Hamilton Institute

Maynooth University
National University of Ireland Maynooth

Semantic-Based Surrogate-Assisted Neuroevolution for Neural Architecture Search in Deep Neural Networks

Fergal Stapleton

Supervisor: Edgar Galván-López

A thesis submitted in fulfillment of the requirements for the Ph.D. degree in Computer Science

Director of Institute: Andrew Parnell
Hamilton Institute
Maynooth University
Maynooth, Co. Kildare, Ireland

April 11, 2025

Summary

Neuroevolution is a popular branch of Neural Architecture Search (NAS) that searches for high-performing artificial neural network architectures using evolutionary algorithms. Neuroevolution of deeper, more complex architectures, like deep neural networks, however, comes at a great computational cost, as often thousands of architectures need to be trained and evaluated over numerous Graphical Processing Unit days. To address this, research has turned to the use of Surrogate-Assisted Evolutionary Algorithms (SAEAs), where less expensive surrogate models can be used to estimate the fitness of an architecture, without the need to fully train it, resulting in a substantial reduction in the associated computational cost. Ultimately, SAEAs have emerged as a graceful response to tackling computational intensive workflows, such as neuroevolution, however, some notable limitations remain, such as, issues relating to high-dimensionality and complex encoding strategies required in current surrogate-assisted neuroevolution methods.

In this thesis, we use a semantic-inspired method to adeptly handle these issues, which in turn, is incorporated into a novel technique named Neuro-Linear Genetic Programming (NeuroLGP). NeuroLGP evolves chain-structured topologies with a representation closely aligned to how neural network architectures are naturally constructed. This allows us to perform an in-depth analysis not only on the surrogate model robustness and architecture performance, but also allows us to analyse how the internal makeup of our architectures change during evolution. From this, we propose a new mechanism, named NeuroLGP-MB, that is capable of evolving truly complex modern networks that exhibit multi-branch connections. Our proposed SAEA approach was shown to not only be robust for both NeuroLGP and NeuroLGP-MB but was also able to find high-performing individuals with a substantial reduction in time.

Acknowledgements

I would like to thank my supervisor Edgar Galván who has given me incredible support and guidance throughout my PhD and has helped me outline a path for a future career in academia, I am truly grateful. It has been an awesome journey and I am looking forward to our future collaborations.

I would like to thank my family for their support throughout my PhD, my parents Noel and Eileen, brothers Mark and Alan, and my sister Anna for her endless encouragement, with a special mention to the curious feline enigma that I refer to as "The Mister'. I would also like to thank my uncle Lenny, who helped foster my pursuit of knowledge from an early age, whether it be art, film or literature (...the finest of which can be found in the 'Library'!). My friends Sean Hall (Rasher), Steven Bellew and Ronan Larkin. It's been a while since I've been back to Dundalk but I am looking forward to the next gig or night out, or for any event that Rasher refers to as 'aesthetic excellence'. I would like to thank my housemates Eleni, Darshana and Alessandra, who were also a massive support, especially in the final weeks before submitting my thesis. I would like to thank the other members of the two households and friends: Akash and Neisha, Blake and Orla (congrats, again!), YC, Johnny, Jason and Ahmed. Ye are all legends and it has been awesome getting to know you.

I would also like to acknowledge the members of our research team, Pramit, Harish, Fred and Dan, as well as our collaborators Leonardo Trujillo and Brendan Cody-Kenny, your support has been incredible. I would also like to acknowledge everyone in the Hamilton and SFI: Rosemary, Kate, Joanna, Janet, David Malone and Ken Duffy, who were a massive help through-out. The rest of my friends and colleagues from the 2020 cohort at Maynooth: Shauna, Chang, Nathan, Bill, Tzirath, Jonny and Amit. Everyone from the various cohorts across MU, UL and UCD, both past and present, who I have talked to and who offered support and of whom it would be to numerous to list. I would also like to offer thanks to Ganesh and Senthil from Valeo, who mentored me during my industrial placement.

