	**5**%	0.3238	0.9212
	Random	0.2019	0.9429	0%	0%	0.1455	0.8873
I_4	Novelty	**1.0000**	**1.0000**	**8**%	**5**%	0.4553	**0.9261**
	Objective	**1.0000**	0.9774	3%	0%	**0.6656**	0.9055
	Random	0.2273	0.9529	1%	2%	0.1651	0.8661
I_5	Novelty	**1.0000**	**1.0000**	**0**%	**1**%	**0.0913**	**0.7812**
	Objective	0.9802	0.9983	**0**%	0%	0.0720	0.7779
	Random	**1.0000**	0.9936	**0**%	0%	0.0839	0.6711
I_6	Novelty	**1.0000**	**1.0000**	**4**%	**0**%	**0.1682**	0.8468
	Objective	0.9527	0.9946	2%	**0**%	0.1073	0.8236
	Random	0.9301	**1.0000**	0%	**0**%	0.1068	**0.8502**
I_7	Novelty	**1.0000**	**1.0000**	**0**%	**0**%	0.1246	**0.5477**
	Objective	0.9917	**1.0000**	**0**%	**0**%	**0.1252**	0.5442
	Random	**1.0000**	**1.0000**	**0**%	**0**%	0.1203	0.5372
I_8	Novelty	**1.0000**	**1.0000**	**0**%	**0**%	**0.1215**	**0.5638**
	Objective	**1.0000**	**1.0000**	**0**%	**0**%	0.1207	0.5246
	Random	**1.0000**	**1.0000**	**0**%	**0**%	0.1154	0.5456
I_9	Novelty	**1.0000**	**1.0000**	**0**%	**0**%	0.0951	0.4644
	Objective	**1.0000**	**1.0000**	**0**%	**0**%	**0.0967**	0.4462

Continued on next page...

Table 6.4 – continued from previous page

		Training				Test	
Size	**Fitness**	f_1	f_2	$H1$	$H2$	f_1	f_2
	Random	**1.0000**	**1.0000**	**0**%	**0**%	0.0950	**0.4819**
I_{10}	Novelty	**1.0000**	**1.0000**	**0**%	**0**%	0.0816	0.4403
	Objective	**1.0000**	**1.0000**	**0**%	**0**%	0.0819	**0.4502**
	Random	**1.0000**	**1.0000**	**0**%	**0**%	**0.0842**	0.4329
I_{11}	Novelty	**1.0000**	**1.0000**	**11**%	**7**%	**0.5381**	**0.9505**
	Objective	0.6492	0.9775	6%	1%	0.3878	0.8555
	Random	0.1906	0.9570	2%	1%	0.1731	0.8000
I_{12}	Novelty	**1.0000**	**1.0000**	**13**%	**17**%	**0.6270**	**0.9585**
	Objective	0.6710	0.9821	2%	5%	0.4221	0.8763
	Random	0.1972	0.9518	2%	0%	0.1748	0.7315
Aver.	Novelty	**1.0000**	**1.0000**	**5**%	**3**%	0.2837	**0.7643**
	Objective	0.8832	0.9911	2%	1%	0.2401	0.7412
	Random	0.6195	0.9766	1%	0%	0.1306	0.7110

Table 6.5: Table that summarizes the performance of the three methods; novelty, objective, and random-based search for a set of 12 instances considering 600 as limit of moves, showing the results for the three different ways to select the training set; manually (Manual), randomly chosen each generation (Rand-Gen), and randomly chosen each run (Rand-Run). Bold indicates the best performance.

			Training		Test			
Size	**Sel**	**Fitness**	f_1	f_2	$H1$	$H2$	f_1	f_2
12	Manual	Novelty	**0.9576**	**0.9996**	**68**%	**81**%	**0.9302**	**0.9985**
		Objective	0.6775	0.9701	29%	30%	0.5310	0.9638
		Random	0.4100	0.9328	1%	4%	0.1713	0.9177
12	Rand-Gen	Novelty	**0.9730**	**1.0000**	**94**%	**98**%	**0.9485**	**0.9979**
		Objective	0.5131	0.9664	29%	39%	0.3609	0.9524
	Rand-Run	Novelty	**0.9731**	**0.9999**	**94**%	**97**%	**0.9501**	**0.9978**
		Objective	0.7520	0.9784	66%	62%	0.6964	0.9735
		Random	0.2524	0.9274	8%	12%	0.1673	0.9223

6.4.3 *Statistical Analysis*

Finally, to validate the results presented above we perform extensive statistical comparisons. For each experimental configuration, and for each performance measure, we perform $N \times N$ pairwise tests, where N is the number of different search methods (Ng, Nr, Og, Or and R). In other words, for each pair of algorithms (Ng-Nr, Ng-Og, and so on) we perform a statistical test based on each performance measure. We employ the Friedman non-parametric test to perform all pairwise comparisons of the tested algorithms as suggested in (Derrac et al., 2011), with Bonferroni-Dunn correction of the p-values. The null hypothesis that booth groups have equal medians was rejected at the $\alpha = 0.01$ significance level. For almost all comparisons the null hypothesis was rejected with p-values ≈ 0, but in some cases this was not the case. Table 6.6 summarizes the cases where the null hypothesis was not rejected for at least one performance measure, showing the corresponding p-values for each comparison based on each of the performance measures. However, even in these cases for some performance measures the null hypothesis was rejected, if so this is marked with an asterisk.

Table 6.6 shows that when the training set is small (1 or 2) there are more cases in which the null hypothesis cannot be rejected. As the boxplots suggested, all algorithms show a similar weak performance with such a small training set. However, as the training set size increases then we can see that in almost all cases the null hypothesis was rejected, indicating a clear statistical significance in the results discussed in the previous subsections, particularly when we compare the NS variants (Ng and Nr) with OS (Og and Or) and random search.

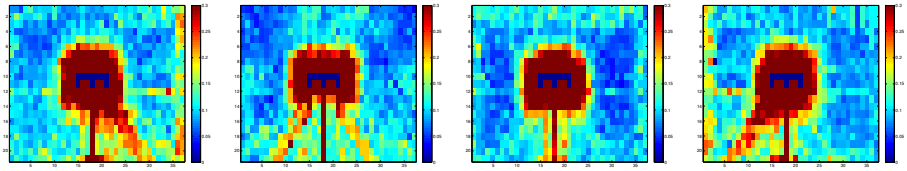
6.5 Heat Maps

In this section a new approach is proposed as a possible research line to extend this work. The main goal is to explore the impact of the training on to evolve generalization abilities, making a preprocessing of the entire set of initial conditions, getting a binary division into re-

Table 6.6: Summary of the statistical tests, showing the resulting p-values of each pairwise comparison using the Friedman test with Bonferroni-Dunn correction, for each experimental configuration and for each performance measure. Since in most cases the resulting p-values were ≈ 0, the table only presents cases in which the null hypothesis is not rejected at the $\alpha = 0.01$ significance level based on at least one performance measure, with an asterisk identifying the cases in which the null hypothesis is rejected.

		Training		Test			
Size	**Methods**	f_1	f_2	$H1$	$H2$	f_1	f_2
1	**Ng vs Nr**	4.000	4.000	0.000 *	0.000*	0.646	0.026
	Ng vs Og	0.101	0.629	0.000*	0.000*	0.000*	0.438
	Ng vs Or	0.000*	0.000*	3.654	0.000*	2.194	0.111
	Nr vs Og	0.101	0.629	0.000*	0.000*	0.000*	0.000*
	Nr vs Or	0.000*	0.000*	0.000*	0.000*	0.020*	0.646
	Og vs R	0.000*	0.000*	0.000*	0.000*	0.047*	0.920
	Or vs R	0.004*	0.000*	0.000*	0.000*	0.139	0.287
2	**Ng vs Nr**	1.269	4.000	0.000*	0.000*	0.006*	0.007*
	Ng vs Or	0.000*	0.000*	0.000*	0.000*	0.124	1.909
	Nr vs Or	0.000*	0.000*	0.000*	0.000*	0.038*	0.053
	Og vs Or	0.059	2.530	0.000*	0.000*	0.000*	0.224
	Og vs R	0.000*	0.000*	2.367	0.000*	2.194	0.010*
6	**Ng vs Nr**	1.269	0.629	3.332	0.000*	2.060	2.929
	Og vs Or	0.000*	3.012	0.000*	0.000*	0.000*	0.117
	Og vs R	0.000*	0.000*	0.000*	0.000*	0.920	0.026*
12	**Ng vs Nr**	4.000	1.269	3.324	1.484	2.764	3.545
	Og vs Or	0.000*	0.542	0.000*	0.000*	0.000*	0.068
24	**Ng vs Nr**	2.955	1.269	0.0000*	2.397	1.968	2.360
	Og vs Or	0.000*	0.311	0.000*	0.000*	0.000*	0.029*
48	**Ng vs Nr**	1.693	2.254	0.0000*	2.344	3.526	3.274
	Nr vs Or	0.106	0.000*	0.000*	0.000*	0.064	0.000*
	Og vs Or	0.135	0.059	0.000*	0.000*	0.021*	0.004*
60	**Ng vs Nr**	3.091	4.000	0.198	0.155	3.563	0.882
	Og vs Or	1.790	0.132	0.000*	0.000*	0.524	0.035*

gions of easy-difficult to generalize initial conditions. In order to get this division, 30 objective-based runs were executed from each possible initial condition in the maze navigation. The average overfit value



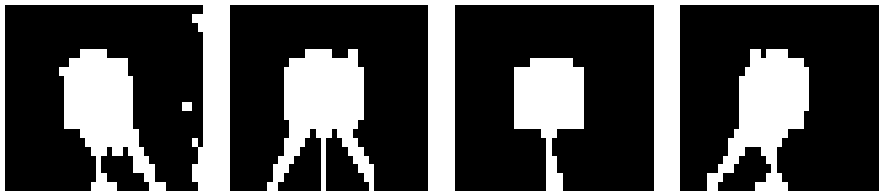
(a) **West** (b) **North** (c) **South** (d) **East**

Figure 6.12: Overfit heat-maps, at the right of each figure is shown a color scale that goes from low values of overfitting in color blue to higher values in color red.

is computed from the difference between the training and test quality using the 30 experimental runs. Using the experimental results we build-up the heat-maps for each orientation is showed in 6.11, where the average overfitting from the 30 runs are showed according with the color scale, low values are in color blue while higher values in red.

Trough a color threshold these heat-maps can be transformed into the binary-maps as shown in the Figure 6.12. Region in color white contains instances that produce higher values of overfitting, while the region in black contains the instances with the lower values of overfitting. The hypothesis is that overfitting is directly correlated with the ability to generalize. Then, instances in the white regions since show higher values of overfitting they tend to generate very specialized controllers which hardly can find the target from different instances specially from those located in the the black regions, and for this reason are considered *difficult* instances to generalize. This reasoning applies for the black region, considering their instances as *easy* to generalize.

With this approach we can test different combinations of easy-difficult instances, for instance if we consider a training set with a size of 6, we can get the following set of combinations easy-difficult: $\{0-6, 1-5, 2-4, 3-3, 4-2, 5-1, 6-0\}$, each initial condition is chosen randomly from the corresponding region. The Figure 6.13 (a) shows an example of a set of 12 initial conditions, particularly for a combina-



(a) **West** (b) **North** (c) **South** (d) **East**

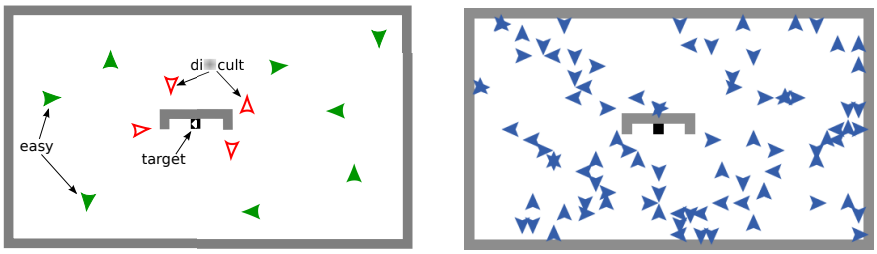
Figure 6.13: Overfit binary maps, obtained from the binarization of the Overfit heat-maps by a color threshold to divide the map into easy and difficult regions.

tion of 8-easy and 4-difficult initial conditions, and the Figure 6.13 (b) shows the same test set used in the previous experiments.

This experimental framework can give us insight about how this distinction between easy and difficult initial conditions can benefit to any of the methods tested, and furthermore to test if any of the possible combinations is the best suited to improve the generalization abilities of the navigator.

6.6 Chapter Conclusions

This work studies the problem of generalization in ER, using a navigation task for an autonomous agent in a 2D environment. The learning system is based on GE and three search strategies are evaluated: objective-based search, NS and a random search process. In particular, this chapter studies the impact that the training set has on the performance of the learning process and the generalization abilities of the evolved solutions. Several aspects are considered, namely: (1) the size of the training set; (2) the manner in which the training set is selected (manually or randomly); and (3) if the training set is fixed during the run or if it is varied every generation. Moreover, two objective functions are used and two behavior descriptors are tested to promote solution diversity with NS.



(a) **Easy-difficult training Set** (b) **Test Set**

Figure 6.14: Figure (a) shows the learning environment, where the target is depicted by a black square, and 12 randomly chosen starting positions, particularly with a combination (4-8) of 8 "easy" and 4 "difficult" instances. Each instance has a triplet (x, y, θ), where x, y is the 2-d location and θ is the orientation (North, East, South, and West). Figure (b) shows 100 initial conditions that were randomly generated to be used as a test set.

Experimental results clearly suggest that NS improves the generalization abilities of the GE system. NS can reach close to optimal performance during training in all scenarios, and reaches the same level of test performance with relatively small training sets (above 6 instances in our experiments). This is the main conclusion of our work, that NS can clearly guide the search towards general solutions within the studied environment. On the other hand, objective-based search cannot reach the same level of performance, both in training as well as testing, especially when considering the percentage of test instances where the target is reached (hits). It is clear that traditional search cannot solve the navigation problem for a diverse set of training instances, and fails to generalize to arbitrary testing conditions. Random search also exhibits very weak performance, confirming that NS is not equivalent to a random search process, even if it does not consider the objective function explicitly.

Varying the training set during evolution was expected to be beneficial towards generalization, but results do not suggest this. For NS,

keeping the training set fixed during the run produces almost the exact same results as randomly varying it every generation. On the other hand, objective-based search is hampered when a dynamic fitness function is used, both in terms of training and testing. Objective-based search achieves better performance when the training set size is increased and when it is kept fixed throughout the evolutionary process.

Finally, we compare a manual selection of training instances, the strategy used in our previous work (Urbano et al., 2014b), with the random strategy studied here. Results clearly indicate that manual selection introduces an undesirable bias that negatively effects generalization. In particular, testing performance is clearly better with a random training set instead of a manual a priori set, suggesting that removing human bias can be very beneficial.

The results presented here open up several possible lines of future research. While this work provides useful insights regarding the impact of the training set, only coarse features are considered, such as its size or the overall strategy used to build it (random or manual/fixed or dynamic). However, more detailed features of the training set can be controlled. For instance, it is evident that not all training instances are created equally, some of them specify more difficult scenarios than others, or some of them might lead towards deceptive landscapes while other might not. Therefore, future work will study the impact that the composition of the training set has on learning and generalization. Considering, for example, the proportion of easy and difficult training instances, or the proportion of deceptive and non-deceptive initial conditions. One final open question is to perform similar tests to study the issues related with the so-called reality-gap in ER.

7

GP based on NS for Regression

ABSTRACT — This chapter presents our approach towards applying NS to symbolic regression with GP. In order to do so, it is highlighted on this work, that the key component is the individual behavior representation for a given domain. NS can lead to complex behavioral traits, trying just to find novel solutions can guide the search from the common low quality solutions to the scarce and better solutions.

7.1 Introduction

The proposal of this chapter is to develop a NS-based GP system for solving real-valued symbolic regression problems, hereafter referred to as NS-GP-R. In the regression analysis a function $f(\mathbf{x})$ is approximated by an operator $K(\mathbf{x})$ considering the independent variable $\mathbf{x} \in \mathbb{D}^n \subseteq \mathbb{R}^n$, where $\mathbb{D}^n$ is the domain of $f(\mathbf{x})$. If we consider the training set of fitness cases as X, then the goal is to find an operator K that minimizes $\left\| f(\mathbf{x}) - K(\mathbf{x}) \right\|$. However, to use NS the performance of $K(\mathbf{x})$ must not be expressed as an error over all fitness cases, but a description of how K behaves over the entire set of fitness cases.

7.1.1 *Behavioral Descriptor*

Since the goal of a symbolic regression problem is straightforward, to minimize the error between the desired output and the proposed approximation, then a behavioral descriptor for symbolic regression must consider the concept of error in some way. The descriptor must provide a sufficiently detailed description of how an individual behaves over all fitness cases. Moreover, the descriptor should abstract away unnecessary details, and focus on describing how unique a particular individual is relative to others within the population.

ϵ-DESCRIPTOR: For a regression problem, let $f : \mathbb{R}^n \mapsto \mathbb{R}$ be the function to be approximated, $X = \{(\mathbf{x}_1, f(\mathbf{x}_1)), (\mathbf{x}_2, f(\mathbf{x}_2)), \ldots, (\mathbf{x}_L, f(\mathbf{x}_L))\}$ the set of fitness cases (input-output tuples), K_i a valid GP individual, and $\mathbf{e}_i = \{e_{i1}, e_{i2}, \ldots, e_{iL}\}$ be the error vector of K_i with $\mathbf{e}_{ij} = |f(\mathbf{x}_j) - K(\mathbf{x}_j)|$; then, the ϵ-descriptor β_i^ϵ of K_i is a binary vector of size L that is constructed as follows.

For simplicity and without loss of generality, consider $L = 1$. Then, at generation t with population P, sort P in ascending order based on e_i, and compute the order statistic p of the set of error vectors $E = \{e_i | \forall K_i \in P\}$, where $p = \frac{|P|}{h}$ with h as an algorithm parameter. Then, set a bounding value $\epsilon_t = min(\epsilon_{t-1}, \epsilon_p)$ if $t > 0$ and set $\epsilon_t = \epsilon_p$ otherwise; such that $\beta_i^\epsilon = 1$ for all K_i with $i \leq p$, and $\beta_i^\epsilon = 0$ otherwise.

7.1.2 *Pictorial Example*

For example, consider $h = 10$, then the ϵ-descriptor β_i^ϵ of K_i identifies the fitness cases j on which individual K_i is in the best 10-percentile of the population, when $\beta_{ij}^\epsilon = 1$. Conversely, if $\beta_{ij}^\epsilon = 0$, this means that the individual performs in the worst 90% of the population. A graphical description of this process is shown in Figures 7.1. More-

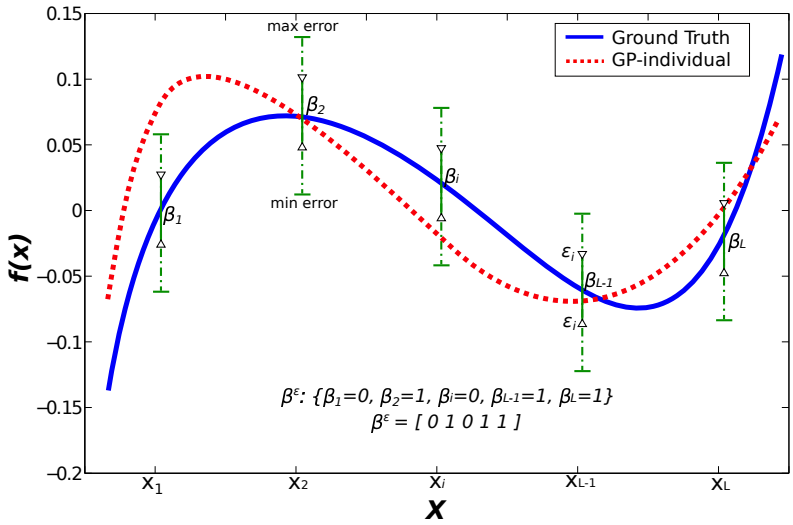


Figure 7.1: Graphical depiction of how the ϵ−descriptor is constructed. Notice how the interval specified by ϵ determines the values of each β_i.

over, a description of the algorithm used to compute the ϵ-descriptor is given in Algorithm 2.

7.1.3 *NS Modifications*

The behavioral descriptor presented above is constructed based on the performance of each individual relative to the rest of the population. Therefore, the descriptor assigned to any given individual will depend on the population to which it belongs. In other words, the context in which a descriptor is computed varies as a function of the current population. This represents a slight modification to the original NS implementation. However, it is consistent with the idea that the uniqueness of an individual depends on the search progress at the moment in which the individual was found.

Algorithm 2 Behavior descriptor β^{ϵ}

Require: $x, f(x), n, L, K, p, m, gen$
▷ $x \in \mathbb{R}^n$
▷ n : individuals, pop : population size
▷ L : sample size
▷ K : GP function
▷ p : heuristic parameter
▷ m : number of generations,
▷ gen : number of generations
Ensure: binary vector β^{ϵ}
for $t = 0 : gen$ **do**
for $i = 1 : pop$ **do**
for $j = 1 : L$ **do**
▷ $E_{ij} = \{e_{ij}\}$: error matrix
$e_{ij} \Leftarrow |f(\mathbf{x}_j) - K_i(\mathbf{x}_j)|$
end for
end for
▷ descending sort by column, $E = e_{ij}$
$E \Leftarrow$ sort(E,j,descend)
▷ order statistic p
$p \Leftarrow |\frac{P}{h}|$
$\epsilon_{t+1} \Leftarrow \epsilon_p$
if $\epsilon_{t+1} > \epsilon_t \& t \neq 0$ **then**
▷ epsilon vector update
$\epsilon_{t+1} \Leftarrow \epsilon_t$
end if
▷ individual descriptor loop
for $i = 1 : pop$ **do**
for $j = 1 : L$ **do**
if $e_{ij} \geq \epsilon_{(t+1),j}$ **then**
$\beta^{\epsilon}_{ij} \Leftarrow 1$
else
$\beta^{\epsilon}_{ij} \Leftarrow 0$
end if
end for
end for
end for

This design choice does raise a problem when sparseness is computed, since the descriptors of individuals stored in the archive will, with high probability, not represent a consistent behavioral descriptor with respect to the individuals in the current population. Therefore, the second modification made to the NS algorithm for regression problems is to omit the archive when sparseness is computed.

The lack of an archive can lead the canonical NS to perform backtracking during the search, cycling within behavioral space without any real progress. However, this problem is avoided by not allowing ϵ to increase in magnitude. Since, for most problems, the initial random population will mostly contain individuals with a large error vector, in the initial generations the ϵ bounds will tend to be high.

Then, with selective preference given to individuals with unique behaviors and given the way in which the ϵ-descriptor is constructed, the algorithm will be biased towards individuals with descriptors that contain a large proportion of ones since most low performance solutions will have behavioral descriptors with a large proportion of zeros; i.e., it will be biased towards individuals that achieve a low error on many fitness cases. Therefore, since ϵ cannot increase over generations, then selective pressure will push the search towards novel individuals that exhibit a low error on many fitness cases. Nonetheless, the archive is still used to save promising individuals across generations, and is used to select the best solution at the end of the run, but not used to compute novelty or guide the search.

NS helped by the ϵ mechanism will push the search towards novel solutions that finally find the global optima β^1. The performance of the behavior proposed is better explained in the Algorithm 2.

7.2 Experiments

The proposed NS-GP-R algorithm is evaluated on three benchmark problems, suggested by (McDermott et al., 2012) and proposed in (Uy et al., 2011a). Moreover, for comparative purposes, the algorithm is compared to the results published in (Uy et al., 2011a), that use a standard GP, hereafter referred to as SGP, and a GP with the Semantic Similarity-based Crossover (SSC) also proposed in (Uy et al., 2011a).

Table 7.1: Benchmark symbolic regression problems taken from (Uy et al., 2011a).

	Problems	Fitness/Test cases
f_1	$x^4 + x^3 + x^2 + x$	20 random points $\subseteq [-1,1]$
f_2	$sin(x^2) * cos(x) - 1$	20 random points $\subseteq [-1,1]$
f_3	$2sin(x) * cos(y)$	100 random points $\subseteq [-1,1]x[-1,1]$

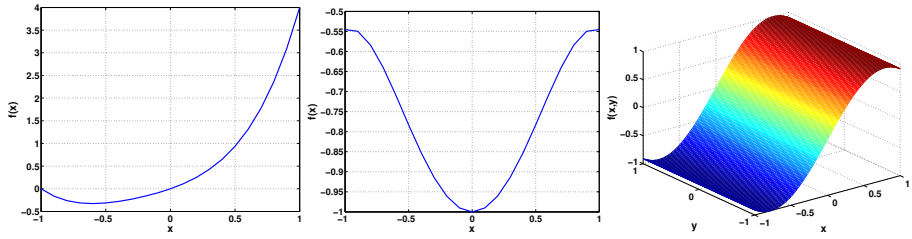
7.2.1 *Test Problems and Parametrization*

The three symbolic regression problems are given in Table 7.1. The problems were chosen based on the difficulty the problems posed to the methods published in (Uy et al., 2011a), and they are ordered from the easiest to the most difficult. It is not claimed that this ordering implies any deeper understanding of the intrinsic difficulty of the problem, it is only based on the performance of the algorithms compared in (Uy et al., 2011a).

The two easiest problems have one independent variable, while the hardest problem has two independent variables; Figure 7.2 shows the ground truth function in each problem. Table 7.1 specifies the desired function and the manner in which the training set (fitness cases) and testing set are constructed, using the same random procedure in both.

7.2.2 *Parameter Settings*

The novelty based Regression Funciton Descriptor (ED) is tested, with one configuration that uses an sparseness threshold of $\rho_{min} = 3$ for single variable problems and $\rho_{min} = 13$ for bi-variable problems. The number of neighbors is set to $k = 15$. The proposed algorithms is a tree-based Koza style, as well a crossover and a mutation subtree-based. Table 7.2 summarizes the parameters for this algorithm. Finally, the algorithm were coded using Matlab 2012b and the GPLAB toolbox (Silva and Almeida, 2003).



(a) $f_1(x)$ (b) $f_2(x)$ (c) $f_3(x,y)$

Figure 7.2: Benchmark regression problems showing the ground truth function taken from (Uy et al., 2011a)

Table 7.2: Parameters for the experiments.

Parameter	**Description**
Population size	200 individuals.
Generations	100 generations.
Initialization	*Ramped Half-and-Half*,with 6 levels of maximum depth.
Operator probabilities	Crossover $p_c = 0.8$, Mutation $p_\mu = 0.2$.
Function set	$(+, -, \times, \div, \lvert\cdot\rvert, x^2, \sqrt{x}, log, sin, cos, if)$.
Terminal set	$\{x_1, ..., x_i, ..., x_p\}$, where x_i is a dimension of the data patterns $\mathbf{x} \in \mathbb{R}^n$.
Bloat control	Dynamic depth control.
Initial dynamic depth	6 levels.
Hard maximum depth	20 levels.
Selection	Tournament of size 6.

7.2.3 *Experimental Results*

After executing the algorithm 30 times on each problem, the following results are obtained. First, Figure 7.3 presents a boxplot analysis

Table 7.3: Parameters for novelty search.

Parameter	Value
NS nearest neighbor	$k = 15$
Sparseness threshold:	
for single variable problems	$\rho_{min} = 3$
for bivariable problem	$\rho_{min} = 13$
ϵ-descriptor threshold	$p = 10$
Number of runs per problem	$runs = 30$

of the performance of NS-GP-R on each benchmark problem, showing the best error on the test-set achieved in each run. For comparative purposes Table III presents a comparison of the average error of NS-GP-R on each benchmark, along with the average error reported in (Uy et al., 2011b) for SGP and SSC. The table shows an interesting trend, NS performs worse on the easiest problem, performs about equally on the intermediate problem, and outperforms all methods on the most difficult problem. Results are compared based only on average error, no statistical tests are performed because only the published results are being used for comparison. The trend is apparent, NS works best when the problems are difficult, because in this scenario, novelty indeed leads towards quality, given that the initial random populations mostly contain individuals that perform poorly. Therefore, in difficult problems the selection pressure in novelty search can lead towards individuals that achieve good performance. In other words, when the gradient for novelty is positively correlated with the gradient for fitness, then NS can indeed find high performance solutions.

7.3 Chapter Conclusions

This chapter presented a NS-based GP algorithm for the classical GP problem of symbolic regression. In this respect, this chapter represents the first attempt to apply NS in this domain, one of the most com-

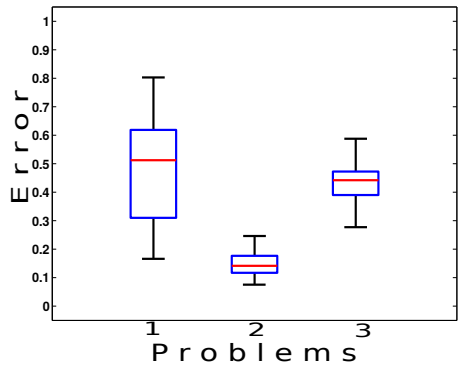


Figure 7.3: Boxplot analysis of NS-GP-R performance on each benchmark problem, showing the best test-set error in each run.

Table 7.4: Comparison of NS-GP-R with two control methods reported in (Uy et al., 2011a): SSC and SGP. Values are the mean error computed over all runs.

Problem	**NS-GP-R**	**SSC**	**SGP**
1	0.47	0.15	0.26
2	0.14	0.10	0.21
3	0.42	1.53	2.26

mon engineering and scientific problems. To achieve this, however, the concept of behavioral space needed to be adapted for it to be applied in this domain. Therefore, a behavioral descriptor was proposed called ϵ-descriptor, that emphasizes the main characteristics of program behavior while abstracting away unnecessary details, in order to properly describe the behavior that each GP individual exhibits with respect to the rest of the population.

Results are encouraging and consistent with recent proposals that expand the use of the NS algorithm to other mainstream areas. The proposed NS-based GP was compared with recently published results on three benchmark problems that are currently suggested for GP eval-

uation within the community (Zhang and Smart, 2006). NS shows a consistent trend, it achieved quite bad performance on easy problems, and performs substantially better on difficult ones, results that are similar to those published in (Naredo et al., 2013b; Naredo and Trujillo, 2013).

For real-world scenarios these results are promising, since most interesting problems are difficult. The reason for these results can be inferred from the nature of the NS algorithm. Random solutions in the initial population mostly exhibit bad performance, thus the selective pressure towards good solutions increases during the search for novelty. On the other hand, for easier problems the exploration performed by NS is mostly counterproductive, since random solutions might provide good initial approximations and all that is lacking is an exploitative search. Therefore, when random solutions have a high fitness then novelty could easily lead the search towards worse results. Conversely, when the problem is difficult and random solutions are of low fitness, then the search for novelty will lead towards high quality solutions.

Future work will center on exploring further experimental tests in this domain, comprehensively evaluating different parametrizations and performance achieved on other benchmarks and real-world problems, comparing the algorithm with other GP systems for symbolic regression. Moreover, it is imperative to provide a deep comparison between the proposed behavior-based search strategy and recent semantics-based approaches, a comparison that goes beyond merely experimental results, one that makes a detailed analysis of the main algorithmic differences between both approaches and their effects on search.

8

GP based on NS for Clustering

abstract — This chapter applies NS to unsupervised machine learning, namely the task of clustering unlabeled data. To this end, a behavioral descriptor is proposed describing a clustering function performance. Experimental results show that NS-based search can be used to derive effective clustering functions. In particular, NS is best suited to solve difficult problems, where exploration needs to be encouraged and maintained.

8.1 Introduction

Data analysis plays an indispensable role for understanding various phenomena. Cluster analysis, primitive exploration with little or no prior knowledge, consists of research developed across a wide variety of communities. One of the vital means in dealing with information data is to classify or group them into a set of categories or clusters.

In order to learn a new object or understand a new phenomenon, people always try to seek the features that can described it, and further compare it with other known objects or phenomena, based on the similarity or dissimilarity, generalized as proximity, according to some certain standard or rules. Basically, classification systems are either supervised or unsupervised, depending on whether they assign new

inputs to one of a finite number of discrete supervised classes or unsupervised categories, respectively.

In unsupervised classification, called clustering or exploratory data analysis, no labelled data are available. The goal of clustering is to separate a finite unlabelled data set into a finite discrete set of "natural", hidden data structures, rather than provide an accurate characterization of unobserved samples generated from the same probability distribution.

Most researchers describe a cluster by considering the internal homogeneity and the external separation, i.e., patterns in the same cluster should be similar to each other, while patterns in different clusters should not.

Given a set of patterns $\mathbf{X} = \{\mathbf{x}_1,...,\mathbf{x}_j,..,\mathbf{x}_N\}$, where $\mathbf{x}_j = (\mathbf{x}_{j1},\mathbf{x}_{j2},...,\mathbf{x}_{jd})^T \in \mathcal{R}^d$ and each measure $\mathbf{x}_{ji}$ is said to be a feature (attribute, dimension, or variable). As an important tool for data exploration, cluster analysis examines unlabelled data, by either constructing a hierarchical structure, or forming a set of groups according to a pre-specified number.

Research in Evolutionary Computation (EC) has produced search and optimization algorithms that frequently achieve promising new results in diverse domains (Koza, 2010). Therefore, the practical value of evolutionary algorithms (EAs) is by now widely accepted. Nonetheless, for some within the field a conceptual, or even philosophical, problem remains regarding most EAs. At their core, EAs are simple abstractions of Neo-Darwinian evolution. However, instead of the open-ended nature of biological evolution, EAs are objective driven, just like any conventional optimization algorithm. Therefore, EAs are expected to converge on a small subset of local optima within a static fitness landscape.

The chapter is organized as follows. Section 8.2 presents the proposed NS-based GP algorithm for data clustering and a behavioral descriptor for evolved clustering functions. Then, Section 8.3 presents the experiments and an analysis of the results. Finally, a summary of the chapter and concluding remarks are given in Section 8.4.

8.2 Clustering with Novelty Search

This section presents first two well known data clustering algorithms; K-means and Fuzzy C-means. Afterward, the proposed behavioral descriptor for GP-based clustering functions is presented, and we provide a discussion regarding its fitness landscape.

8.2.1 *K-means*

This is by far among clustering techniques that are based on minimizing an objective function, the most widely used and studied algorithm (Jain, 2010). K-means is a clustering method that intends to partition n observations into k clusters, where each observation belongs to the cluster with the nearest mean, using the cluster centres to model the data.

This algorithm by mean of an iterative partitioning minimizes the sum, over all clusters, of the within-cluster sums of point-to-cluster-centroid distances, using the squared Euclidean distance.

Given a set of observations $(\mathbf{x}_1, \mathbf{x}_2), ..., \mathbf{x}_n$, where each observation is a d-dimensional real vector, K-means clustering intends to partitioning the n observations into k sets $(k \leqslant n)$, $\mathbf{S} = S_1, S_2, ..., S_K$, so as to minimize the within-cluster sum of squares, according with

$$\arg\min_{S} \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in S_i} \|\mathbf{x}_j - \mu_i\|^2 \tag{8.30}$$

where μ_i is the mean of point in S_i.

8.2.2 *Fuzzy C-means*

Fuzzy C-means (FCM) is a method of clustering which allows a data element to belong to more than one cluster, the assignation to a cluster is relaxed, and the data element can belong to all of the clusters with a certain degree of membership (Bezdek et al., 1984). FCM attempts

to find a partition for a set of data points $\mathbf{x_j} \in \mathcal{R}^d, j = 1,..,N$ while minimizing the following objective function:

$$J(\mathbf{U},\mathbf{M}) = \sum_{i=1}^{c}\sum_{j=1}^{N}(u_{i,j})^m D_{ij} \tag{8.31}$$

where, $\mathbf{U} = [u_{i,j}]_{cxN}$ is the fuzzy partition matrix and $u_{i,j} \in [0,1]$ is the membership coefficient of the jth object in the ith cluster; $\mathbf{M} = [\mathbf{m}_1,...,\mathbf{m}_c]$ is the cluster prototype (mean or center) matrix; $m \in [1,\infty)$ is the fuzzification parameter and usually is set to 2; $D_{ij} = D(\mathbf{x}_j,\mathbf{m}_i)$ is the distance measure between $\mathbf{x}_j$ and $\mathbf{m}_i$.

8.2.3 *Cluster Descriptor (CD)*

The training set $\mathcal{T}$ used contains sample patterns from each cluster. Then, for a two-cluster problem with $\Omega = \{\omega_1,\omega_2\}$ the clustering descriptor (CD) is constructed in the following way. If $\mathcal{T} = \{\mathbf{y}_1,\mathbf{y}_2,...\mathbf{y}_L\}$, then the behavioural descriptor for each GP clustering function K_i is a binary vector $\mathbf{a_i} = (a_1,a_2,...a_L)$ of size L, where each vector element a_j is set to 1 if clustering function K_i assigns label ω_1 to pattern $\mathbf{y}_j$ and is set to 0 otherwise.

Suppose that the number of training examples from each cluster is $\frac{L}{2}$, and suppose that they are ordered in such a way that the first $\frac{L}{2}$ elements in $\mathcal{T}$ correspond to cluster label ω_1. Let $\mathbf{x}$ represent a binary vector, and function $u(\mathbf{x})$ return the number of 1s in $\mathbf{x}$. Moreover, let K_O be the *optimal* clustering function that achieves a perfect accuracy on the training set.

Then, the CD of K_Os behaviour is given by $\mathbf{a}^1 = (1_1,1_2,..., 1_{\frac{L}{2}},0_{\frac{L}{2}+1},...,0_L)$. Moreover, for a two-cluster problem, an equally useful solution is to take the opposite (complement) behaviours and invert the clustering, such that a 1 is converted to a 0 and vice-versa. These mirror behaviours are $\mathbf{a}^0 = (0_1,0_2,...0_{\frac{L}{2}},1_{\frac{L}{2}+1}....1_L)$ for the CD descriptors. The complete fitness landscapes in behavioural space are depicted in Figure 8.1.

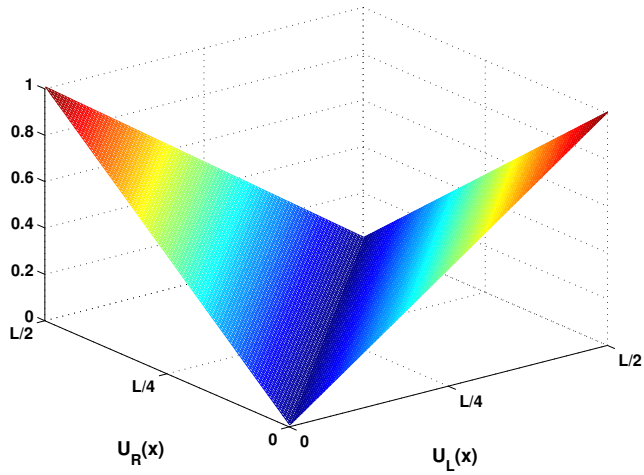


Figure 8.1: Fitness landscape in behavioral space for the CD descriptor.

For a two-cluster problem with a reasonable degree of difficulty, the initial generations of a GP search should be expected to contain close to random clustering functions, with roughly a 50% accuracy. For the CD descriptor, behavioral space is organized on a two dimensional surface, such that one axis u_L considers the number of ones on the left hand side, first $\frac{L}{2}$ bits, of a behavior descriptor $\mathbf{a}$, and u_R considers the remaining $\frac{L}{2}$ bits; see Figure 8.1.

Notice that the middle valley of the fitness landscape corresponds to basically random clustering functions, with worst case scenario performance. Hence, NS will push the search towards either of the two global optima, $\mathbf{a}^1$ and $\mathbf{a}^0$. On the other hand, for the AD descriptor, early behaviours will almost exclusively exhibit descriptors with equal proportions of zeros and ones; see Figure 8.1. Then, NS will progressively explore towards two opposite points in behavioral space, $\mathbf{b}^1$ or $\mathbf{b}^0$. The effect of these differences between the CD is explored experimentally in the following section. Finally, given the above binary descriptor, we use the Hamming distance to compute the novelty measure.

8.2.4 *Cluster Distance Ratio (CDR)*

This measure compares the dispersion within the clusters, considering both the intra and inter-cluster, we use the distances of the points from their cluster centre to determine whether the clusters are compact. First the intra-cluster measure is computed, using the Euclidean distance from each point to its nearest neighbor within its own cluster, then calculate the average nearest neighbor distance for all points, defined as

$$IntraCluster = \frac{1}{N} \sum_{i=1}^{K} \sum_{x \in C_i} \|x - v_i\|^2, \tag{8.32}$$

where N is the number of data elements, K is the number of clusters, and v_i is the cluster centre of cluster C_i. The intra-cluster give us a clustering compaction measure; the less intra-cluster distance the more compact the cluster is.

The inter-cluster distance or the distance between clusters, is computed using the Euclidean distance between each cluster centre to the rest of the cluster centres taking the minimum of these values, defined as

$$InterCluster = min(\|v_i - v_j\|^2), \tag{8.33}$$

where $i = 1, 2, ..., K-1$, and $j = i+1, ..., K$. Notice that it is considered more than one cluster, and since there must exist at least one data point within its corresponding cluster different from the other cluster, it is assured that the inter cluster to be greater than zero.