Funding and Resources

- This publication has emanated from research conducted with the financial support of Science Foundation Ireland under Grant number 18/CRT/6049. For the purpose of Open Access, the author has applied a CC BY public copyright licence to any Author Accepted. Manuscript version arising from this submission.
- The authors wish to acknowledge the Irish Centre for High-End Computing (ICHEC) for the provision of computational facilities and support.
- The simulations in Chapter 6 were performed on the Luxembourg national supercomputer MeluXina. The authors gratefully acknowledge the LuxProvide teams for their expert support.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Deep learning has had a transformative effect on society in the last number of years, from the application of deep learning in large language models like ChatGPT [14], to its application in autonomous vehicles [15] and for diagnostics in the health care sector [16], to name just a few. Deep Neural Networks (DNNs) [17] are a specific type of neural network used in deep learning, characterised by multiple layers that enable them to learn complex patterns in the underlying input data.

Traditionally, DNNs required either some specialised expert knowledge in their construction or random search approaches, however, attention has now turned to Neural Architecture Search (NAS) as a means to find high-performing networks. There are a number of different machine learning approaches for conducting NAS, such as one-shot models [18] and reinforcement learning [19], but for this thesis we focus exclusively on neuroevolution, a technique which uses Evolutionary Algorithms (EAs) [20] to evolve DNN architectures. In this work we predominately evolve Convolutional Neural Networks (CNNs), which are often used to process structured grid data such as image data, and are often used for tasks such as image classification or object detection, though we look exclusively at image classification. CNNs apply kernels to extract important feature information, like edges, textures, and patterns based on the input image data. For image classification, a fully connected layer is then used to make a classification decision on the extracted features and a final softmax layer can then be used to produce probabilities for different classes.

One of the major challenges of neuroevolution is the extraordinary cost associated with training networks for evaluation. For instance, with very large models, evaluating even a single epoch comes with a considerable cost [21]. To this end, attention has turned to using surrogate-assisted evolutionary algorithms (SAEAS) [22]. SAEAs can be used to estimate the performance of neural network architecture, without the need to fully train

every network, by substituting the traditional expensive model with a more cost-effective technique. In particular, interpolation-based surrogate modelling techniques have shown much promise [23] [24].

Performance prediction of DNNs however often use genotypic information, i.e., information on the physical characteristics of the network such as the layers, parameters and so forth, to encode information to use with distance-based interpolation surrogate approaches [25]. A drawback, however, is that for complex topologies, such as multi-branch networks, the genotypes of each network may differ by having varying dimension sizes and as such require clever encoding strategies to compare genotypes, such as using graph-edit distance [26], though such methods may be sub-optimal [23] [24]. Using phenotypic information, however, has shown some promise [23] [27], although to date they have not been applied to large networks like DNNs and issues around the dimensional limitations of using phenotypic distances for interpolation remain.

Aligned to this notion of phenotypic distance is the use of semantic-based distance metrics in Genetic Programming (GP) [28], which use the behaviour of a GP program in order to create distance metrics. Likewise, using our knowledge of semantics, we designed a surrogate-assisted evolutionary approach using a Linear Genetic Programming (LGP) [10] approach to find a more cost-effective approach for neuroevolution. This method, referred to as Neuro-Linear Genetic Programming (NeuroLGP) is capable of evolving variable-length, chain-structured architectures. Using NeuroLGP, we performed an in-depth analysis on the robustness of the surrogate model as well as the cost savings in terms of time and energy, while addressing the aforementioned dimensional limitations. Next, we scaled the topological complexity of possible networks by further enhancing the encoding of our LGP approach. This new method, entitled NeuroLGP Multi-Branch (NeuroLGP-MB), allowed our neuroevolution technique to search for architectures with more modern characteristics such as multiple branches [29] and skip-connection [30]. Again, we performed an in-depth analysis of this novel approach.

1.2 Research Goals

The main research goals for this thesis are as follows:

(1) Demonstrate the validity and adaptability of using semantic-based distances in intrinsically different EA workflows, using multi-objective (MO) optimisation as a test case, for their later use in SAEAs (2).

(2) Develop a robust and effective surrogate-assisted model management strategy centred around phenotypic distance vectors, based on our knowledge of semantic-based

distance metrics in (1), demonstrating significant time and energy saving when considering this surrogate approach.

(3) Address current dimensionality limitations of using traditional Kriging by considering for the first time Kriging Partial Least Squares [31] for neuroevolution, in conjunction with (2).