The ratio between the intra and inter-cluster determines the quality about the hypothesis used to group the data, defined as

$$CDR = \frac{IntraCluster}{InterCluster} \tag{8.34}$$

8.3 Experiments

The performance of the NS-based GP clustering function is examined against two well known clustering methods; K-means and Fuzzy C-means. The novelty based Class Descriptoy (CD) is tested, with two different versions, one configuration uses an sparseness threshold of S=20 and the other one S=40.

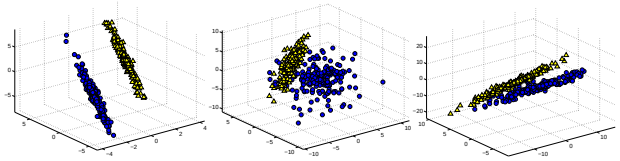
8.3.1 *Test Problems*

Gaussian Mixture Models are used to generate five random synthetic problems, each with three dimensions and different amounts of class overlap and geometry. All problems are set in the $\mathbb{R}^2$ plane with $(x, y) \in [-10, 10]^2$, and 200 sample points were randomly generated for each class. The parameters for the GMM of each class were also randomly chosen, following the same strategy reported in (Trujillo et al., 2011a). The five problems are of increasing difficulty, these problems are graphically depicted in Figure 8.2 and denoted with the label Ground Truth (GT).

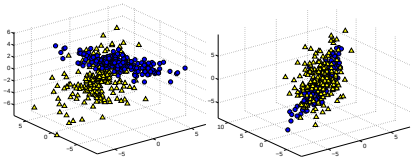
8.3.2 *Configuration and Parameter Settings*

As stated above, four different algorithms are experimentally compared: K-means, Fuzzy C-menas, CD-S20 and CD-S40. However, all algorithms share the same GP representation and genetic operators, a tree-based Koza style algorithm with subtree mutation and crossover. The parameters shared by all algorithms are summarized in Table 8.1.

For the NS-based algorithms two different parameter settings are used. In particular, two different values for the archive threshold ρ_{min} are used, 20 and 40. Parameter k on the other hand is set to 15 for all algorithms. Finally, all algorithms were coded using Matlab and the GPLAB toolbox (Silva and Almeida, 2003).



(a) *Problem-1* (b) *Problem-2* (c) *Problem-3*



(d) *Problem-4* (e) *Problem-5*

Figure 8.2: Five synthetic clustering problems; the observed clusters represent the ground truth data.

Table 8.1: Parameters for the GP-based search.

Parameter	**Description**
Population size	200 individuals.
Generations	100 generations.
Initialization	*Ramped Half-and-Half*,with 6 levels of maximum depth.
Operator probabilities	Crossover $p_c = 0.8$, Mutation $p_\mu = 0.2$.
Function set	$(+, -, \times, \div, \lvert\cdot\rvert, x^2, \sqrt{x}, log, sin, cos, if)$.
Terminal set	$\{x_1, ..., x_i, ..., x_p\}$, where x_i is a dimension of the data patterns $\mathbf{x} \in \mathbb{R}^n$.
Bloat control	Dynamic depth control.
Initial dynamic depth	6 levels.
Hard maximum depth	20 levels.
Selection	Tournament.

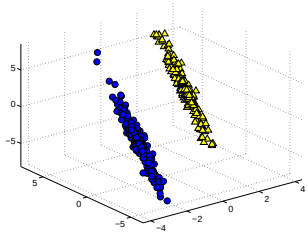
8.3.3 *Results*

This section presents an experimental evaluation of NS-GP clustering. All of the clustering algorithms were executed 30 times to find statistically significant results. Table 8.2 compare the performance of

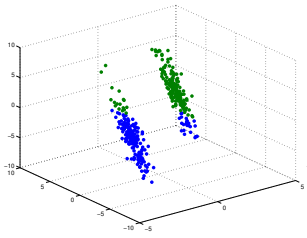
Table 8.2: Average classification error and standard error of the best solution found by each algorithm on each problem; GT: Ground Truth, KM: K-means, FC: Fuzzy C-means, NS-based algorithms both use $k = 15$, but CD1 with $\rho_{min} = 20$ and CD2 use $\rho_{min} = 40$.

	Class Distance Ratio (CDR)				
	GT	**KM**	**FC**	**CD1**	**CD2**
Problem-1	9.1720	10.2193	10.2511	0.006	0.006
Problem-2	4.4629	5.5085	5.4570	0.006	0.006
Problem-3	5.5502	5.4053	5.2887	0.006	0.006
Problem-4	4.1075	6.2132	6.1094	0.006	0.006
Problem-5	1.5977	4.5417	4.5184	0.006	0.006
	Clustering Error				
	GT	**KM**	**FC**	**CD1**	**CD2**
Problem-1	9.1720	0.1025	0.1000	0.0500 ± 0.005	0.0050 ± 0.005
Problem-2	4.4629	0.0750	0.0725	0.0925 ± 0.005	0.1300 ± 0.005
Problem-3	5.5502	0.2175	0.2625	0.2675 ± 0.005	0.4375 ± 0.005
Problem-4	4.1075	0.3600	0.3550	0.1875 ± 0.005	0.1725 ± 0.005
Problem-5	1.5977	0.4550	0.4500	0.4688 ± 0.005	0.4300 ± 0.005

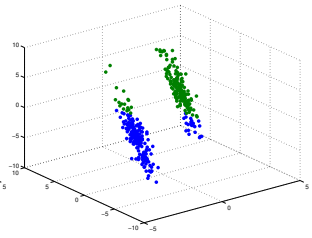
NS-GP with two baseline methods, KM and FC. The table presents two comparative views of average performance over all runs. First, the algorithms are compared based on their CDR score, and the CDR of the ground truth of each problem is also presented. Additionally, using the ground truth, a classification error was computed, based on the ordering suggested by each clustering method. In general, the results indicate two noteworthy trends. First, NS seems to performs much worse on the simpler problems, it seems like it is basically doing a random search. On the other hand, NS noticeably outperforms the control methods on the harder problems, this is especially true for the hardest, Problem 5. Second, it seems that a lower ρ_{min} encourages better performance in most cases. A detailed view of how the data is being clustered can provide a different analysis of the results. Figures 8.3 - 8.7 present a graphical illustration of the clustering output achieved by each method. All figures show the ground truth clusters for visual comparison, along with a typical clustering output from each method.



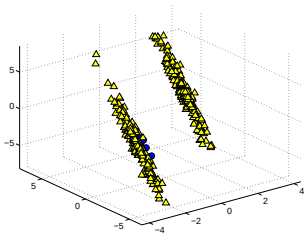
(a) *Ground Truth*



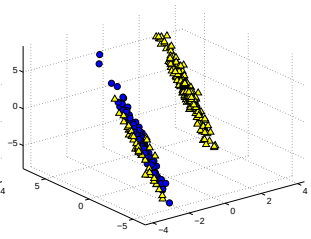
(b) *K-means*



(c) *FC-means*



(d) *NS-GP-20*

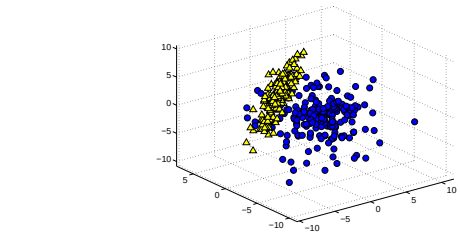


(e) *NS-GP-40*

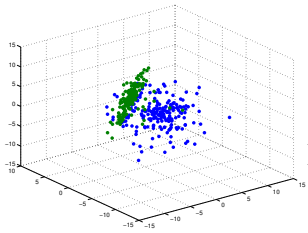
Figure 8.3: Comparison of clustering performance on Problem No.1.

These figures confirm the data presented in Table 8.2, NS-GP performs worse on the easy problems and performs better on the difficult ones.

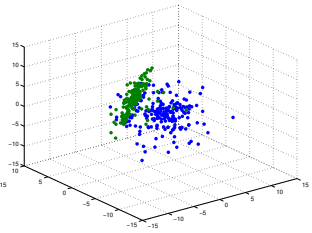
Figures 8.8 and 8.9 examine how sparseness evolves during the NS-GP search, for NS-GP-20 and NS-GP-40 respectively. Each figure presents a similar plot that shows how the sparseness of the best individual (based on fitness) evolves over each generation. The plots are averages of the 30 runs of each experiment and present a curve for each problem. A horizontal line shows the corresponding threshold value



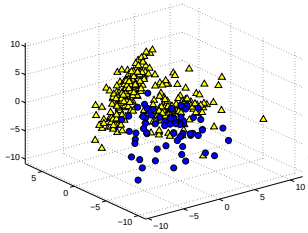
(a) *Ground Truth*



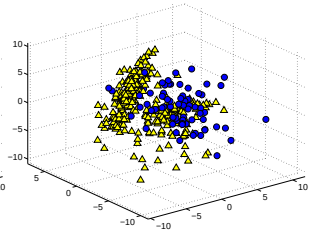
(b) *K-means*



(c) *FC-means*



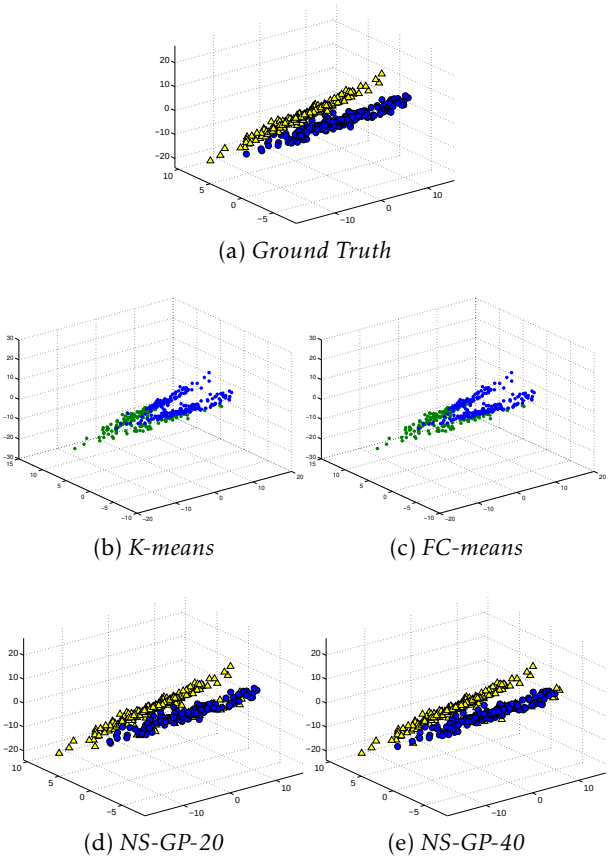
(d) *NS-GP-20*



(e) *NS-GP-40*

Figure 8.4: Comparison of clustering performance on Problem No.2.

for each configuration, set to 20 in Figure 8.8 and set to 40 in Figure 8.9. In both cases it is possible to observe a similar pattern. For the easy problems (1, 2 and 3) the sparseness value of the best individual at each generation does not reach the threshold, and is therefore not included in the population archive. This means that the individual is lost, and maybe it is never found again. This explains the reason why NS fails to produce good results on the easy problems. It appears that since the problems are not difficult, then novelty does not necessarily



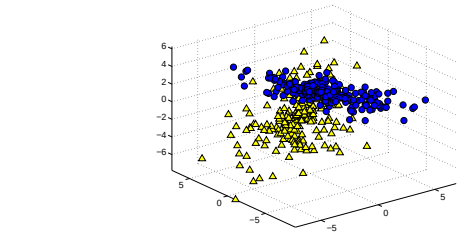
(a) *Ground Truth*

(b) *K-means* (c) *FC-means*

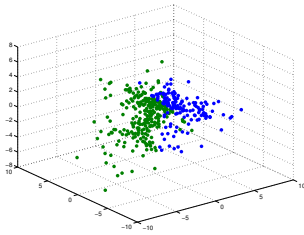
(d) *NS-GP-20* (e) *NS-GP-40*

Figure 8.5: Comparison of clustering performance on Problem No.3.

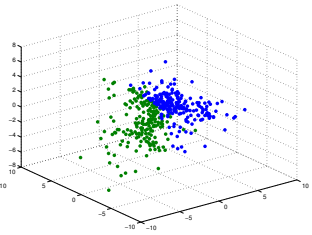
lead to quality, and NS is not influencing the search as desired. On the other hand, for the more difficult problems (4 and 5) it is clear that good solutions almost always correspond with novel solutions; i.e., the solutions that are being introduced into the archive of novel solutions also exhibit good fitness values. In these cases, the search for novelty is indeed leading evolution towards better overall solutions. This is reasonable, since for difficult problems the initial (random) programs will mostly exhibit bad performance, and novelty can lead towards quality.



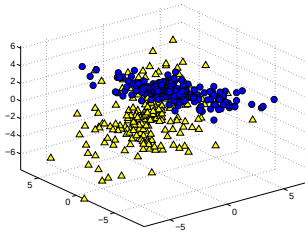
(a) *Ground Truth*



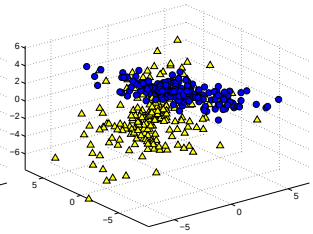
(b) *K-means*



(c) *FC-means*



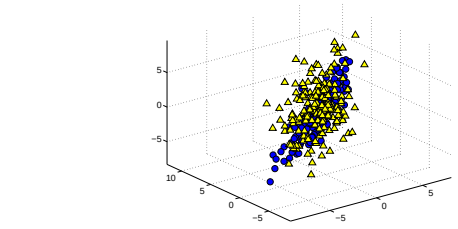
(d) *NS-GP-20*



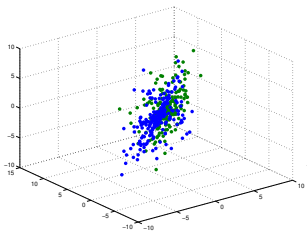
(e) *NS-GP-40*

Figure 8.6: Comparison of clustering performance on Problem No.4.

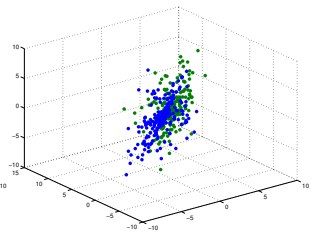
Finally, to illustrate how evolution progresses with the NS-GP clustering algorithm, Figure 8.10 presents four snapshots of NS-GP-40 applied to Problem 5. Each plot shown in the figure represents the best NS solution (taken from the current population and population archive) based on the CDR value at four different generations. For this difficult problem, it is clear that NS is progressively exploring the search space, and finding better solutions.



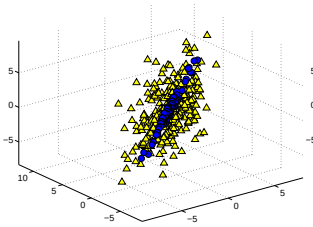
(a) *Ground Truth*



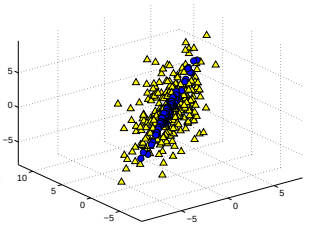
(b) *K-means*



(c) *FC-means*



(d) *NS-GP-20*



(e) *NS-GP-40*

Figure 8.7: Comparison of clustering performance on Problem No.5.

8.4 Chapter Conclusions

This chapter presented a NS-based GP system to search for data clustering functions. To do so, a domain-specific behavioral descriptor was proposed; the Cluster Descriptor (CD). It appears that NS-based search exhibits the best results when confronted with difficult problems. Since generating a high-quality solution at random is less probable for difficult problems, the incentive for behavioural exploration is incremented and the search for novelty indeed leads towards qual-

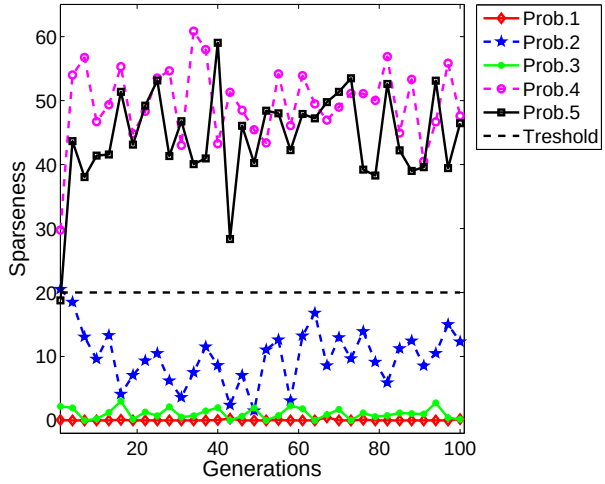


Figure 8.8: Evolution of sparseness for NS-GP-20, showing the average sparseness of he best individual at each generation.

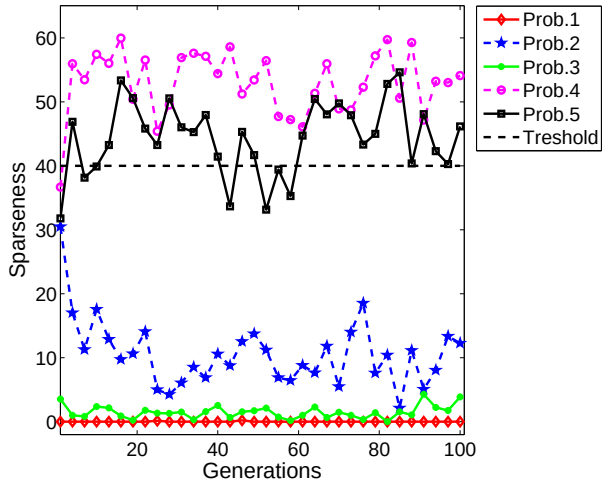
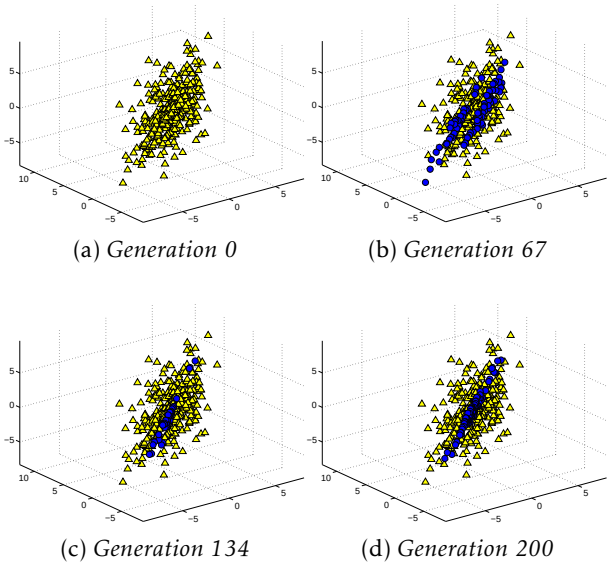


Figure 8.9: Evolution of sparseness for NS-GP-40, showing the average sparseness of he best individual at each generation.



(a) *Generation 0* (b) *Generation 67*

(c) *Generation 134* (d) *Generation 200*

Figure 8.10: Evolution of the best solutions found at progressive generations for Problem 5 with NS-GP-40.

ity during the search. On the other hand, for simple problems the exploratory capacity of NS is mostly unexploited. In particular, CD descriptor is less restrictive and for this reason it can be used on clustering. According with the results, CD descriptor probed that can be used with NS to solve clustering problems. Therefore, future work will focus on exploring the usefulness of NS on the more difficult problem of non-supervised learning.

9

GP based on NS for Classification

ABSTRACT — This chapter presents a NS-based genetic programming (GP) algorithm for supervised classification. Results show that NS can solve real-world classification tasks, the algorithm is validated on real-world benchmarks for binary and multiclass problems. These results are made possible by using a domain-specific behavior descriptor. Moreover, two new versions of the NS algorithm are proposed, Probabilistic NS (PNS) and a variant of Minimal Criteria NS (MCNS). The former models the behavior of each solution as a random vector and eliminates all of the original NS parameters while reducing the computational overhead of the NS algorithm. The latter uses a standard objective function to constrain and bias the search towards high performance solutions. The chapter also discusses the effects of NS on GP search dynamics and code growth. Results show that NS can be used as a realistic alternative for supervised classification, and specifically for binary problems the NS algorithm exhibits an implicit bloat control ability.

9.1 Introduction

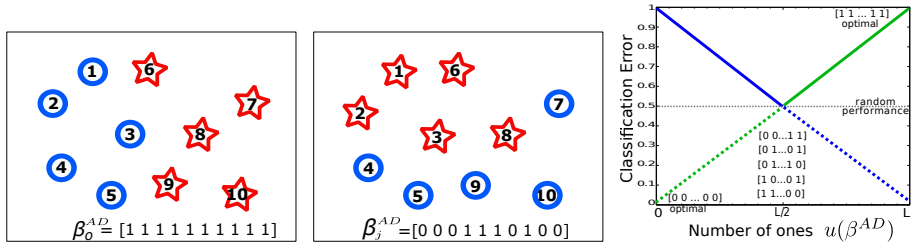
In this chapter we will continue our work on applying NS to common machine learning tasks, and will now focus on supervised clas-

sification. In particular, we will consider the new NS variants we proposed in Chapter 4, namely PNS and MCNS, as well as a hybrid method that combines both we call MCPNS. To recap, in PNS the novelty of a solution is estimated probabilistically by modeling each behavior as a random vector, a new way to compute novelty, which in addition eliminates all of the underlying NS parameters as well as reduces the computational overhead from the original NS algorithm. Moreover, MCNS proposes a blending of the objective function with the sparseness measure, where the minimal solution quality represents a dynamic criterion that is proportional to the quality of the best solution found so far.

To apply NS successfully, a behavior descriptor must first be proposed (Kistemaker and Whiteson, 2011). For instance, in a maze navigation problem Lehman and Stanley used the final robot position as the behavior descriptor (Lehman and Stanley, 2008, 2010b,c, 2011a). In this work, our main goal is to apply NS in GP-based classification. In particular, we use two GP classifiers: a simple binary classifier based based on a static threshold (Zhang and Smart, 2006) and a recently proposed multiclass approach (Ingalalli et al., 2014; Muñoz et al., 2015). Both latter algorithms are wrapper methods that evolve features transformations of the form $g : \mathbb{R}^n \rightarrow \mathbb{R}^p$, such that each input pattern $\mathbf{x} \in \mathbb{R}^n$ is transformed into a new feature vector $\mathbf{x} \in \mathbb{R}^p$ that is easier to classify using a predefined classification rule $\mathcal{R}$. In this domain, the context is provided by $\mathcal{R}$; in this work we use a simple threshold (Zhang and Smart, 2006) and a minimum Mahalanobis distance criterion (Muñoz et al., 2015); as described below.

9.1.1 *Accuracy Descriptor (AD)*

The proposed descriptor considers the accuracy of a classifier at a fine scale, which is closely related to its semantics but also takes into account the context provided by the classification rule $\mathcal{R}$. Here we consider supervised classification problems specified by a training set T which contains sample patterns from each class. The accuracy descrip-



(a) Optimal behavior (b) Current behavior (c) Behavior landscape

Figure 9.1: Graphical depiction of the Accuracy Descriptor (AD): (a) shows the optimal behavior; (b) shows a possible behavior; and (c) shows the underlying fitness landscape abased on behavior space.

tor (AD) is constructed in the following way. If $T = \{\mathbf{x}_1, ..., \mathbf{x}_L\}$, then the behavior descriptor for each GP individual K_j is a binary vector $\beta_j^{AD} = (\beta_1, \beta_{j,2}, ...\beta_{j,i}.., \beta_{j,L})$ of size L, where each vector element $\beta_{j,i}$ is set to 1 if the classification rule $\mathcal{R}$ correctly classifies sample $\mathbf{x}_i$ based on the output provided by K_j, and is set to 0 otherwise. The proposed descriptor is illustrated in Figure 9.1.

To understand the underlying behavior space specified by AD, lets consider a binary classification problem. The AD descriptor specifies β as a binary vector, and suppose that function $u(\beta)$ returns the number of 1s in β. Moreover, let K_O be the *optimal* transformation that achieves a perfect accuracy on the training set based on $\mathcal{R}$. The AD of K_O is given by β_O^{AD} where $u(\beta_O^{AD}) = L$. Moreover, for a two-class problem, an equally useful solution is to take the *worst* transformation K_W, that achieves a complete misclassification in the whole training set given by β_W^{AD} where $u(\beta_W^{AD}) = 0$, then use the opposite (complement) behavior and invert the classification. This mirror behaviors is a β^{AD} with $u(\beta^{AD}) = 0$. These two optima are shown in the underlying behavior fitness landscape depicted in Figure 9.1(c), when fitness is given by the classification accuracy. For a problem with a reasonable degree of difficulty, the initial generations of a GP search should be expected to contain close to random classifiers, with roughly a 50% accuracy.

Therefore, early individuals will mostly exhibit behavior descriptors with equal proportions of zeros and ones. Then, NS will necessarily explore towards two opposite points in behavior space.

Just like in semantic space, the specified fitness landscape based on behaviors defines clear global optima (Uy et al., 2011b; Beadle and Johnson, 2008, 2009; Krawiec and Pawlak, 2013). However, for supervised classification semantic space would not be useful, since the global semantic optima is not known beforehand, it necessarily depends upon the training set T and the specified classification rule $\mathcal{R}$. It is important to mention that (Moraglio et al., 2012) also proposed geometric operators for GP-based classifiers based on semantics, however that formulation is not applicable for wrapper methods such as (Zhang and Smart, 2006; Muñoz et al., 2015). Moreover, while semantic approaches have been widely used for symbolic regression (Castelli et al., 2015), they have not been fully studied in supervised classification. Finally, given the above binary descriptors, a natural distance function to apply NS with Equation 4.11 showed in Chapter 4, is the Hamming distance (any other suitable distance function for binary strings could be used), which counts the number of bits that differ between two binary vectors.

9.1.2 *Binary Classifier: Static Range Selection*

For binary classification, we use the Static Range Selection GP Classifier (SRS-GPC) described by Zhang and Smart (Zhang and Smart, 2006). In a binary problem, a pattern $\mathbf{x} \in \Re^n$ has to be classified as belonging to one of either two classes, ω_1 or ω_2. In this method, the goal is to evolve a mapping $g(\mathbf{x}) : \Re^n \rightarrow \Re$. The classification rule $\mathcal{R}$ states that pattern $\mathbf{x}$ is labelled as belonging to class ω_1 if $g(\mathbf{x}) > r$, and belongs to ω_2 otherwise. The fitness function is simple, it consists on maximizing the total classification accuracy after $\mathcal{R}$ is applied, normally setting the decision boundary to $r = 0$; the process is depicted in Figure 9.2.

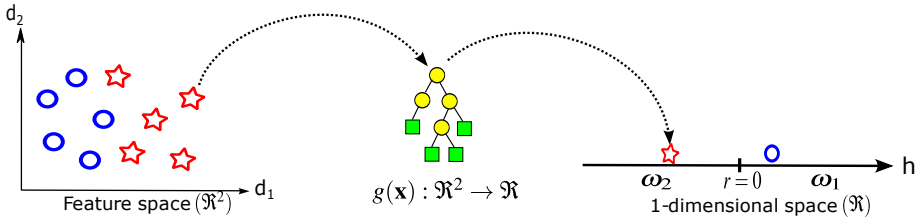


Figure 9.2: Graphical depiction of SRS-GPC.

9.1.3 *Multiclass Classifier: M3GP*

To test the NS approach on multiclass problems, we choose the recently proposed Multidimensional Multiclass GP with multidimensional populations (M3GP) (Muñoz et al., 2015). The goal of M3GP is to evolve a mapping $g(\mathbf{x}) : \Re^n \rightarrow \Re^p$, where p is determined by the evolutionary process. The classification rule $\mathcal{R}$ used by M3GP consists of the following steps: (1) build M p-dimensional Gaussian clusters, one for each class, using the training set; (2) classify each training instance based on the minimum Mahalanobis distance to each class cluster, this provides the total accuracy used as fitness measure. For testing, the clusters found during training are used to determine class membership using the same distance measure. Results reported with M3GP are quite competitive with state of the art methods, such as Random Forests (Breiman, 2001) and Random Subspaces (Ho, 1998; Bryll et al., 2003) (Muñoz et al., 2015), providing a more advanced GP-based classification algorithm compared with SRS-GPC. Nonetheless, given the proposed behavior descriptor we can use the same NS algorithms in both cases. In essence, each classifier provides a different context for the outputs generated by the GP trees, just like previous works in evolutionary robotics solved different navigation tasks in different environments, using the same descriptor and NS algorithm (Lehman and Stanley, 2008, 2010b,c, 2011a).

Summarizing, SRS-GPC and M3GP are used to evaluate the NS approach, respectively on binary and multiclass problems. For both

algorithms, the traditional approach will hereafter be referred to as objective-based search (OS). However, before testing these algorithms on real-world datasets, we evaluate the proposed NS approach on synthetic binary problems.

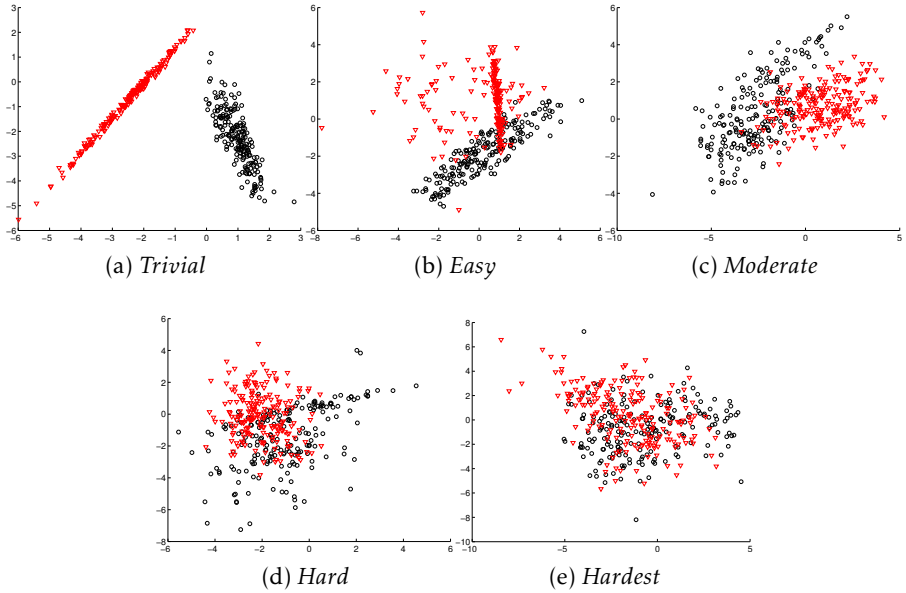
9.1.4 *Preliminary Results*

Gaussian Mixture Models (GMM) are used to generate five random synthetic problems. All problems are set in the $\mathbb{R}^2$ plane with $x, y \in [-10, 10]^2$, and 200 sample points were randomly generated for each class. The parameters for the GMM of each class were also randomly chosen, following the same strategy reported in (Trujillo et al., 2011a). The five problems are of increasing difficulty (according to SRS-GPC performance), denoted as: *Trivial*; *Easy*; *Moderate*; *Hard*; and *Hardest*; these problems are depicted in Figure 9.3.

Here we compare SRS-GPC using either the objective function (OS) or novelty (NS). The parameters of both GP systems are given in Table 9.1. Moreover, for NS method the parameters are set as follows: (1) the number of neighbors considered for sparseness computation is set to $k = 15$; (2) the sparseness threshold is set to $\rho_{th} = 40$; and (3) the archive is implemented as a FIFO queue of three times the population size. Both algorithms are executed 30 times and performance is analyzed based on averages over all runs. For each problem, 70% of the data is used for training and the rest for testing, with random partitions for each run. Performance is measured based on the total classification error and all algorithms were coded using Matlab and the GPLAB toolbox (Silva and Almeida, 2003).

9.1.5 *Discussion*

Two aspects will be discussed; classification error and the average size (number of nodes) of the individuals in the population. First, Table 9.2 presents the average classification error on the test data. The results are consistent with those reported in (Naredo et al., 2013b), NS



(a) *Trivial* (b) *Easy* (c) *Moderate*

(d) *Hard* (e) *Hardest*

Figure 9.3: Five synthetic 2-class problems used to evaluate each algorithm in ascending order of difficulty (according with the GP objective-based standard performance) from left to right.

performs well, achieving basically the same results compared to OS. However, there is a trend, NS seems to perform relatively better on the more difficult problems and worse on the easier ones. Basically, the explorative search performed by NS is fully exploited when random initial solution perform badly, in these conditions the search for novelty can lead towards better solutions. Conversely, for easy problems random solutions can perform quite well, thus the search for novelty can lead the search towards solutions with undesirable performance.

The results are encouraging, especially if we consider the evolution of average program size which is related to bloat (Silva and Costa, 2009). Figure 9.4 shows how the average size of all individuals in the population evolves across generations. First, consider OS, Figure 9.4(a) shows typical GP behavior, with a clear tendency of code growth, with

Table 9.1: Parameters for the GP systems.

Parameter	Description
Population size	100 individuals.
Generations	100 generations.
Initialization	*Ramped Half-and-Half*, with 6 levels of maximum depth.
Operator probabilities	Crossover $p_c = 0.8$, Mutation $p_\mu = 0.2$.
Function set	$\{+, -, \times, \div, \lvert\cdot\rvert, x^2, \sqrt{x}, log, sin, cos, if\}$.
Terminal set	$\{x_1, ..., x_i, ..., x_p\}$, where x_i is a dimension of the data patterns $\mathbf{x} \in \Re^n$.
Hard maximum depth	20 levels.
Selection	Tournament of size 4.

Table 9.2: Average and standard deviation of the classification error on the test data, of the best solution found by each algorithm.

Problem	OS	NS
Trivial	0.004 ±0.007	0.007 ±0.008
Easy	0.105 ±0.040	0.144 ±0.044
Moderate	0.136 ±0.033	0.159 ±0.041
Hard	0.260 ±0.052	0.266 ±0.053
Hardest	0.365 ±0.033	0.370 ±0.043

more bloat on difficult problems. In the case of NS, Figure 9.4(b) shows that NS controls code growth quite effectively, exhibiting the same average program size on all problems.

Based on these results, we can revisit the fitness-causes-bloat theory of Langdon and Poli (Langdon and Poli, 1997). It basically states that the search for better fitness (given by the objective function) will bias

Figure 9.4: Evolution of the average size of individuals at each generation, computed on thirty runs; where: (a) represents OS and (b) is for NS.

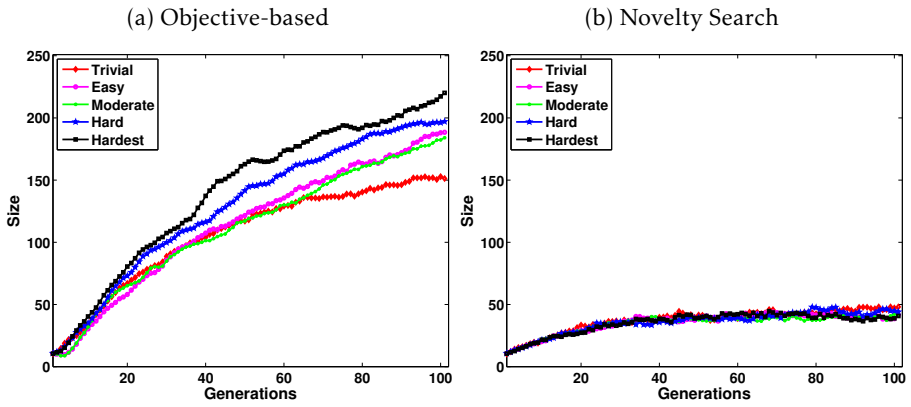


Table 9.3: Average program size at the final generation for each algorithm. For the NS algorithm, the population (Pop), archive (Arch) and both (Pop+Arch) are considered.

Problem	**OS**	**NS Pop**	**NS Arch**	**NS Pop+Arch**
Trivial	151.3	47.9	38.5	46.1
Easy	188.5	41.4	33.6	35.4
Moderate	184	41.3	3.7	38.6
Hard	197	43.8	37.9	42.9
Hardest	220.1	40.7	32.7	38.2

the search towards larger trees, simply because there are more large programs than small ones. Silva and Costa (Silva and Costa, 2009) state it clearly:

> ... one cannot help but notice the one thing that all the [bloat] theories have in common, the one thing that if re-

> moved would cause bloat to disappear, ironically the one thing that cannot be removed without rendering the whole process useless: the search for fitness.

Our experimental results show similarly with those reported in (Lehman and Stanley, 2010b), that show how NS consistently evolves smaller program trees, and this findings correlate nicely with the fitness-causes-bloat theory. This results suggest that the bloat phenomenon was avoided by abandoning an explicit objective function.

9.2 Real-world classification experiments

In this section we present an experimental comparison of all the novelty-based variants discussed thus far, compared against a standard objective-based search. The algorithms are compared on real-world classification problems taken from several public repositories summarized in Table 9.4. The first 8 datasets are used to construct binary classification problems and solved using the SRS-GPC decision rule, the datasets with more than two classes are divided into 12 binary classification tasks. Datasets 1 – 5 are balanced regarding the number of instances per class, while datasets 6 to 12 pose imbalanced problems. The remaining 3 datasets pose the multiclass problems which are solved with M3GP.

In all, 4 different novelty-based algorithms are compared with objective-based search: these are NS, PNS, MCNS and MCPNS, the latter of which is a hybrid that combines PNS with the MC presented in Section 3.1. For the binary classification problems all algorithms use the parameters given in Table 9.1, except that the population size was reduced to 50. Similarly, to compare with M3GP the parameters are slightly modified following (Muñoz et al., 2015), where: population size is 500, initialization used the full method, crossover and mutation probabilities are 0.5, the function set is $F = \{+, -, \times, \div\}$, maximum depth is 17 levels, and selection uses lexicographic tournament of size 4 . All algorithms are executed 30 times and performance is analyzed based on the median value over all runs. In each run 70% of the data is

Table 9.4: Real-world and synthetic datasets for binary and multiclass classification problems, taken from the UC Irvine Machine Learning Repository ◇, from the U.S. geological survey (USGS) earth resources observation systems (EROS) data center ⊗, and from the KEEL dataset repository ⊙.

Type	Dataset Name	Short	Class	Attrib	Inst
Binary	◇Pima Indians Diabetes	Diab	2	8	536
	◇Iris	Iris	3	4	150
	◇Parkinsons	Parkin	2	22	96
	◇Teaching Assist. Eval.	TAE	3	5	147
	◇Wine	Wine	3	13	144
	◇Cardiotocography	Cardio	3	21	2126
	◇Indian Liver Patient	Liver	2	10	579
	◇Fertility	Fertil	2	9	100
Multiclass	⊗Satellite dataset	IM-3	3	6	322
	⊙Segment	SEG	7	19	2310
	⊙Movement-Libras	M-L	15	90	360

used for training and the rest for testing, the data partition is randomly selected for each run. The objective function is given by the classification error, which is used by all NS variants to choose the best solution at the end of the run, and by OS to guide the search.

Since all classification problems differ on the sample size, and because PNS considers all of the individuals generated during the search, for NS and MCNS all the behaviors in the current population and in the population archive are used to compute the novelty measure in Equation 4.11. Moreover, ρ_{th} is set to 50% of the largest possible distance, as well as k–neighbors is set to 50% of the population size, and the archive is a FIFO queue with a size three times that of the population.

For the MCNS algorithm, the minimal criteria for each individual is that it's fitness must be within a certain percentage of the best solution found so far. Six different versions of MCNS are tested, from 5%

to 30% (MCNS5 - MCNS30) of the fitness from the best-so-far solution, in increments of 5%. In this way, any solution that is more than x% worse than the best solution has it's novelty value set to 0. It is important to mention that all the PNS variants do not require the parameters used for NS. Finally, all algorithms were coded using MATLAB and the GPLAB toolbox (Silva and Almeida, 2003).

The algorithms are compared based on training error, test error (error on test data) and the average size of all individuals in the population based on the number of tree nodes (hereafter referred to as average population size). To verify statistical significance, the Friedman test with Bonferroni-Dunn correction of the p-values is performed between the control algorithm (OS) and each NS variant, as suggested in (Derrac et al., 2011). In Tables 9.6 and 9.8 an asterisk indicates that the corresponding novelty-based algorithm achieves statistically better results than OS, with a $\alpha = 0.05$ significance level. The p-values of the statistical tests are given in Tables 9.7 and 9.9.

9.2.1 *Results: Binary Classification*

First, we analyze the performance of different versions of the MCNS algorithm. Table 9.5 shows the median test error for all classification problems; each problem is denoted by an abbreviation (Diab, Parkin, etc., as showed in Table 9.4) followed by the classes used to pose the binary problem (C1, C2, C3). In general, most variants are quite similar, with MCNS15 achieving the overall best results. Therefore, for the sake of clarity and conciseness only MCNS15 is included in the subsequent comparisons with other methods and will simply be referred to as MCNS.

Figure 9.5 presents convergence plots of the median training error of the best solution over generations. In general, OS converges quicker than most NS variants, with PNS showing the most similar convergence. For some problems (plots i-l) PNS converge faster than OS. Indeed, plots (a)-(h) correspond with well balanced problems, with similar amounts of samples for each class. On the other hand, the problems

Table 9.5: Binary classification performance for all MCNS$_{best}$ variants. Table shows the median classification error on the test data for the best solution found, bold indicates the best performance.