(4) Develop a novel neuroevolution approach based on LGP, demonstrating that this method is capable of creating architectures that are competitive with state-of-the-art hand-crafted architectures.

(5) Demonstrate that our approach is still robust when our architecture is scaled from a chain structure topology to a more complex multi-branch topology.

1.3 Thesis Structure

Chapter 2 details the background and general methodologies of this work. First, we detail the evolutionary process of Evolutionary Algorithms and provide details on GP focusing on Tree-based Genetic Programming, LGP and then provide a formal definition of Semantics with an example of semantics in GP. Next, we detail surrogate-assisted evolutionary algorithms focusing on Kriging and its variant Kriging Partial Least Squares (KPLS) before defining the phenotypic distance vector used in this work. Next, we outline the Multi-Objective (MO) frameworks used exclusively in Chapter 4, namely, detailing in full how these algorithms work. Finally, we discuss DNNs, detailing the most important concept around CNN architectures before discussing neuroevolution and the two main architecture types in terms of their topology.

Chapter 3 details the relevant literature for this work. First, a brief, independent history is given of EAs and artificial neural networks, highlighting major works from the last 70+ years. Next, we discuss NAS focusing on three of the major areas in this field. Focusing explicitly then on neuroevolution, we look at major publications relating to neuroevolution of traditional artificial neural networks and then neuroevolution of DNNs. We then discuss notable works relating to surrogate-assisted approaches for neuroevolution, before finally, discussing works on semantics in GP and the closely aligned concept of phenotypic distance, detailing some relevant publications as they relate to surrogate-assisted neuroevolution.

Chapter 4 provides a study on semantics in GP from a unique perspective, focusing on an indirect semantic method centred on semantic-based distances, with the ultimate goal of incorporating it into SAEAs for neuroevolution, as shown in Chapters 5 and 6. To do so,

we use MO optimisation as a test case, which uses intrinsically different EA frameworks and, as such, highlights the adaptability and versatility of considering semantics-based approaches on a whole. The semantic-based distance metrics we use help promote diversity of solutions, which in turn help to improve evolutionary search. We provide a detailed discussion at the end of this chapter on the applicability of semantic-based approaches for SAEAs in neuroevolution.

Chapter 5 introduces the original NeuroLGP approach. Here we provide a further motivation for why we are considering linear genetic programming and motivation for considering KPLS. After detailing the NeuroLGP approach and the surrogate model management strategy, we analyse our results from three experimental models: the baseline model, which constructs network layers in a random order but maintains a repair mechanism to ensure validity; the expensive model, which is the NeuroLGP approach where every network is fully trained; and the surrogate model, which is the surrogate model variant NeuroLGP-SM. We then perform an in-depth analysis of the surrogate model performance, as well as an analysis of the genotype. Furthermore, we detail some limitations of our approach.

Chapter 6 introduces the NeuroLGP-MB approach. First, we detail the encoding updates that have been developed in order to integrate multi-branch connections for the NeuroLGP-MB approach followed by a discussion on genetic operations that have been developed to specifically address validity issues for networks that have negative dimension or memory issues during compilation. We then detail a new surrogate variant that uses a pre-selection design-of-experiments approach to select individuals from a larger pool of individuals during initialisation. In the results section, we perform a similar analysis to Chapter 5, with a focus on the new surrogate model variant, demonstrating that the SAEA is robust and produces high-performing networks. We then analyse network depth and topological complexity, before discussing the potential for further computational reduction by varying the number of initial epochs we use to train our surrogate model.

Chapter 7 details the original contributions of the thesis while drawing some conclusions on the use of SAEAs in neuroevolution for NAS in DNNs. Finally, we conclude with a discussion on some of the limitations while also discussing future work.

1.4 List of Publications

1.4.1 Peer-reviewed Publications Pertinent to this Thesis

Journal Articles

[1] Edgar Galván, Leonardo Trujillo, and Fergal Stapleton. Semantics in multi-objective genetic programming. *Applied Soft Computing*, 115:108143, 2022

Conference Papers

[2] Edgar Galván and Fergal Stapleton. Semantic-based distance approaches in multi-objective genetic programming. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 149–156. IEEE, 2020

[3] Fergal Stapleton and Edgar Galván. Semantic neighborhood ordering in multi-objective genetic programming based on decomposition. In *2021 IEEE Congress on Evolutionary Computation (CEC)*, pages 580–587. IEEE, 2021

[4] Edgar Galván, Leonardo Trujillo, and Fergal Stapleton. Highlights of semantics in multi-objective genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 19–20, 2022

[5] Fergal Stapleton and Edgar Galván. Initial steps towards tackling high-dimensional surrogate modeling for neuroevolution using kriging partial least squares. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, GECCO '23 Companion, page 83–84, New York, NY, USA, 2023. Association for Computing Machinery

[6] Fergal Stapleton, Brendan Cody-Kenny, and Edgar Galván. Neurolgp-sm: A surrogate-assisted neuroevolution approach using linear genetic programming. In *International Conference on Optimization and Learning (OLA)*, 2024

[7] Fergal Stapleton and Edgar Galván. Neurolgp-sm: Scalable surrogate-assisted neuroevolution for deep neural networks. In *2024 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, 2024

1.4.2 Peer-reviewed Publications Not Pertinent to this Thesis

Journal Articles

[8] Edgar Galván and Fergal Stapleton. Evolutionary multi-objective optimisation in neurotrajectory prediction. *Applied Soft Computing*, 146:110693, 2023

Conference Papers

[9] Fergal Stapleton, Edgar Galván, Ganesh Sistu, and Senthil Yogamani. Neuroevolutionary multi-objective approaches to trajectory prediction in autonomous vehicles. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '22, page 675–678, New York, NY, USA, 2022. Association for Computing Machinery

2
Background

In Chapter 2, we will cover the background information and core concepts that serve as a foundation for later chapters. First, we give a general overview of Evolutionary Algorithms (EAs), discussing some of the main paradigms within this field, as well as briefly covering the main steps of an evolutionary algorithm (Chapter 2.1). Next, we delve into a specific branch of EA known as Genetic Programming (GP) (Chapter 2.2), an approach that seeks to evolve program trees to find candidate solutions for a given problem. We also discuss Linear Genetic Programming (LGP), where programs are represented as a linear sequence of instructions. Following on, we discuss Surrogate-Assisted Evolutionary Algorithms (SAEAs), which are evolutionary-based approaches that make use of surrogate models, which are cost-effective estimators for some more expensive optimisation function (Chapter 2.3). Finally, we discuss Multi-Objective Optimisation (MOO) and then Deep Neural Networks (DNN) (Chapters 2.4 and 2.5, respectively). The aim of MOO is to find candidate solutions that maximise the fitness of more than one objective. DNNs are an extension of artificial neural networks that use architectures that are capable of learning more complex representations compared to more traditional artificial neural networks, in particular, we discuss Convolutional Neural Networks (CNNs) that are often adopted in image classification tasks.

2.1 Evolutionary Algorithms

EAs are a branch of search heuristics inspired by biological reproduction and are well-established problem-solving approaches that are adept at handling problems with challenging features, such as discontinuities in the fitness landscape, multiple local optima, non-linear interactions between variables, to name a few [20]. Furthermore, the popularity of EAs can be attributed to their robust nature, flexibility, being assumption-free and capable of focussing on more than one solution [32]. Broadly speaking, there are four main groups which EAs can fall under; Genetic Algorithms [33], Evolutionary Strategies [34],

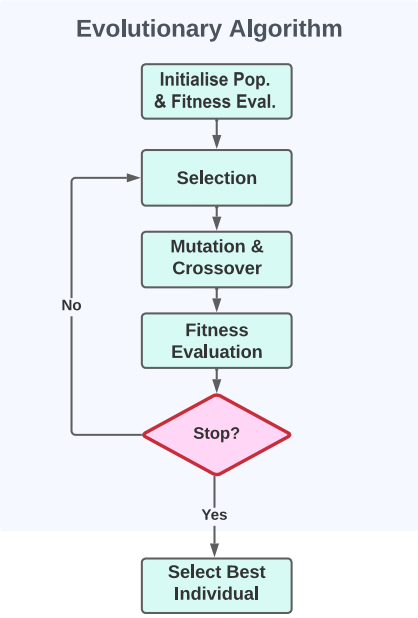


Figure 2.1: Demonstrating the workflow of a typical EA. See text for details on each step.