Dataset	MC5	MC10	MC15	MC20	MC25	MC30
Diab C1C2	**0.306**	0.325	0.309	0.328	0.338	0.322
Iris C2C3	0.100	0.100	**0.067**	0.100	0.100	0.100
Wine C1C3	0.036	0.054	**0.000**	**0.000**	**0.000**	**0.000**
Wine C2C3	0.125	0.143	**0.036**	0.107	0.071	0.089
Parkin C1C2	0.250	**0.214**	**0.214**	**0.214**	0.214	0.250
TAE C1C2	0.429	0.429	**0.393**	**0.393**	**0.393**	0.429
TAE C1C3	0.411	0.429	**0.321**	0.357	0.357	0.357
TAE C2C3	0.464	0.429	0.411	**0.375**	0.429	0.429
Wine C1C2	0.143	0.143	**0.125**	0.143	0.161	**0.125**
Wine C1C3	0.036	0.054	**0.000**	**0.000**	**0.000**	**0.000**
Wine C2C3	0.125	0.143	**0.036**	0.107	0.071	0.089
Cardio C1C2	0.126	**0.109**	0.119	0.110	0.113	0.111
Liver C1C2	0.298	0.309	**0.286**	0.298	0.289	0.292
Fertil C1C2	**0.138**	**0.138**	**0.138**	0.155	0.172	**0.138**

of plots (i)-(l) present larger class imbalance, suggesting that convergence of some NS algorithms is better on unbalanced problems. However, NS shows the slowest convergence rates, in many cases not reaching the best training error found by other algorithms.

Figure 9.6 presents plots of how the mean size of the population evolves. It is clear that OS pushes GP toward larger program sizes on these problems, a trend that is particularly evident in plots (a)-(j). In general most NS variants produce smaller GP trees, with some noteworthy tendencies. First, NS evolves smaller trees in all cases except for the Fertility C1C2 problem. In most cases the difference with OS is quite large, a result that is consistent with the experiments presented in Section 9.1.4. Second, MCNS also evolves very small trees, in some cases ((b),(c),(g),(h),(i)) the average program size is smaller than those generated by NS. This is a particularly encouraging result given the

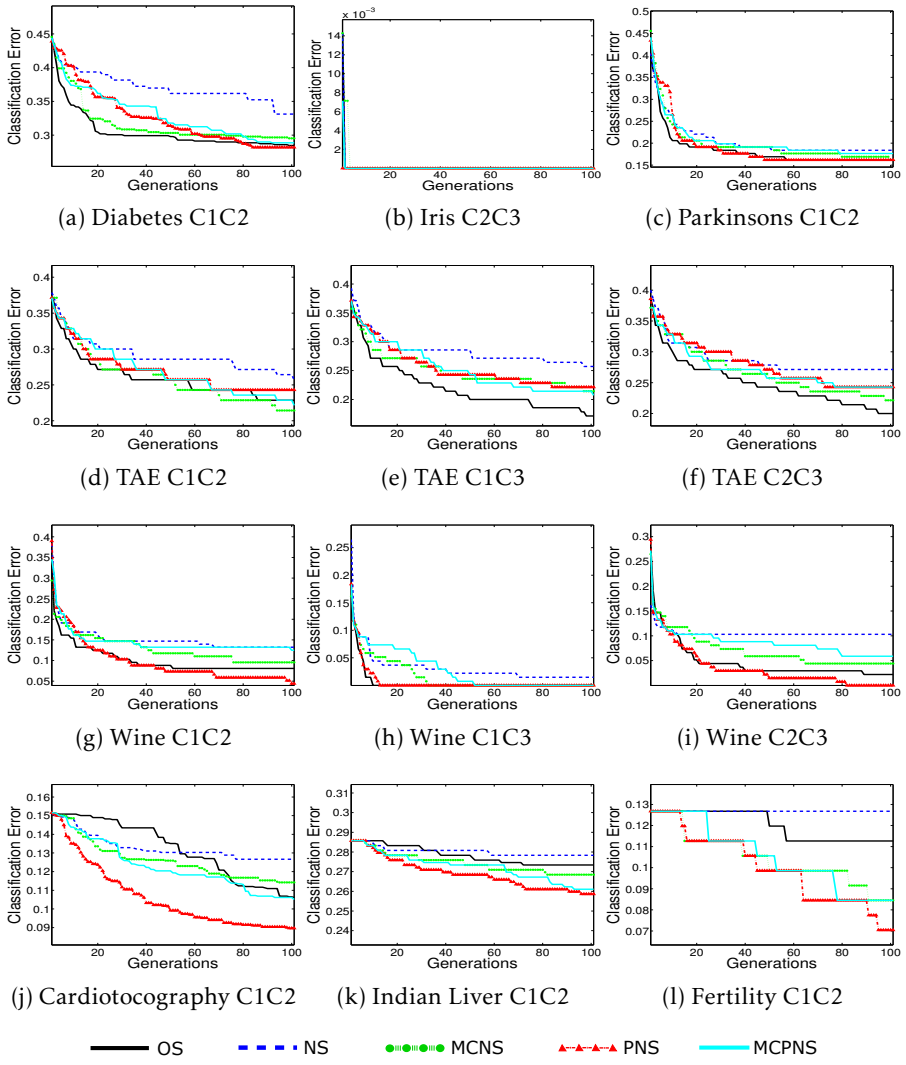


Figure 9.5: Convergence of the classification error on the training data for the best solution found, showing the median value over all runs.

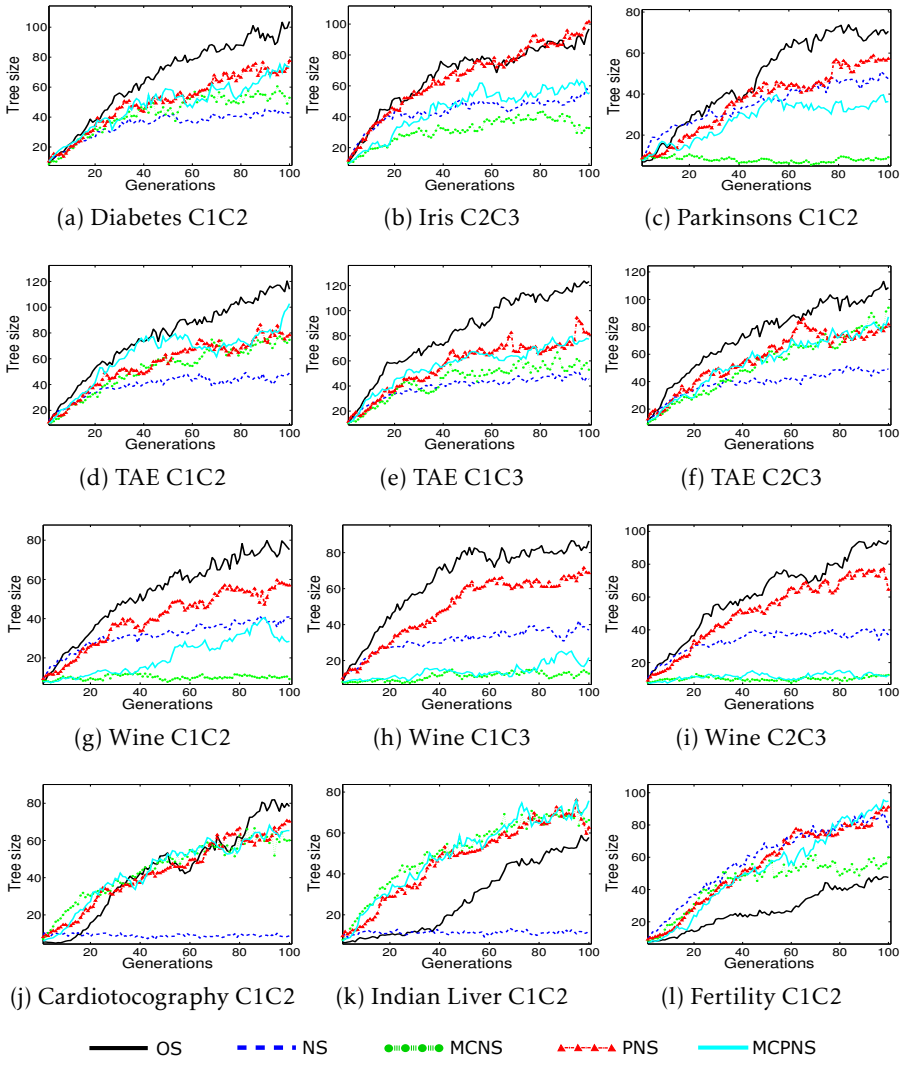


Figure 9.6: Evolution of the average size of the population at each generation, showing the median value over all runs.

quality of the solutions found by MCNS. Third, PNS in some cases ((a),(c),(d),(e),(f),(g),(h)) also evolves smaller trees than OS. Finally, OS evolves smaller trees than most methods on two problems, Liver C1C2 and Fertility C1C2. It seems that this result may be correlated with the slow convergence of the search on these problem, produced by the high class imbalance in both datasets.

Numerical comparisons based on the test error and average program size are presented in Table 9.6. Moreover, the p-values of the statistical tests, comparing each method with the control method OS, are given in Table 9.7. To illustrate the performance differences among the methods, Figure 9.7 shows a box plot comparison of the test error from the best solution found in each run.

In general, all NS algorithms are very competitive relative to OS. In fact, all methods show statistically equivalent results based on test error with only three exceptions: NS is worse on Wine C2C3; PNS is better on Cardio C1C2; PNS is worse on Liver C1C2; and MCPNS is worse on Fertility C1C2. Even in the cases where the differences are statistically significant, the relative difference is rather low. However, when we consider the average program size it is clear that all NS variants evolve much smaller populations. In particular NS and MCNS show an intrinsic bloat-control property in this domain. There is one exception, in Fertility C1C2 NS evolves larger populations, which might be related to the class imbalance issue. Regarding PNS and MCPNS, both also control code growth on several problems with respect to OS, but the difference is less compared to NS and MCNS.

Finally, it is instructive to comment on the results for the most imbalanced problems, Liver C1C2 and Fertility C1C2. If we inspect the convergence plots in Figure 9.5, several trends appear. In both cases (plots (k) and (l)), the search performed by OS and NS seem to be stagnated, with only minimal improvements across generations. It is evident that these algorithms have converged to an almost trivial solution for imbalanced problems; i.e, assign to all (or a large majority) of the samples the class label of the majority class. In these cases, such trivial solutions can be generated randomly and will have the same objective score, thus the selective pressure will be null, making OS function as

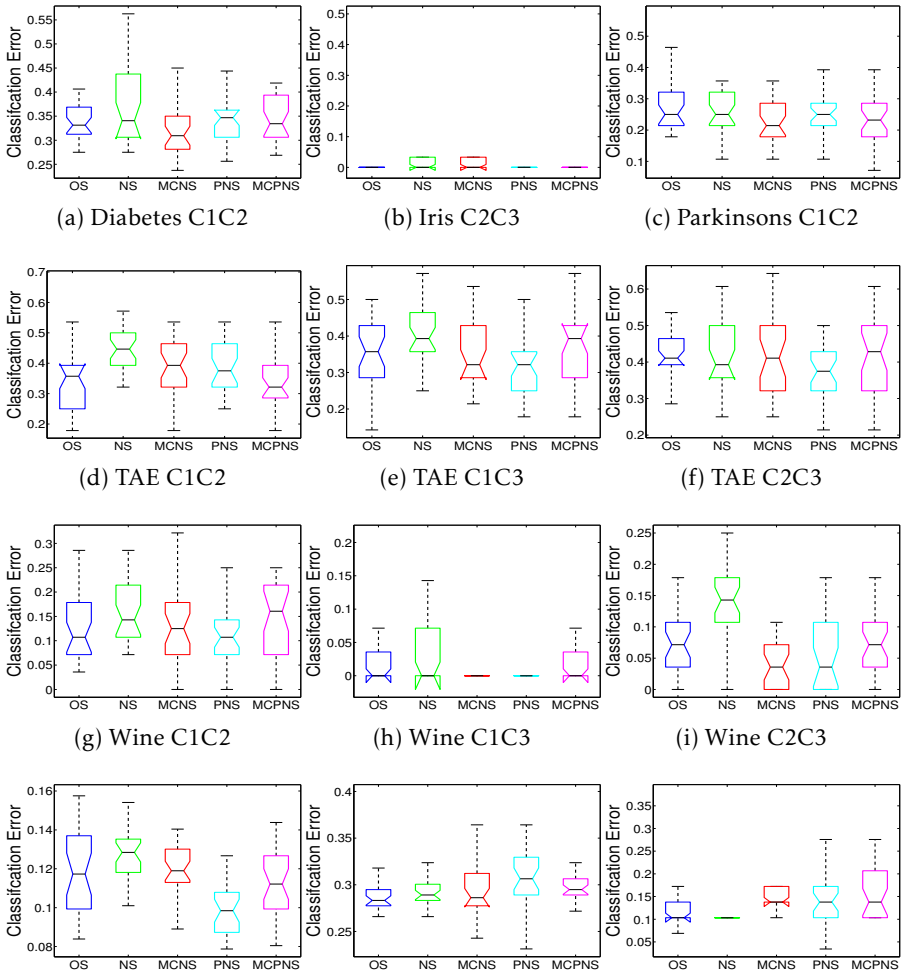
Table 9.6: Binary classification performance, showing the median classification error on the test data (Test) for the best solution found, and the median of the average program size in the last generation (A-size). Statistically significant with respect to the control method with a p-value less than 0.05 is marked with an asterisk (*).

Dataset	**Measure**	**OS**	**NS**	**MCNS**	**PNS**	**MCPNS**
Diabet	Test	0.331	0.341	0.309	0.347	0.334
C1C2	A-size	103.84	42.77*	52.92*	77.47	73.54
Iris	Test	0.067	0.067	0.067	0.067	0.100
C2C3	A-size	96.81	55.15*	32.69*	101.56	54.70
Parkin	Test	0.250	0.250	0.214	0.250	0.232
C1C2	A-size	70.61	47.48*	9.20*	57.26	36.40*
TAE	Test	0.357	0.446	0.393	0.375	0.321
C1C2	A-size	113.91	49.04*	74.66*	78.39*	102.74
TAE	Test	0.357	0.393	0.321	0.321	0.393
C1C3	A-size	123.30	45.36*	52.93*	81.36*	77.32*
TAE	Test	0.411	0.393	0.411	0.375	0.429
C2C3	A-size	108.64	49.03*	93.94*	81.73*	87.27
Wine	Test	0.107	0.143	0.125	0.107	0.161
C1C2	A-size	75.20	40.93*	9.25*	56.97	28.45*
Wine	Test	0.000	0.000	0.000	0.000	0.000
C1C3	A-size	86.51	37.03*	12.79*	68.67	21.71*
Wine	Test	0.071	0.143*	0.036	0.036	0.071
C2C3	A-size	94.43	36.83*	12.46*	64.51	12.44*
Cardio	Test	0.117	0.128	0.119	0.098*	0.112
C1C2	A-size	78.27	8.79*	60.10	70.17	65.27
Liver	Test	0.283	0.289	0.286	0.306*	0.295
C1C2	A-size	57.81	11.91*	66.31	62.59	75.77
Fertil	Test	0.103	0.103	0.138	0.138	0.138*
C1C2	A-size	47.50	78.01*	59.91	91.07*	94.75*

Table 9.7: Resulting p-values of the Friedman's test with Bonferroni-Dunn correction, for the binary classification problems using OS as the control method. The null hypothesis is rejected with a p-value less than 0.05, marked with an asterisk (*).

Dataset	Measure	NS	MCNS	PNS	MCPNS
Diabet	Test	2.86	0.27	2.86	3.41
C1C2	A-size	0.00*	0.01*	0.58	0.27
Iris	Test	0.182	0.057	0.629	1.269
C1C3	A-size	0.00	0.00	0.58	1.09
Parkin	Test	3.39	0.11	2.78	1.41
C1C2	A-size	0.04*	0.00*	1.09	0.00*
TAE	Test	0.07	0.71	0.24	3.39
C1C2	A-size	0.00*	0.00*	0.04*	1.09
TAE	Test	0.09	1.34	1.41	4.00
C1C3	A-size	0.00*	0.00*	0.00*	0.01*
TAE	Test	3.39	1.80	0.41	3.41
C2C3	A-size	0.00*	0.04*	0.01*	0.11
Wine	Test	2.26	2.78	4.00	3.39
C1C2	A-size	0.00*	0.00*	1.09	0.00*
Wine	Test	0.63	4.00	3.13	0.36
C1C3	A-size	0.00*	0.00*	0.11	0.00*
Wine	Test	0.01*	0.47	0.13	1.73
C2C3	A-size	0.00*	0.00*	0.58	0.00*
Cardio	Test	0.78	1.41	0.02*	0.33
C1C2	A-size	0.00*	0.58	1.09	1.86
Liver	Test	1.41	1.03	0.03*	0.05
C1C2	A-size	0.00*	1.09	1.09	0.58
Fertil	Test	0.79	0.16	0.29	0.01*
C1C2	A-size	0.00*	0.58	0.00*	0.01*

a random search. This is consistent with the size of the evolved programs generated by OS on these problems, which are among the small-



(a) Diabetes C1C2 (b) Iris C2C3 (c) Parkinsons C1C2

(d) TAE C1C2 (e) TAE C1C3 (f) TAE C2C3

(g) Wine C1C2 (h) Wine C1C3 (i) Wine C2C3

(j) Cardiotocography C1C2 (k) Indian Liver C1C2 (l) Fertility C1C2

Figure 9.7: Classification error on the test data for the best solution found, showing box plots of the median value over all runs.

est of all the algorithms as shown in Figure 9.6. Conversely, the convergence plots for PNS, MCNS and MCPNS are substantially different,

displaying a clear tendency towards incremental improvement during the training phase. However, their respective classification error on the test data is a bit worse but not statistically significant.

9.2.2 *Results: Multiclass Classification*

In these tests, we use three problems (IM-3, SEG and M-L) with different number of classes (3, 7 and 15). Moreover, the SEG problem has 2,310 instances, which leads to a very large behavior descriptor; i.e., using 70% of the data for training, we obtain a descriptor length of 1,617 bits, which gives a very large behavior space. Assessing the performance of the NS variants on this problem is of particular interest, since previous works have shown that the performance of NS degrades when behavior space is very large (Kistemaker and Whiteson, 2011).

The numerical comparison of the algorithms is given in Tables 9.8 and 9.9, the former shows the median test error and median population size, while the latter shows the corresponding p-values of the statistical tests. Similar to the binary case, all NS variants achieve basically the same performance as OS, with slight improvements on some problems (particularly M-L), but not statistically significant. Indeed the similarity in terms of performance is even more evident when we analyze the convergence plots for each problem shown in Figure 9.8, which shows how the classification error evolves for the best solution found on the training and testing sets. On the other hand, code growth is not controlled like in the previous tests, in only one problem (M-L) the NS variants produces statistically significant differences in terms of average program size.

9.2.3 *Results: Analysis*

The above results show that NS can be used to solve binary and multiclass classification problems, without a performance drop-off relative to standard OS. This was not expected, given that the search process omits the use of a standard objective function. Moreover, on some prob-

Table 9.8: Multiclass classification performance, showing the median classification error on the test data (Test) for the best solution found, and the median of the average program size in the last generation (A-size). Statistically significant with respect to the control method with a p-value less than 0.05 is marked with an asterisk (*).

Dataset	Measure	OS	NS	MCNS	PNS	MCPNS
IM-3	Test	0.046	0.052	0.062	0.052	0.046
	A-size	66.22	71.08	55.05	55.42	53.84
SEG	Test	0.044	0.043	0.042	0.043	0.041
	A-size	111.10	143.61	104.57	138.48	123.01
M-L	Test	0.429	0.414	0.394	0.394	0.444
	A-size	13.05	12.77*	12.82*	12.63*	12.69*

Table 9.9: Resulting p-values of the Friedman's test with Bonferroni-Dunn correction, for the multiclass classification problems using OS as the control method. The null hypothesis is rejected with a p-value less than 0.05, marked with an asterisk (*).

Dataset	Measure	NS	MCNS	PNS	MCPNS
IM-3	Test	1.73	0.57	2.78	2.12
	A-size	1.09	2.86	0.58	1.86
SEG	Test	2.86	1.41	0.28	1.79
	A-size	0.11	2.86	1.86	2.86
M-L	Test	1.79	0.37	1.86	3.36
	A-size	0.00*	0.04*	0.04*	0.00*

lems, mainly in binary classification tasks, the NS variants provide an intrinsic bloat control property.