Evolutionary Programming [35] and Genetic Programming [36]. However, the lines of separability between these approaches have become less defined in recent years, owing to the flexibility in how these approaches are constructed. Moreover, EAs are highly adaptable with other machine-learning (ML) approaches and have helped improve the design and development of ML techniques, for instance, EAs can be used in hyper-parameter optimisation. More recently, EAs have been widely used in neural architecture search for Artificial Neural Networks (ANNs) [37], finding optimal architectures and hyperparameters in a field commonly referred to as neuroevolution. EAs have also been used in the training of DNNs, but to a lesser degree.

Typically, the evolutionary process, as demonstrated in Fig. 2.1, for an EA works as follows: i) a population of randomly generated candidate solutions are initialised and their fitness evaluated, ii) a selection process determines which individuals will be genetically modified, iii) genetic operations are performed, changing the internal structure of the candidate solutions, iv) a new population is generated by, first, taking a portion of the population from before genetic operations were applied and, second, the newly generated individuals, which in turn will have their fitness evaluated, and v) the process is repeated until some convergence criteria is met or the maximum number of generations have occurred. The above description, particularly step iv), depends on whether the methodology is steady-state or generational. In a steady-state approach technically there are no generations, and genetic operations are performed on only one or two individuals per iteration, whereas, in the case of a generational approach a large selection of the

population is selected for genetic operations. In the case of steady-state an appropriate replacement strategy must be employed to ensure convergence. In this work, we focus exclusively on generational approaches.

Another key component of EAs is the genetic representation. There are two forms of representation, the genotypic representation, which is some encoding of the problem solution and the phenotypic representation which is a representation of the observed solution itself. During the evolutionary process, the genotype is what will be altered by our EA. In the context of neuroevolution, for instance, an EA might have a genotype consisting of a bitstring, a program tree or a graph, but the phenotype will be the observed solution, which in this case is the resulting neural network.

2.2 Genetic Programming

2.2.1 Tree-based Genetic Programming

Genetic programming (GP), popularised by Koza in the early 90s [36], is one of the major paradigms of EAs [20], a branch of biological-inspired machine learning algorithms that mimic evolution to solve optimisation problems. GP, in particular, evolves program structures with the aim of finding candidate solutions to domain-specific tasks such as classification or symbolic regression problems. Like many EAs, traditional tree-based GP takes a population of initially randomised programs that are evolved over a set number of generations via genetic operations known as selection, crossover and mutation, to find candidate solutions to a pre-defined problem, where the ultimate goal is to find the best-performing program for a given solution.

The program structure of tree-based GP consists of branching leafs and nodes, where nodes may represent operators or functions and leafs represent variables or constants. The possible set of instructions or values available to leafs and nodes are referred to as the function set and terminal set, respectively. For instance, in a symbolic regression problem $\mathcal{F} = \{+, -, *, \%\}$ may represent the function set and $\mathcal{T} = \{x, -1, 0, 1, 2\}$ may represent the terminal set. For a classification problem, the terminal set will also include the features of a dataset, which is of particular relevance to our work on imbalanced datasets as discussed in Chapter 4. Next, we discuss the steps involved in GP.

Initialise population

To start, we must first create an initial population of programs which we will work on. Generally, programs are initialised to have random structures with some control over the depth that the initial program tree has. Some initialisation techniques that control the

depth of tree-like individuals are full, grow and ramped half-and-half [36]. If we define the depth of a tree as the depth of its deepest leaf, then the full method initialises the programs so that their leaves are all the same depth. The grow method on the other hand terminates once terminals are filled for a subtree that has reached the maximum depth. The ramped half-and-half method is a compromise between these two methods where half the population is created using the full method and the other half with grow.

Fitness Evaluation

Fitness evaluation determines the fitness of any particular program within the population. For example, in a binary classification problem, fitness evaluation is an indication of how accurate that program is at correctly classifying a particular class. In a symbolic regression problem, it is generally some measure of the difference between the candidate program and the true program, given a set of test cases. For more details on particular problem domains that can be explored using GP, see [36].