This section provides a deeper analysis of the experimental results. Regarding the MC variants, namely MCNS and MCPNS, we show the impact of the penalty assigned in Equation 4.12 when the MC is not satisfied. Figure 9.9 plots the percentage of individuals that did not

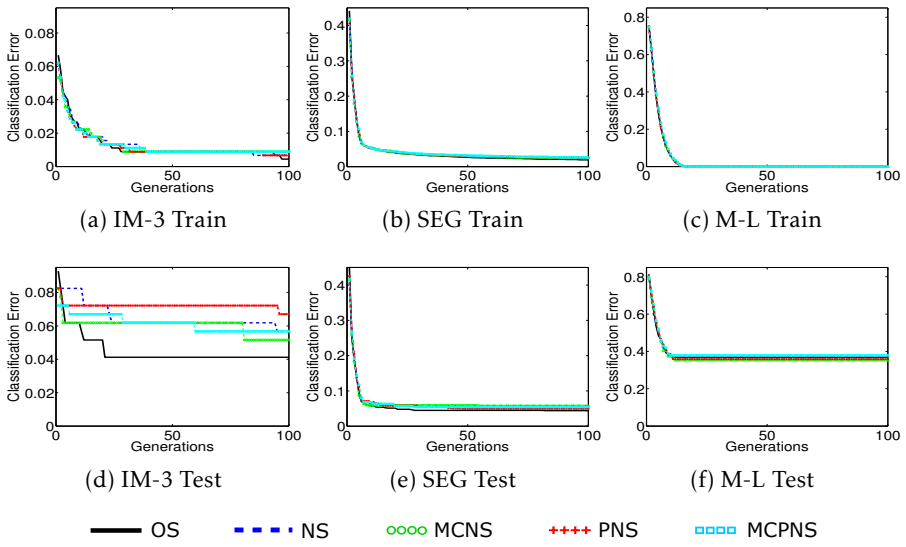


Figure 9.8: Convergence of training and testing error for multiclass problems.

satisfy the MC at each generation, for all three multiclass problems. For both algorithms, we can see very similar patterns on all problems. At the beginning of the run most individuals do not satisfy the MC, the number of rejected individuals then quickly declines and then more or less stabilizes after about 20 to 40 generations.

Another important aspect is to consider the computational cost of each NS method, relative to standard OS. Moreover, as stated before, PNS is posed as an approximation of the original NS algorithm, working under a naive Bayesian assumption. PNS characterizes the complete distribution of behaviors, but uses a possibly unsatisfiable assumption to simplify the novelty measure. NS with a FIFO archive computes a more realistic measure of sparseness but sacrifices historical information to keep computational overhead low. One possible alternative would be to run NS without an archive limit, such that all individuals generated by the search are included into the archive. How-

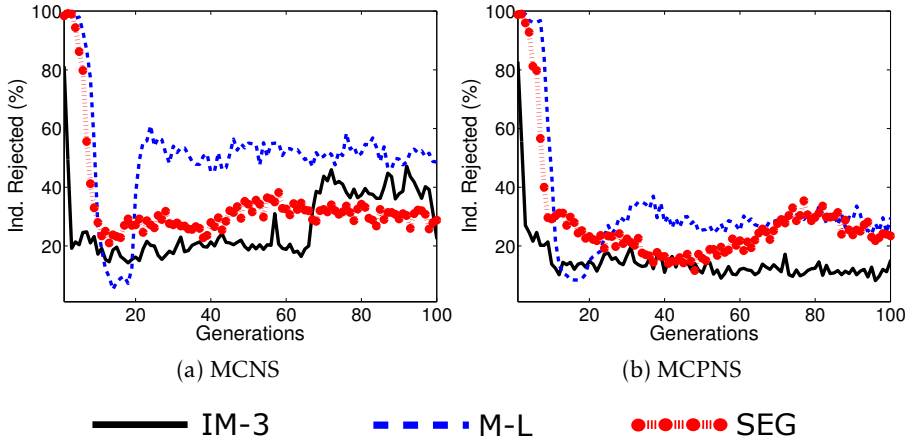
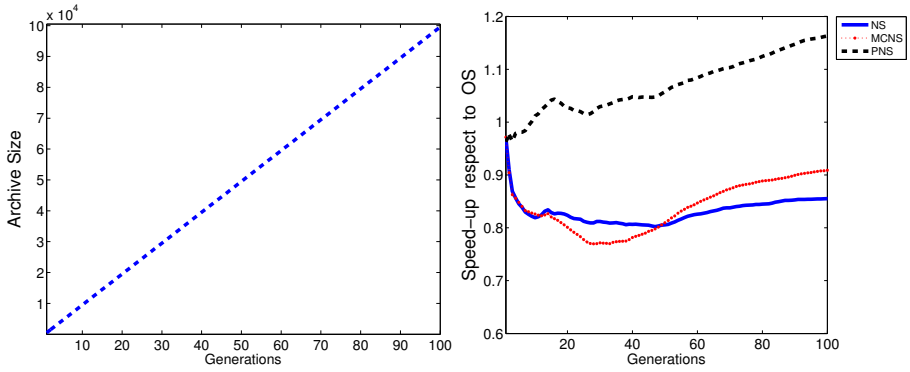


Figure 9.9: Plot showing the percentage of rejected individuals, with respect to the population size, that did not satisfy the MC.

ever, computing the NS in this scenario would surely have a detrimental effect on computational efficiency. Figure 9.10(a) shows how the size of the population archive grows when all individuals are stored during NS runs on the IM-3 problem, we refer to this variant as NSn. We can see that the size of the archive grows linearly, reaching very large values by the end of the run.

Figure 9.10(b) plots the median speed-up of each method relative to OS on the same problem, based on all runs. Values above unity represent a proportional reduction in total CPU time and values below unity represent an increase in total CPU time. In this plot we compare NS, PNS and MCNS. These experiments were carried out on a Windows 7 PC with an Intel Xeon e3-1220 v3 64-bit processor @ 3.10 Ghz and 8 GB of RAM. Results show the advantage of using PNS, with CPU run times showing a slight speed-up relative to OS of about 10%. On the other hand, NS and MCNS clearly pay the price of the more expensive sparseness estimation method. It is correct to assume that NSn would fair even worse, since it uses a much larger novelty archive.



(a) Archive Size (b) Relative Speed-up

Figure 9.10: Analysis of NS variants on the IM-3 problem: (a) Archive size for NSn; (b) Relative speed-up for NS, PNS and MCNS.

Finally, to compare the selective pressure applied by each algorithm, Figures 9.11 and 9.12 compare the relative ranking of each individual in the population for problems IM-3 and SEG; the plots for problem M-L are omitted because they are very similar to those of IM-3. The left column of plots in each figure (indices a,c and e) shows the percentage of individuals in the population that are ranked as the best solution of each generation (the top-ranked solutions). In Figures 9.11(a) and 9.12(a) the runs were guided by OS, in Figures 9.11(c) and 9.12(c) by NS, and in Figures 9.11(e) and 9.12(e) selective pressure was applied by PNS. Conversely, the right column of plots in these figures (indices b,d and f) shows the average ranking of the best individuals at each generation based on the other two fitness measures.

For instance, Figure 9.11(a) shows that the number of top-ranked individuals is relatively small in the first 25 generations using OS, afterwards the number of top-ranked individuals grows quickly and then stabilizes and oscillates at around 20% of the total population. Notice that this 25 generation threshold coincides with the moment at which training error converges on this problem, as shown in Figure 9.8(a).

Figure 9.11(b) takes these top-ranked individuals and computes new ranks for them based on the NS and PNS novelty measures, and the plot shows their average rank based on these methods. For NS, the plot shows that the ranking provided by the sparseness measure disagrees substantially with OS in the initial generations, reaching a peek at generation 20. Afterwards, the ranking of individuals by NS is basically equivalent with that of OS, this corresponds with the similar convergence plots of both algorithms. PNS, on the other hand, differs with OS only at the very beginning of the run, afterwards the ranking is the same across all generations. Figure 9.11(c) and Figure 9.11(d) show a similar comparison, but in this case NS provides the selective pressure. Notice that the percentage of top-ranked individuals is very similar to OS, but in this case we can see larger disagreement in terms of ranking by the other methods. In particular, we can see that OS shows large disagreement at the beginning of the runs (up to generation 25), afterwards the differences start to reduce and are equal after generation 70. On the other hand, PNS ranking is more erratic, we can see that in some generations the ranking of the best individuals is in complete agreement with NS, while in other moments the average difference can become quite high (around generation 60). Finally, when PNS applies the selective pressure, Figure 9.11(e) and Figure 9.11(f), we can see a different pattern. First, the percentage of top-ranked individuals is always small, suggesting that PNS does not converge to many similar behaviors. Second, OS and NS mostly agree with PNS ranking after the first initial generations, but surprisingly NS shows a larger ranking difference than OS.

Figure 9.12 presents a similar analysis on the SEG problem. However, in this case we can see different behaviors. The percentage of top-ranked individuals oscillates with OS, while NS and PNS exhibit similar patterns, with a small number of individuals achieving the top-rank every generation. When selective pressure is applied by OS, the relative ranking of NS and PNS is quite similar, with large differences at the beginning of the run and then converging to similar rankings at about 40 generations, see Figures 9.12(b). Conversely, when NS and PNS apply selective pressure, we can observe a larger disagreement by the other

two methods, shown in Figures 9.12(d) and 9.12(f). In particular, when NS applies selective pressure we can see that OS disagrees with the rankings throughout most of the run, with the differences progressively declining. On the other hand, the disagreement between PNS and NS seems more erratic, with total agreement in some generations and large disagreements in others, as seen in Figure 9.12(d). Figure 9.12(f) shows the relative differences in ranking when PNS applies selective pressure. In this case, both OS and NS disagree with PNS throughout most of the run, but both methods converge to similar rankings at the end of the search.

In summary, these plots provide useful insights regarding the different selective pressure applied by each method. The results confirm that the naive Bayesian assumption made by PNS does in fact lead to differences in search dynamics relative to NS, this might partially explain the differences in average solution size by both methods in binary classification tasks. However, these plots also show that while PNS and NS are preferring different individuals during some portions of the run, after a certain number of generations these differences start to decline. This is a plausible explanation for the similar performance achieved by all methods in terms of classification error.

9.3 Chapter Conclusions

This chapter presented for the first time an application of the NS approach to supervised classification with GP, with several contributions. First, the concept of behavior space is framed as a conceptual middle-ground between the well-known concept of objective space and the recently popular semantic space in GP. Second, a domain-specific descriptor has been proposed and tested on supervised classification tasks, considering synthetic and real-world data as well as binary and multi-class problems. The proposed descriptor is a binary vector, where each element corresponds with each fitness case in the training set, taking a value of 1 when that fitness case is correctly classified and a 0 value otherwise. Third, two extensions to the basic NS approach have been de-

veloped, PNS and MCNS, as well as a hybrid method MCPNS. PNS provides a probabilistic framework to measure a solution's novelty, eliminating all of the underlying NS parameters while reducing the computational overhead that the original NS algorithm suffers from. On the other hand, the proposed MCNS extends the minimal criteria approach by combining the objective function with the sparseness measure, constraining the NS algorithm by specifying a minimal solution quality, a dynamic criterion that is proportional to the quality of the best solution found so far.

Experimental results are evaluated based on two measures, solution quality and average size of all solutions in the population. In terms of performance, results show that all NS variants achieve competitive results relative to the standard OS approach in GP. These results show that the general open-ended approach towards evolution followed by NS can compete with objective driven search in traditional machine learning domains. On the other hand, in terms of solutions size and the bloat phenomenon, the NS approach can lead the search towards maintaining smaller program trees, particularly in the simpler binary tasks. In particular, NS and MCNS show substantial reductions in program size relative to OS.

Finally, a promising aspect of the present work is that several future lines of research can be explored, in no particular order we contemplate the following. Firstly, there seems to be a possible link between the PNS algorithm and two similar methods in evolutionary computation, estimation of distribution algorithms (EDAs) (Larrañaga and Lozano, 2001), the frequency fitness assignment (FFA) method (Weise et al., 2014), and fitness sharing methods (Nguyen et al., 2012). While EDAs use a distribution over genotype space to generate new individuals, PNS uses a distribution in behavior space to measure the novelty of each solution. FFA favors solutions with unique objective scores, instead of uniqueness in behavior space as done in PNS.

Nonetheless, many of the theoretical and practical insights derived from EDA and FFA research might be brought to bear during further development of the PNS approach, while further comparisons with recent diversity preservation techniques might also be of interest

(Nguyen et al., 2012). Secondly, we might extend the proposed PNS variants in other ways, such as testing PNS with real-valued behavior descriptors or simply apply PNS within semantic space, similar to the approach suggested in (Castelli et al., 2014). Fourthly, the effect that NS has on bloat should be studied further, it is clear that standard NS and MCNS provide the best bloat control, but is unclear why this effect was not observed on the multiclass problems. Finally, the proposed algorithms should be evaluated in other machine learning problems, such as unsupervised clustering (Naredo and Trujillo, 2013) and symbolic regression (Martínez et al., 2013).

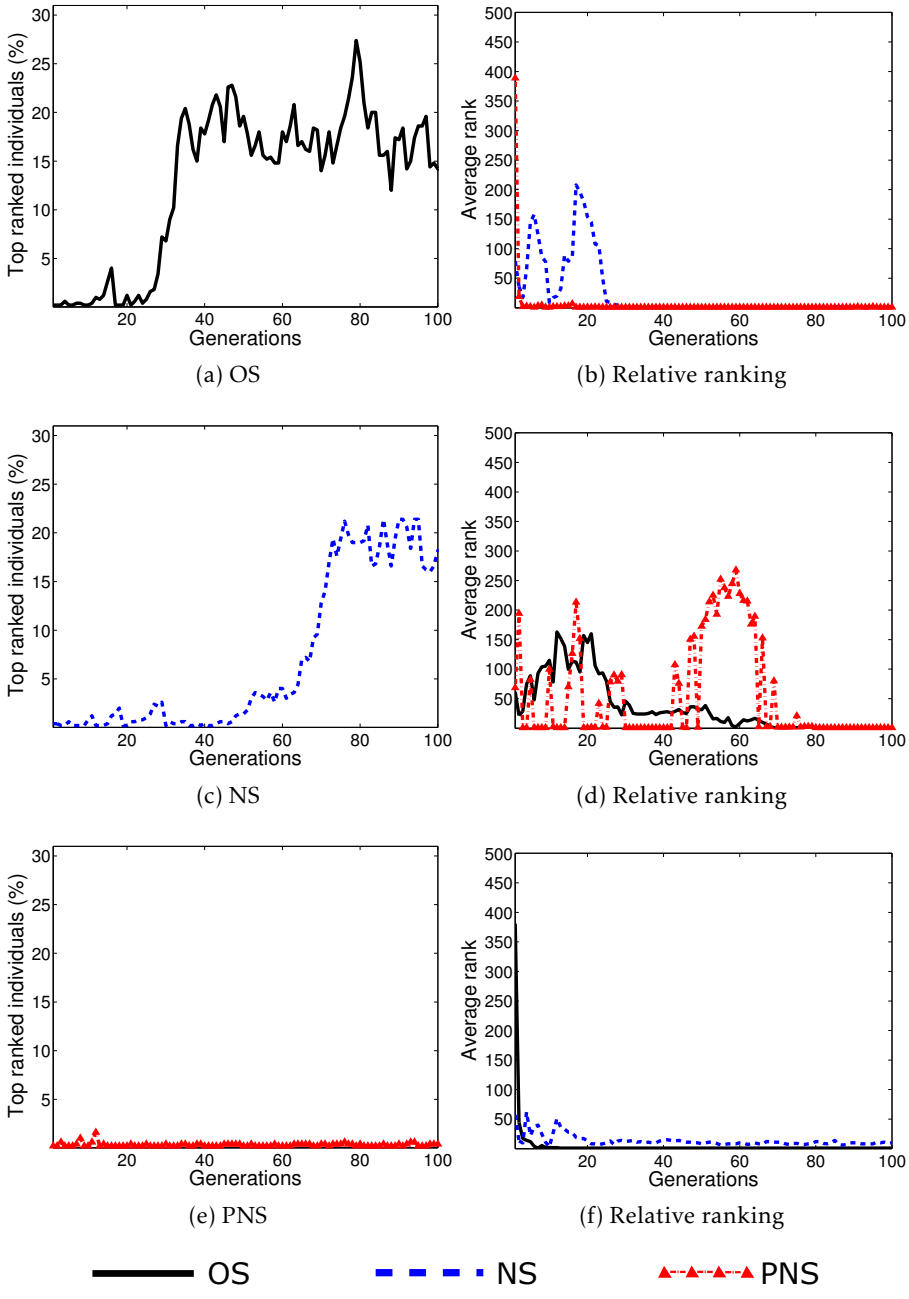


Figure 9.11: Relative ranking of NS, PNS and OS on the IM-3 problem.

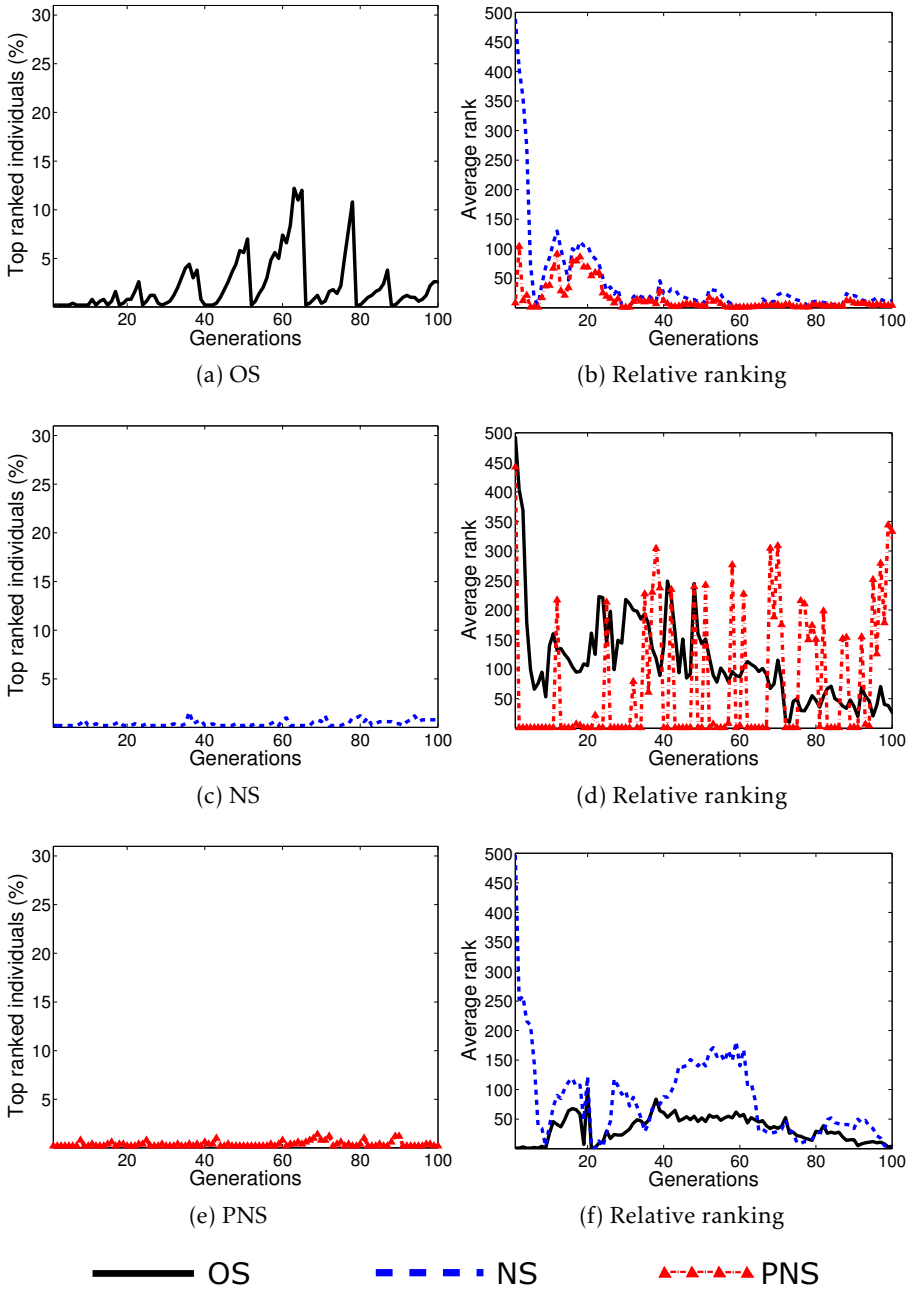


Figure 9.12: Relative ranking of NS, PNS and OS on the SEG problem.

10

Conclusions & Future Directions

The initial motivation of this research was to explore how the open issues in GP (O'Neill et al., 2010) could be tackled. We must admit that at the beginning we were somewhat skeptical of the benefits that could be obtained by the use of an unorthodox search strategy based on solution novelty instead of solution quality. Even more, because we found some criticism about NS at conferences, comparing it with a random or exhaustive search. On the other hand, more recently Velez and Clune (2014) showed indeed that NS has general exploration skills and does not behaves as a random search, which we were also able to verify and extend. With due caution, we started studying NS and exploring how could we tackle the open issues in GP. First applying NS with a simple GA for a real-world circuit synthesis problem, and then by showing the generalization that can be achieved by using NS in evolutionary robotics. Afterward we were able to apply NS on several machine learning tasks, especially regression, clustering and more completely on supervised classification. We also proposed new variants of NS, showing that several of its issues could be mitigated.

In general, we must say that out initial findings of applying NS in machine learning are very encouraging, even though GP systems based on NS were not better than traditional objective-based search in many cases, they showed competitive results against its traditional counterpart, with several advantages; namely, its better performance on more difficult problems and the reduction in bloat during evolution. All this was somewhat unexpected, because NS is a counter-intuitive approach; searching without an explicit objective!

10.1 Summary and Conclusions

In Chapter 2 we introduced the basics of GP (Koza, 1992a) and present a case study to show how it can be used to address the disparity map problem in stereo computer vision. The notion of deception was introduced in Chapter 3, which is frequently present in real-world problems and closely related with problem difficulty. To the best of our knowledge this work is the first attempt to design a deceptive classification problem (Naredo et al., 2015).

In Chapter 4 the NS algorithm (Lehman and Stanley, 2011a) was presented, particularly we highlight its main difference against traditional evolutionary search which is driven by an objective, while NS drives the search by rewarding the most novel (unique) solutions (Stanley and Lehman, 2015). Two new versions of NS are proposed; MCNS_{bsf} and PNS (Naredo et al., 2016b). The first version is an extension of the progressive minimal criteria NS (PMCNS). The second one is a probabilistic approach to compute novelty, this last proposal has the advantage that it eliminates all of the underlying NS parameters, and at the same time reduces the computational overhead from the original NS algorithm.

A first and original case study of applying NS in a GA-based search was presented to synthesize topologies of current follower (CF) circuits in Chapter 5. Experimental results confirms that NS can be used as a promising alternative in the field of automatic circuit synthesis (Naredo et al., 2016a).

The problem of generalization in evolutionary robotics (Urbano et al., 2014a; Naredo et al., 2016c) was studied in Chapter 6. To the best of our knowledge, previous works have not studied the effect that the training set size has on generalization for a navigation problem in ER. Experimental results clearly suggest that NS improves the generalization abilities of the GE system, outperforming both objective-based search and random search.

Next step in our research using NS was to test it on machine learning and pattern recognition problems: regression, clustering and supervised classification. NS-based GP applied to solve the problem of sym-

bolic regression was studied in Chapter 7, the first attempt to apply NS in this domain (Martínez et al., 2013). To do so, a domain-specific behavioral descriptor was proposed; the epsilon Descriptor (β^{ϵ}). Results were encouraging and we considered to expand the use of the NS algorithm to other mainstream areas.

NS-based GP was used to search for data clustering functions in Chapter 8, also the first attempt to apply NS in this domain (Naredo and Trujillo, 2013). To do so, the Cluster Descriptor (β^{CD}) was proposed, to characterize behaviors in this domain. Results show that NS-based GP performs better on the most difficult problems, out performing standard techniques.

In Chapter 9 a GP system based on NS was used to search for novel classifiers (Naredo et al., 2013b, 2016b). To the best of our knowledge this is the first time NS was used to solve supervised classification problems. To do so, a domain-specific behavioral descriptor was proposed: the Accuracy Descriptor (β^{AD}). We tested our approach on two GP classifiers: a simple binary classifier based based on a static threshold (Zhang and Smart, 2006) and a recently proposed multiclass approach (Ingalalli et al., 2014; Muñoz et al., 2015). Two new versions of NS were also extensively evaluated, namely MCNS$_{bsf}$ and PNS. Experimental results are evaluated based on two measures, solution quality and average size of all solutions in the population. In terms of performance, results show that all NS variants achieve competitive results relative to the standard OS approach in GP. These results show that the general open-ended approach towards evolution followed by NS can compete with objective driven search in traditional machine learning domains. On the other hand, in terms of solutions size and the bloat phenomenon, the NS approach can lead the search towards maintaining smaller program trees, particularly in the binary problems. In particular, NS and MCNS show substantial reductions in program size relative to OS.

10.2 Open Issues in GP

As stated in Chapter 6.1, one of the motivations for studying NS in GP was to contribute in addressing some of the main open issues in the field. Let us run through those issues again and explain why it is that we have met this general but guiding goal.

OPEN-ENDED EVOLUTION IN GP: *Design an evolutionary system capable of continuously adapting and searching.*

This issue is obviously addressed by the NS approach, indeed one of a limited amount of heuristic strategies that all but guarantees that the search can proceed in an open-ended manner. Indeed, we showed that open-ended search is not a niche strategy, it can be used to solve traditional learning tasks effectively, and based on some measures outperforms standard OS.

GP BENCHMARKS: *Is it possible to define a set of test problems that can be rigorously evaluated by the scientific community and then accepted as a more or less agreed upon set of benchmarks for experimental studies in GP?*

We only cover this issue in a very reduced form, but a very novel one in Chapter 3. We have defined the first synthetic benchmark problems that define a deceptive fitness landscape for supervised classification. The contribution is unique and shows promise, but future work will have to extend our initial contribution and experimentally validate its usefulness.

FITNESS LANDSCAPES AND PROBLEM DIFFICULTY IN GP: *Identifying how hard a particular problem, or problem instance, will be for some GP system, enabling a practitioner to make informed choices before and during application.*

This is one of the most important open issues in GP, and our results might help contribute towards understanding it better. In particular, most of our results show that NS-based GP usually performs better when the problem is harder. If we accept that NS is a good strategy to solve deceptive problems, then the obtained results would suggest

that difficult learning problems do have a degree of deceptiveness. As future work builds on our research, and NS is applied to more problem instances and problem domains, then these insights might help the development of better GP search algorithms. Moreover, as we point out before, Chapter 3 proposed a set of classification problems with deceptive fitness landscapes, the first such benchmarks in related literature.

GENERALIZATION IN GP: *Defining the nature of generalization in GP to allow the community to deal with generalization more often and more rigorously, as in other ML fields and statistical analysis.*

This issue is one of the most important ones in practice, and for stochastic methods it is of particular importance. Chapter 6 is focused exclusively on this issue and provides unique results that show how NS-based learning is more general than the traditional approach. Future work will have to extend this analysis to the machine learning tasks we studied here.

10.3 Future Work

There are several future research lines to extend the work related with deception, one is to take into account the methods based on data separation to design synthetic classification problems that can introduce a deception degree for this kind of methods. Another research line is to apply non-linear classifiers, incorporating different approaches such as genetic programming to generate the classifiers.

With respect to work related with the evolution of CF circuits, one possible research line is to optimize the evolved topologies of the CF circuits and to subject them to real-world experimental validation. Another is to apply the NS paradigm to synthesis other specialized circuits of interest in the field of electronic design automation. Furthermore, we can enhance the NS approach by attempting to force the search away from specific areas of the search space. For example, it should be possible to seed the population with previously known designs that should be avoided by the search, since they are not as interesting. In this way,

the NS algorithm could be used to explicitly search for circuits that are unique in electronic literature.

We can extend our original work on generalization into two different directions; first by grouping the initial conditions into easy-difficult regions, and second by using different deceptive task navigations.

With respect to the machine learning problems, future work can focus on to provide a deep comparison between the proposed behavior-based search strategy and recent semantics-based approaches, a comparison that goes beyond merely experimental results, but a detailed analysis of the main algorithmic differences between both approaches and their effects on search. Furthermore, future work on this domain should also study how NS affects the bloat phenomenon during a GP search.

Particularly with respect to the proposal of computing novelty through a probability approach, there seems to be a possible link between the PNS algorithm and two similar methods in evolutionary computation, estimation of distribution algorithms (EDAs) Larrañaga and Lozano (2001), the frequency fitness assignment (FFA) method Weise et al. (2014). The proposed algorithms should be evaluated in other machine learning problems, such as unsupervised clustering Naredo and Trujillo (2013) On the other hand, we might extend the proposed PNS variants in other ways, such as testing PNS with real-valued behavior descriptors or applying PNS within semantic space, similar to the approach suggested in Castelli et al. (2014).

Finally, future work will also focus on making our proposal more stable, since the results indicate a large variance in the NS-based runs, which is understandable given the nature of the search. However, we believe that the best approach is not to combine the objective function and the novelty into a single fitness value or to use a multiobjective formulation. It is our opinion that the best way to move forward is to use NS to explore the search space and to integrate a local search method to exploit individuals that exhibit promising new behaviors **?**.

Banzhaf, W. (2014). Genetic programming and emergence. *Genetic Programming and Evolvable Machines*, 15(1):63–73.

Banzhaf, W., Francone, F. D., and Nordin, P. (1996). The effect of extensive use of the mutation operator on generalization in genetic programming using sparse data sets. In *In Parallel Problem Solving from Nature IV, Proceedings of the International Conference on Evolutionary Computation, edited by*, pages 300–309. Springer Verlag.

Beadle, L. and Johnson, C. (2008). Semantically driven crossover in genetic programming. In *Proceedings of the Tenth Conference on Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, CEC'08, pages 111–116. IEEE Press.

Beadle, L. and Johnson, C. G. (2009). Semantically driven mutation in genetic programming. In *Proceedings of the Eleventh Conference on Congress on Evolutionary Computation*, CEC'09, pages 1336–1342. IEEE Press.

Bezdek, J., Ehrlich, R., and Full, W. (1984). Using direct competition to select for competent controllers in evolutionary robotics. *Fcm: The fuzzy c-means clustering algorithm*, 10(2–3):191–203.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Brameier, M. F. and Banzhaf, W. (2010). *Linear Genetic Programming*. Springer Publishing Company, Incorporated, 1st edition.

Breiman, L. (2001). Random forests. *Machine learning*, pages 5–32.

Brooks, R. A. (1999). *Cambrian intelligence: the early history of the new AI*. MIT Press, Cambridge, MA, USA.

Bryll, R., Gutierrez-Osuna, R., and Quek, F. (2003). Attribute bagging: improving accuracy of classifier ensembles by using random feature subsets. *Pattern Recognition*, 36(6):1291 – 1302.

Burke, E. K., Gustafson, S., Kendall, G., and Krasnogor, N. (2004). *Is Increased Diversity in Genetic Programming Beneficial? An Analysis of Lineage Selection*. PhD thesis, University of Nottingham, UK.

Castelli, M., Manzoni, L., Silva, S., and Vanneschi, L. (2010). A comparison of the generalization ability of different genetic programming frameworks. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8.

Castelli, M., Manzoni, L., Silva, S., and Vanneschi, L. (2011). A quantitative study of learning and generalization in genetic programming. In Silva, S., Foster, J. A., Nicolau, M., Machado, P., and Giacobini, M., editors, *EuroGP*, volume 6621 of *Lecture Notes in Computer Science*, pages 25–36. Springer.

Castelli, M., Trujillo, L., Vanneschi, L., and Popovič, A. (2015). Prediction of energy performance of residential buildings: A genetic programming approach. *Energy and Buildings*, 102:67 – 74.

Castelli, M., Vanneschi, L., and Silva, S. (2014). Semantic search-based genetic programming and the effect of intron deletion. *IEEE Transactions on Cybernetics*, 44(1):103–113.

Cuccu, G., Gomez, F. J., and Glasmachers, T. (2011a). Novelty-based restarts for evolution strategies. In *IEEE Congress on Evolutionary Computation*, pages 158–163. IEEE.

Cuccu, G., Gomez, F. J., and Glasmachers, T. (2011b). Novelty-based restarts for evolution strategies. In *IEEE Congress on Evolutionary Computation*, pages 158–163. IEEE.

Das, R. and Whitley, D. (1991). *The only challenging problems are deceptive: global search by solving order-1 hyperplanes*. Number no. 102 in Technical report (Colorado State University. Department of Computer Science). Colorado State University, Department of Computer Science.

Dawkins, R. (1986). *The Blind Watchmaker: Why the evidence of evolution reveals a universe without design*. W.W. Norton and Company.

Dawkins, R. (1996). *Climbing Mount Improbable*. W.W. Norton & Company.

Day, R. O. and Lamont, G. B. (2004). Multi-objective fast messy genetic algorithm solving deception problems. In *Congress on Evolutionary Computation*, page 23.

Deb, K. and Goldberg, D. (1994). Sufficient conditions for deceptive and easy binary functions. *Annals of Mathematics and Artificial Intelligence*, 10(4):385–408.

Deb, K. and Goldberg, D. E. (1993). Analyzing deception in trap functions. In Whitley, L. D., editor, *Foundations of Genetic Algorithms 2*, pages 93–108. Morgan Kaufmann, San Mateo, CA.

Dempsey, I., O'Neill, M., and Brabazon, A. (2009). *Foundations in Grammatical Evolution for Dynamic Environments*, volume 194 of *Studies in Computational Intelligence*. Springer.

Derrac, J., García, S., Molina, D., and Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3–18.

Doucette, J. and Heywood, M. I. (2010). *Genetic Programming: 13th European Conference, EuroGP 2010, Istanbul, Turkey, April 7-9, 2010. Proceedings*, chapter Novelty-Based Fitness: An Evaluation under the Santa Fe Trail, pages 50–61. Springer Berlin Heidelberg, Berlin, Heidelberg.

Duarte-Villaseñor, M. A., Tlelo-Cuautle, E., and Fraga, L. G. (2011). Binary genetic encoding for the synthesis of mixed-mode circuit topologies. *Circuits, Systems, and Signal Processing*, 31(3):849–863.

Duda, R. O., Hart, P. E., and Stork, D. G. (2000). *Pattern Classification (2Nd Edition)*. Wiley-Interscience.

Ebner, M. (2009). A real-time evolutionary object recognition system. In Vanneschi, L., Gustafson, S., Moraglio, A., Falco, I. D., and Ebner, M., editors, *Genetic Programming*, pages 268–279. Springer.

Flores Becerra, G., Polanco Martagon, S., Duarte Villaseñor, M., Tlelo Cuautle, E., de la Fraga, L., and Guerra Gomez, I. (2014). Selection of the optimal sizes of analog integrated circuits by fuzzy sets intersection. *Latin America Transactions, IEEE (Revista IEEE America Latina)*, 12(6):1005–1011.

Francone, F. D., Nordin, P., and Banzhaf, W. (1996). Benchmarking the generalization capabilities of a compiling genetic programming system using sparse data sets. In *Proceedings of the 1st Annual Conference on Genetic Programming*, pages 72–80, Cambridge, MA, USA. MIT Press.

Gathercole, C. and Ross, P. (1994). Dynamic training subset selection for supervised learning in genetic programming. In *Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature: Parallel Problem Solving from Nature*, PPSN III, pages 312–321, London, UK, UK. Springer-Verlag.

Georgiou, L. (2012). *Constituent Grammatical Evolution*. PhD thesis, School of Computer Science, Bangor University, Bangor, Gwynedd, United Kingdom.

Georgiou, L. and Teahan, W. J. (2006). jge - a java implementation of grammatical evolution. In *10th WSEAS International Conference on Systems*, pages 534–869, Athens, Greece.

Georgiou, L. and Teahan, W. J. (2010). Grammatical evolution and the santa fe trail problem. In *International Conference on Evolutionary Computation (ICEC)*, pages 10–19, Valencia, Spain. SciTePress.

Gielen, G. and Rutenbar, R. (2000). Computer-aided design of analog and mixed-signal integrated circuits. *Proceedings of the IEEE*, 88(12):1825–1854.

Goldberg, D. E. (1987). Simple genetic algorithms and the minimal, deceptive problem. In Davis, L., editor, *Genetic algorithms and simulated annealing*, Research Notes in Artificial Intelligence, pages 74–88. Pitman.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional.

Gomes, J., Mariano, P., and Christensen, A. L. (2015). Devising effective novelty search algorithms: A comprehensive empirical study. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO '15, pages 943–950, New York, NY, USA. ACM.

Gomes, J., Urbano, P., and Christensen, A. (2012). Progressive minimal criteria novelty search. In Pavón, J., Duque-Méndez, N., and Fuentes-Fernández, R., editors, *Advances in Artificial Intelligence – IBERAMIA 2012*, volume 7637 of *Lecture Notes in Computer Science*, pages 281–290. Springer Berlin Heidelberg.

Gomes, J., Urbano, P., and Christensen, A. (2013). Evolution of swarm robotics systems with novelty search. *Swarm Intelligence*, 7(2-3):115–144.

Gonçalves, I. and Silva, S. (2011a). Experiments on controlling overfitting in genetic programming. In *15th Portuguese Conference on Artificial Intelligence (EPIA 2011)*.

Gonçalves, I. and Silva, S. (2011b). Experiments on controlling overfitting in genetic programming. In *15th Portuguese Conference on Artificial Intelligence*. EPIA 2011.

Gonçalves, I. and Silva, S. (2013). Balancing learning and overfitting in genetic programming with interleaved sampling of training data. In Krawiec, K., Moraglio, A., Hu, T., Etaner-Uyar, A., and Hu, B., editors, *Genetic Programming*, volume 7831 of *Lecture Notes in Computer Science*, pages 73–84. Springer Berlin Heidelberg.

Gonçalves, I., Silva, S., and Fonseca, C. (2015). On the generalization ability of geometric semantic genetic programming. In *18th European Conference on Genetic Programming (EuroGP 2015)*, n/a.

Gonçalves, I., Silva, S., Melo, J., and Carreiras, J. a. M. B. (2012). Random sampling technique for overfitting control in genetic programming. In Moraglio, A., Silva, S., Krawiec, K., Machado, P., and Cotta, C., editors, *Genetic Programming*, volume 7244 of *Lecture Notes in Computer Science*, pages 218–229. Springer Berlin Heidelberg.

Grefenstette, J. J. (1993). Deception considered harmful. In Whitley, D. L., editor, *Foundations of Genetic Algorithms 2*, pages 75–91. Morgan Kaufmann, San Mateo, CA.

Haupt, R. L. and Haupt, S. E. (2004). *Practical genetic algorithms*. J. Wiley, Hoboken, N.J.

Hernández, B., Olague, G., Hammoud, R., Trujillo, L., and Romero, E. (2007). Visual learning of texture descriptors for facial expression recognition in thermal imagery. *Computer Vision and Image Understanding, Special Issue on Vision Beyond the Visual Spectrum*, 106(2-3):258–269.

Ho, T. K. (1998). The random subspace method for constructing decision forests. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(8):832–844.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI.

Horn, J. and Goldberg, D. (1995). Genetic algorithms, problem difficulty, and the modality of fitness landscapes. In *FOGA'95*, volume 3.

Howard, D., Roberts, S. C., and Brankin, R. (1999). Target detection in sar imagery by genetic programming. *Advances in Engineering Software*, 30(5):303–311.

Ingalalli, V., Silva, S., Castelli, M., and Vanneschi, L. (2014). A multi-dimensional genetic programming approach for multi-class classification problems. In Nicolau, M., Krawiec, K., Heywood, M. I., Castelli, M., García-Sánchez, P., Merelo, J. J., Rivas Santos, V. M., and Sim, K., editors, *Genetic Programming*, volume 8599 of *Lecture Notes in Computer Science*, chapter A Multi-dimensional Genetic Programming Approach for Multi-class Classification Problems, pages 48–60. Springer Berlin Heidelberg.

Jain, A. K. (2010). Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651–666.

Jones, T. (1994). A description of holland's royal road function. *Evol. Comput.*, 2(4):409–415.

Jones, T. and Forrest, S. (1995). Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 184–192. Morgan Kaufmann.

Kaelbling, L. P., Littman, M. L., and Moore, A. P. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285.

Kistemaker, S. and Whiteson, S. (2011). Critical factors in the performance of novelty search. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, pages 965–972. ACM.

Kowaliw, T., Dorin, A., and McCormack, J. (2012). Promoting creative design in interactive evolutionary computation. *Evolutionary Computation, IEEE Transactions on*, 16(4):523 –536.

Koza, J. (2010). Human-competitive results produced by genetic programming. *Genetic Programming and Evolvable Machines*, 11(3):251–284.

Koza, J. R. (1992a). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

Koza, J. R. (1992b). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Complex adaptive systems. MIT Press, Cambridge, MA, USA.

Koza, J. R., Jones, L. W., Keane, M. A., Streeter, M. J., and Al-Sakran, S. H. (2005). *Genetic Programming Theory and Practice II*, chapter Toward Automated Design of Industrial-Strength Analog Circuits by Means of Genetic Programming, pages 121–142. Springer US, Boston, MA.

Koza, J. R., Keane, M. A., Yu, J., Forrest H. Bennett, I., and Mydlowec, W. (2000). Automatic creation of human-competitive programs and controllers by means of genetic programming. *Genetic Programming and Evolvable Machines*, 1(1-2):121–164.

Koza, J. R., Streeter, M. J., and Keane, M. A. (2008). Routine high-return human-competitive automated problem-solving by means of genetic programming. *Information Sciences*, 178(23):4434–4452.