Selection

Selection is used to determine which individuals we wish to perform genetic operations on. A common approach is to use Tournament Selection [20]. This approach works by randomly selecting a small subset of the population and selecting one or two of the best-performing individuals in terms of fitness, which in turn we will use to perform our genetic operations on. Carefully selecting this subset size helps balance the diversity of individuals while also helping to converge on better solutions. Many other selection operators have been proposed in specialised literature such as wheel selection and rank selection [20], but are not used in this thesis.

Crossover and Mutation

There are two main forms of genetic operation that alter the genetic structure of our program, namely crossover and mutation. Conceptually, crossover is analogous to biological sexual reproduction where the genetic material of an offspring is a composite of its parents' genetic material. Crossover is responsible for exploiting the search space while mutation explores it. For example, with subtree crossover, a node (referred to as the crossover point) for each copied parent program is randomly selected, then the child programs are created by first removing the subtrees of each parent below the crossover point, and then swapping the subtrees at this point to effectively create two offspring. This is demonstrated in Fig. 2.2.

Mutation operates on a single parent to produce offspring and as such no genetic material is shared within the population. Mutation may occur on a single randomly

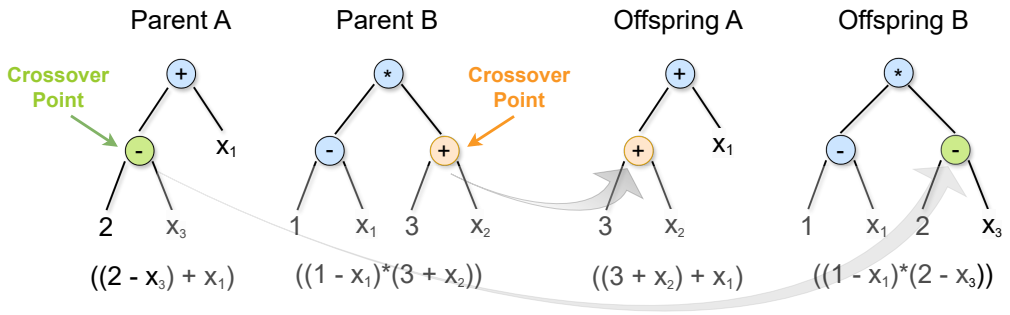


Figure 2.2: Demonstrating how crossover operation creates two offspring programs through subtree crossover.

selected leaf or node, as in the case of single-point mutation. Here if a node is selected a primitive is swapped with a primitive of the same arity. Another example is subtree mutation, where a node is selected and the subtree branching from that node is randomly replaced with a new subtree.

Elitism

The above steps will loop until a set number of generations have been reached or until some convergence criterion has been met (red diamond in Fig. 2.1). At each generation, an elitism mechanism ensures that the best-performing individuals are retained into the next generation. This ensures we are continually updating the population with better-performing or equally good individuals. In the final generation, the best individual or program can then be selected from the final generation and used to report the best solution found so far.

2.2.2 Linear Genetic Programming

A variation of traditional tree-like GP is Linear Genetic Programming (LGP) [10]. Unlike tree-based GP where the encoding encapsulates branching operations within the program structure, LGP instead encodes a linear sequence of program instructions similar to how assembly languages operate. As part of this encoding, LGP makes use of registers, which are a type of computer memory for storing or manipulating data, while executing instructions, in a low-level programming context. The content of these registers are altered using instruction operations.

There are three main components to instruction; an *operand* which performs a specific function on one or more *registers* which store the result in a *destination register*. In the case of 2-register instruction encoding, the operand operates on a single instruction

```
void gp(r)
double r[8];
{ ...
   r[0] = r[5] + 71;
   // r[7] = r[0] - 59;
   if (r[1] > 0)
   if (r[5] > 2)
      r[4] = r[2] * r[1];
   // r[2] = r[5] + r[4];
   r[6] = r[4] * 13;
   r[1] = r[3] / 2;
   // if (r[0] > r[1])
      // r[3] = r[5] * r[5];
   r[7] = r[6] - 2;
   // r[5] = r[7] + 15;
   if (r[1] <= r[6])
      r[0] = sin(r[7]);
}
```

Figure 2.3: LGP example in C language. Reproduced from Brameier and Banzhaf's book 'Linear Genetic Programming' [10]. The original representation uses registers to control the flow of data between a series of instructions which are executed sequentially.