Krawiec, K. (2002). Genetic programming-based construction of features for machine learning and knowledge discovery tasks. *Genetic Programming and Evolvable Machines*, 3(4):329–343.

Krawiec, K. and Bhanu, B. (2005). Visual learning by coevolutionary feature synthesis. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 35(3):409–425.

Krawiec, K. and Pawlak, T. (2013). Locally geometric semantic crossover: a study on the roles of semantics and homology in recombination operators. *Genetic Programming and Evolvable Machines*, 14(1):31–63.

Kushchu, I. (2002a). An evaluation of evolutionary generalisation in genetic programming. *Artif. Intell. Rev.*, 18(1):3–14.

Kushchu, I. (2002b). Genetic programming and evolutionary generalization. *IEEE Trans. Evolutionary Computation*, 6(5):431–442.

Langdon, W. and Poli, R. (2001). *Foundations of Genetic Programming*. Springer, Berlin, Heidelberg, New York.

Langdon, W. B. and Poli, R. (1997). Fitness causes bloat. In *Proceedings of the Second On-line World Conference on Soft Computing in Engineering Design and Manufacturing*, pages 13–22. Springer-Verlag.

Larrañaga, P. and Lozano, J. A. (2001). *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, Norwell, MA, USA.

Lehman, J. and Stanley, K. O. (2008). Exploiting open-endedness to solve problems through the search for novelty. In *Proceedings of the Eleventh International Conference on Artificial Life, Cambridge, MA*, ALIFE XI. MIT Press.

Lehman, J. and Stanley, K. O. (2010a). Efficiently evolving programs through the search for novelty. In Pelikan, M. and Branke, J., editors, *GECCO*, pages 837–844. ACM.

Lehman, J. and Stanley, K. O. (2010b). Efficiently evolving programs through the search for novelty. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, GECCO '10, pages 837–844. ACM.

Lehman, J. and Stanley, K. O. (2010c). Revising the evolutionary computation abstraction: Minimal criteria novelty search. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, pages 103–110. ACM.

Lehman, J. and Stanley, K. O. (2011a). Abandoning objectives: Evolution through the search for novelty alone. *Evol. Comput.*, 19(2):189–223.

Lehman, J. and Stanley, K. O. (2011b). Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, pages 211–218, New York, NY, USA. ACM.

Levitis, D. A., Lidicker Jr, W. Z., and Freund, G. (2009). Behavioural biologists do not agree on what constitutes behaviour. *Animal Behaviour*, 78(1):103–110.

Luke, S. (2013). *Essentials of Metaheuristics*. Lulu, second edition. Available for free at http://cs.gmu.edu/~sean/book/metaheuristics/.

Mahler, S., Robilliard, D., and Fonlupt, C. (2005). Tarpeian bloat control and generalization accuracy. In Keijzer, M., Tettamanzi, A., Collet, P., van Hemert, J. I., and Tomassini, M., editors, *Proceedings of the 8th European Conference on Genetic Programming*, volume 3447 of *Lecture Notes in Computer Science*, pages 203–214, Lausanne, Switzerland. Springer.

Martens, E. and Gielen, G. (2008). Classification of analog synthesis tools based on their architecture selection mechanisms. *Integration, the {VLSI} Journal*, 41(2):238–252.

Martínez, Y., Naredo, E., Trujillo, L., and López, E. G. (2013). Searching for novel regression functions. In *IEEE Congress on Evolutionary Computation*, pages 16–23.

Martínez, Y., Naredo, E., Trujillo, L., Pierrick, L., and López, U. (2016). A comparison of fitness-case sampling methods for genetic programming. *Submitted to: Journal of Experimental & Theoretical Artificial Intelligence, currently working with the reviewers' comments*.

Martínez, Y., Trujillo, L., Naredo, E., and Legrand, P. (2014). A comparison of fitness-case sampling methods for symbolic regression with genetic programming. In *EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation V*, volume 288 of *Advances in Intelligent Systems and Computing*, pages 201–212. Springer International Publishing.

Mazumder, P. and Rudnick, E. M., editors (1999). *Genetic Algorithms for VLSI Design, Layout & Test Automation*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

McDermott, J., Galván-Lopéz, E., and O'Neill, M. (2011). A fine-grained view of phenotypes and locality in genetic programming. In Riolo, R., Vladislavleva, E., and Moore, J. H., editors, *Genetic Programming Theory and Practice IX*, Genetic and Evolutionary Computation, pages 57–76. Springer New York.

McDermott, J., White, D. R., Luke, S., Manzoni, L., Castelli, M., Vanneschi, L., Jaskowski, W., Krawiec, K., Harper, R., De Jong, K., and O'Reilly, U.-M. (2012). Genetic programming needs better benchmarks. In *Proceedings of the 14th Annual Genetic and Evolutionary Computation Conference*, GECCO '12, pages 791–798, New York, NY, USA. ACM.

Mengshoel, O. J., Goldberg, D. E., and Wilkins, D. C. (1998). Deceptive and other functions of unitation as bayesian networks.

Moraglio, A., Krawiec, K., and Johnson, C. G. (2012). Geometric semantic genetic programming. In *Proceedings of the 12th international conference on Parallel Problem Solving from Nature - Volume Part I*, PPSN'12, pages 21–31, Berlin, Heidelberg. Springer-Verlag.

Mouret, J.-B. (2011). Novelty-based multiobjectivization. In Doncieux, S., Bredèche, N., and Mouret, J.-B., editors, *New Horizons in Evolutionary Robotics*, volume 341 of *Studies in Computational Intelligence*, pages 139–154. Springer Berlin Heidelberg.

Mouret, J.-B. and Doncieux, S. (2012). Encouraging behavioral diversity in evolutionary robotics: An empirical study. *Evolutionary Computation*, 20(1):91–133.

Muñoz, L., Silva, S., and Trujillo, L. (2015). M3gp – multiclass classification with gp. In Machado, P., Heywood, M. I., McDermott, J.,

Castelli, M., García-Sánchez, P., Burelli, P., Risi, S., and Sim, K., editors, *Genetic Programming*, volume 9025 of *Lecture Notes in Computer Science*, pages 78–91. Springer International Publishing.

Naik, T. R. and Dabhi, V. K. (2013). Improving generalization ability of genetic programming: Comparative study. *CoRR*, abs/1304.3779.

Naredo, E., Duarte-Villaseñor, M. A., García-Ortega, M. d. J., Vázquez-López, C. E., and Trujillo, L. (2016a). Novelty search for the synthesis of current followers. *Submitted to: Information Siences Journal*.

Naredo, E., Dunn, E., and Trujillo, L. (2013a). Disparity map estimation by combining cost volume measures using genetic programming. In Schütze, O., Coello Coello, C. A., Tantar, A.-A., Tantar, E., Bouvry, P., Del Moral, P., and Legrand, P., editors, *EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation II*, volume 175 of *Advances in Intelligent Systems and Computing*, pages 71–86. Springer Berlin Heidelberg.

Naredo, E. and Trujillo (2013). Searching for novel clustering programs. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13. ACM.

Naredo, E., Trujillo, L., Fernández De Vega, F., Silva, S., and Legrand, P. (2015). Diseñando problemas sintéticos de clasificación con superficie de aptitud deceptiva. In *X Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB 2015)*, Mérida, España.

Naredo, E., Trujillo, L., Legrand, P., Silva, S., and Muño, L. (2016b). Evolving genetic programming classifiers with novelty search. *To appear: Information Siences Journal*.

Naredo, E., Trujillo, L., and Martínez, Y. (2013b). Searching for novel classifiers. In *Proceedings from the 16th European Conference on Genetic Programming, EuroGP 2013*, volume 7831 of *LNCS*, pages 145–156. Springer-Verlag.

Naredo, E., Urbano, P., and Trujillo, L. (2016c). The training set and generalization in grammatical evolution for autonomous agent navigation. *Soft Computing*, pages 1–18.

Nelson, A. L., Barlow, G. J., and Doitsidis, L. (2009). Fitness functions in evolutionary robotics: A survey and analysis. *Robot. Auton. Syst.*, 57(4):345–370.

Nguyen, Q., Nguyen, X., O'Neill, M., and Agapitos, A. (2012). An investigation of fitness sharing with semantic and syntactic distance metrics. In *Proceedings of the 15th European Conference on Genetic Programming, EuroGP'12*, pages 109–120. Springer Berlin Heidelberg.

Nicoară, E. S. (2009). Mechanisms to avoid the premature convergence of genetic algorithms. *Petroleum - Gas University of Ploiesti Bulletin, Mathematics - Informatics - Physics Series*, 61(1):87 – 96.

Nolfi, S. and Floreano, D. (2000). *Evolutionary Robotics: The Biology, Intelligence,and Technology*. MIT Press, Cambridge, MA, USA.

Ofria, C. and Wilke, C. O. (2004). Avida: a software platform for research in computational evolutionary biology. *Artif. Life*, 10(2):191–229.

Olague, G. and Trujillo, L. (2011). Evolutionary-computer-assisted design of image operators that detect interest points using genetic programming. *Image Vision Comput.*, 29(7):484–498.

O'Neill, M. and Ryan, C. (2001). Grammatical evolution. *IEEE Trans. Evolutionary Computation*, 5(4):349–358.

O'Neill, M., Vanneschi, L., Gustafson, S., and Banzhaf, W. (2010). Open issues in genetic programming. *Genetic Programming and Evolvable Machines*, 11(3-4):339–363.

Pelikan, M., Goldberg, D. E., and Lobo, F. G. (2002). A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, 21(1):5–20.

Pérez, C. B. and Olague, G. (2008). Learning invariant region descriptor operators with genetic programming and the f-measure. In *19th International Conference on Pattern Recognition (ICPR 2008), December 8-11, 2008, Tampa, Florida, USA*, pages 1–4. IEEE.

Perez, C. B. and Olague, G. (2009). Evolutionary learning of local descriptor operators for object recognition. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1051–1058, New York, NY, USA. ACM.

Peter, J. M. (2000). *Cartesian Genetic Programming*. Natural Computing Series. Springer-Verlag Berlin Heidelberg, 1st edition.

Poli, R. (1996). Genetic programming for feature detection and image segmentation. In Forgarty, T. C., editor, *AISB Workshop Evolutionary Computing*, pages 110–125.

Poli, R., Langdon, W. B., and McPhee, N. F. (2008a). *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`. (With contributions by J. R. Koza).

Poli, R., Langdon, W. B., and McPhee, N. F. (2008b). *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd.

Puente, C., Olague, G., Smith, S., Bullock, S., Hinojosa-Corona, A., and González-Botello, M. (2011). A genetic programming approach to estimate vegetation cover in the context of soil erosion assessment. *Photogrametric Engineering and Remote Sensing*, 77(4):363–376.

Rada-Vilela, J., Johnston, M., and Zhang, M. (2014). Deception, blindness and disorientation in particle swarm optimization applied to noisy problems. *Swarm Intelligence*, 8(4):247–273.

Rana, S. (1999). Examining the role of local optima and schema processing in genetic search.

Razavi, B. (2001). *Design of Analog CMOS Integrated Circuits*. McGraw-Hill, first edition.

Robilliard, D., Mahler, S., Verhaghe, D., and Fonlupt, C. (2006). Santa fe trail hazards. In Talbi, E.-G., Liardet, P., Collet, P., Lutton, E., and Schoenauer, M., editors, *7th International Conference on Artificial Evolution EA 2005*, volume 3871 of *Lecture Notes in Computer Science*, pages 1–12, Lille, France. Springer.

Romero, J. and Machado, P., editors (2007). *The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music*. Natural Computing Series. Springer Berlin Heidelberg.

Rosca, J. (1996). Generality versus size in genetic programming. In Koza, J. R., Goldberg, D. E., Fogel, D. B., and Riolo, R. L., editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 381–387, Stanford University, CA, USA. MIT Press.

Rutenbar, R., Gielen, G., and Roychowdhury, J. (2007). Hierarchical modeling, optimization, and synthesis for system-level analog and rf designs. *Proceedings of the IEEE*, 95(3):640 – 669.

Schaffer, J. D., Eshelman, L. J., and Offutt, D. (1990). Spurious correlations and premature convergence in genetic algorithms. In *FOGA'90*, pages 102–112.

Shorten, D. and Nitschke, G. (2015). Evolving generalised maze solvers. In Mora, A. M. and Squillero, G., editors, *Applications of Evolutionary Computation*, volume 9028 of *Lecture Notes in Computer Science*, pages 783–794. Springer International Publishing.

Silva, S. and Almeida, J. (2003). Gplab–a genetic programming toolbox for matlab. In Gregersen, L., editor, *Proceedings of the Nordic MATLAB conference*, pages 273–278.

Silva, S. and Costa, E. (2009). Dynamic limits for bloat control in genetic programming and a review of past and current bloat theories. *Genetic Programming and Evolvable Machines*, 10(2):141–179.

Song, A. and Ciesielski, V. (2008). Texture segmentation by genetic programming. *Evol. Comput.*, 16(4):461–481.

Spector, L. (2012). Assessment of problem modality by differential performance of lexicase selection in genetic programming: A preliminary report. In *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '12, pages 401–408, New York, NY, USA. ACM.

Spector, L. and Robinson, A. (2002). Genetic programming and autoconstructive evolution with the push programming language. In *Genetic Programming and Evolvable Machines*, pages 7–40.

Squillero, G. (2005). Microgp - an evolutionary assembly program generator. *Genetic Programming and Evolvable Machines*, 6(3):247–263.

Stanley, K. O. (2004). *Efficient Evolution of Neural Networks Through Complexification*. PhD thesis, Department of Computer Sciences, The University of Texas at Austin.

Stanley, K. O. (2007). Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines*, 8(2):131–162.

Stanley, K. O. and Lehman, J. (2015). *Why Greatness Cannot Be Planned: The Myth of the Objective*. Springer Publishing Company, Incorporated.

Tan, X., Bhanu, B., and Lin, Y. (2005). Fingerprint classification based on learned features. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 35(3):287–300.

Theodoridis, S. and Koutroumbas, K. (2008). *Pattern Recognition, Fourth Edition*. Academic Press, 4th edition.

Theodoridis, S., Pikrakis, A., Koutroumbas, K., and Cavouras, D. (2010). *Introduction to Pattern Recognition: A Matlab Approach*. Academic Press.

Tlelo-Cuatle, E. and Duarte-Villaseñor, M. A. (2008). *Success in Evolutionary Computation*, chapter Evolutionary Electronics: Automatic

Synthesis of Analog Circuits by GAs, pages 165–187. Springer Berlin Heidelberg, Berlin, Heidelberg.

Tlelo-Cuautle, E., Duarte-Villaseñor, M. A., Reyes-García, C. A., and Reyes-Salgado, G. (2007). Automatic synthesis of electronic circuits using genetic algorithms. *Computación y Sistemas*, 10:217–229.

Tlelo-Cuautle, E., Guerra-Gomez, I., Duarte-Villaseñor, M. A., de la Fraga, L. G., Flores-Becerra, G., Reyes-Salgado, G., Reyes-Garcia, C., and Rodriguez-Gomez, G. (2010). Applications of evolutionary algorithms in the design automation of analog integrated circuits. *Journal of Applied Sciences*, 10:1859–1872.

Trujillo, L., Legrand, P., and Lévy-Véhel, J. (2010). The estimation of hölderian regularity using genetic programming. In *GECCO '10: Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 861–868, New York, NY, USA. ACM.

Trujillo, L., Martínez, Y., Galván-López, E., and Legrand, P. (2011a). Predicting problem difficulty for genetic programming applied to data classification. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, pages 1355–1362, New York, NY, USA. ACM.

Trujillo, L., Muñoz, L., Naredo, E., and Martínez, Y. (2014). *Genetic Programming: 17th European Conference, EuroGP 2014, Granada, Spain, April 23-25, 2014, Revised Selected Papers*, chapter NEAT, There's No Bloat, pages 174–185. Springer Berlin Heidelberg, Berlin, Heidelberg.

Trujillo, L., Naredo, E., and Martínez, Y. (2013a). Preliminary study of bloat in genetic programming with behavior-based search. In *EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation IV*, volume 227 of *Advances in Intelligent Systems and Computing*, pages 293–305. Springer International Publishing.

Trujillo, L., Olague, G., Lutton, E., and de Vega, F. F. (2008a). Behavior-based speciation for evolutionary robotics. In *GECCO*, pages 297–298.

Trujillo, L., Olague, G., Lutton, E., and De Vega, F. F. (2008b). Discovering several robot behaviors through speciation. In *Proceedings of the 2008 conference on Applications of evolutionary computing*, Evo'08, pages 164–174. Springer-Verlag.

Trujillo, L., Olague, G., Lutton, E., de Vega, F. F., Dozal, L., and Clemente, E. (2011b). Speciation in behavioral space for evolutionary robotics. *Journal of Intelligent and Robotic Systems*, 64(3-4):323–351.

Trujillo, L., Olague, G., Lutton, E., and Fernández de Vega, F. (2008c). Multiobjective design of operators that detect points of interest in images. In Cattolico, M., editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), Atlanta, GA, July 12-16*, pages 1299–1306, New York, NY, USA. ACM.

Trujillo, L., Silva, S., Legrand, P., and Vanneschi, L. (2011c). An empirical study of functional complexity as an indicator of overfitting in genetic programming. In Silva, S., Foster, J. A., Nicolau, M., Machado, P., and Giacobini, M., editors, *EuroGP*, volume 6621 of *Lecture Notes in Computer Science*, pages 262–273. Springer.

Trujillo, L., Spector, L., Naredo, E., and Martínez, Y. (2013b). A behavior-based analysis of modal problems. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation Companion*, GECCO Companion '13, pages 1047–1054.

Urbano, P. and Loukas, G. (2013). Improving grammatical evolution in santa fe trail using novelty search. In *Advances in Artificial Life, ECAL*, pages 917–924.

Urbano, P., Naredo, E., and Trujillo, L. (2014a). Generalization in maze navigation using grammatical evolution and novelty search. In *Theory and Practice of Natural Computing*, volume 8890 of *Lecture Notes in Computer Science*, pages 35–46. Springer International Publishing.

Urbano, P., Naredo, E., and Trujillo, L. (2014b). Generalization in maze navigation using grammatical evolution and novelty search. In Dediu, A.-H., Lozano, M., and Martín-Vide, C., editors, *Theory and Practice of Natural Computing*, volume 8890 of *Lecture Notes in Computer Science*, pages 35–46. Springer International Publishing.

Uy, N. Q., Hien, N. T., Hoai, N. X., and O'Neill, M. (2010). Improving the generalisation ability of genetic programming with semantic similarity based crossover. In *Proceedings of the 13th European Conference on Genetic Programming*, EuroGP'10, pages 184–195, Berlin, Heidelberg. Springer-Verlag.

Uy, N. Q., Hoai, N. X., O'Neill, M., Mckay, R. I., and Galván-López, E. (2011a). Semantically-based crossover in genetic programming: application to real-valued symbolic regression. *Genetic Programming and Evolvable Machines*, 12(2):91–119.

Uy, N. Q., Hoai, N. X., O'Neill, M., Mckay, R. I., and Galván-López, E. (2011b). Semantically-based crossover in genetic programming: application to real-valued symbolic regression. *Genetic Programming and Evolvable Machines*, 12(2):91–119.

Vanneschi, L., Castelli, M., and Silva, S. (2010). Measuring bloat, overfitting and functional complexity in genetic programming. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, GECCO '10, pages 877–884, New York, NY, USA. ACM.

Velez, R. and Clune, J. (2014). Novelty search creates robots with general skills for exploration. In *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation*, GECCO '14, pages 737–744. ACM.

Vellasco, M. M. B., Zebulum, R. S., and Pacheco, M. A. (2001). *Evolutionary Electronics: Automatic Design of Electronic Circuits and Systems by Genetic Algorithms*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition.

Weinberger, E. (1990). Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics*, 63(5):325–336.

Weise, T., Wan, M., Wang, P., Tang, K., Devert, A., and Yao, X. (2014). Frequency fitness assignment. *Evolutionary Computation, IEEE Transactions on*, 18(2):226–243.

Whitley, L. D. (1991). Fundamental principles of deception in genetic search. In *Foundations of Genetic Algorithms*, pages 221–241. Morgan Kaufmann.

Wilensky, U. (1999). Netlogo, Evanston, IL: Center for Connected Learning and Computer-Based Modeling. https://ccl.northwestern.edu/netlogo/. Accessed: 09-Feb-2016.

Woolley, B. G. and Stanley, K. O. (2012). Exploring promising stepping stones by combining novelty search with interactive evolution. *CoRR*, abs/1207.6682.

Yang, S. (2004). Adaptive group mutation for tackling deception in genetic search. *WSEAS Transactions on Systems*, 3(1):107–112.

Zhang, M. and Smart, W. (2006). Using gaussian distribution to construct fitness functions in genetic programming for multiclass object classification. *Pattern Recogn. Lett.*, 27(11):1266–1274.