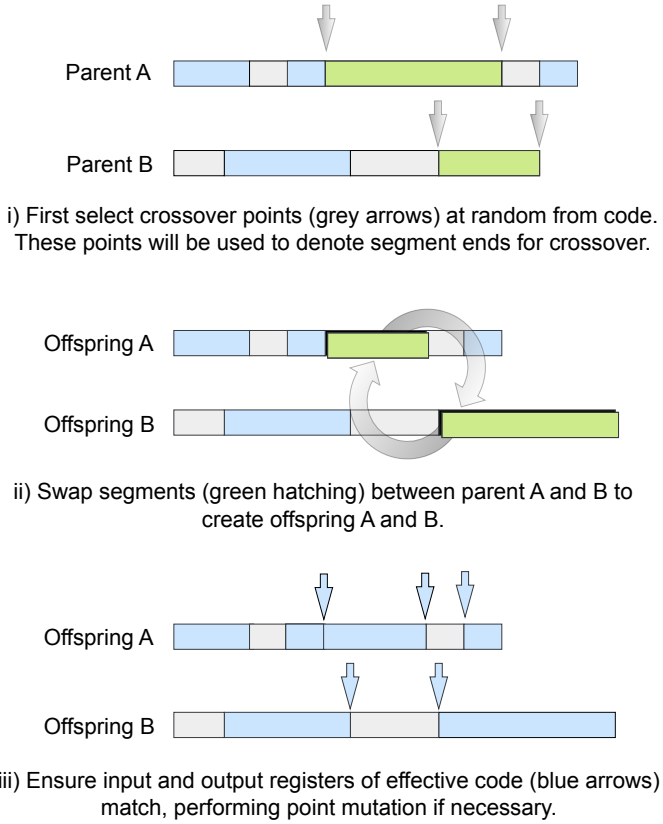


Figure 2.4: Demonstrating how crossover operation works in Linear Genetic Programming.

and for a 3-register instruction encoding operates on two instructions (i.e., consider an add function that adds two numbers, where each number is stored in its own register with an additional destination register). Fig. 2.3 highlights an example of LGP written in C code, where `r[i]` denotes the i^{th} register. This example contains both effective and non-effective lines of code, where effective lines of code represent lines that will be compiled and executed and the non-effective lines of code are commented out and subsequently not compiled. Each line of code is executed imperatively. The register `r[0]` is a specially designated register for the final output of the program. In the case of conditional statements, like the IF statement, if the condition has not been met then the code within the IF statement can be treated as non-effective.

Many of the concepts outlined in Chapter 2.2.1 can still be used with this type of representation (LGP)[1], albeit now genetic operations in LGP work by either modifying these registers or the instructions that operate on them. In Fig. 2.4, we demonstrate how crossover works within the context of LGP. The representation denotes a sequence of effective code (light blue) and non-effective code (light grey). In our work, we perform crossover specifically on effective code only and the segments that are swapped (green hatching) use point mutation to ensure resulting offspring are executable by linking them to corresponding effective sections of the code base. We can see from Fig. 2.2, that both the representation and genetic operations contrast significantly between LGP and tree-based GP. There are a number of properties that make LGP appealing for neuroevolution, such as the interpretability of solutions or the ease at which to employ crossover and mutation, but for a full discussion, please see Chapter 5.2.

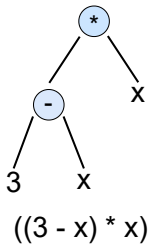
2.2.3 Semantics in Genetic Programming

Semantics can be seen as the behaviour of a GP program [1, 38]. This behaviour is the output of a GP program when executed on a set of fitness cases. Fig. 2.5 demonstrates this concept. We can see that for a number of fitness cases x, as shown in the table on the right, we get an output vector that represents the semantics for the program tree as shown on the left. Following [39], let $p \in P$ be a program from a given programming language P. The program p will produce a specific output $p(in)$ where input $in \in I$. The set of inputs I can be understood as being mapped to the set of outputs O which can be defined as $p : I \to O$.

Def 1. *Semantic mapping function is a function $s : P \to S$ mapping any program p from P to its semantics $s(p)$, where we can show the semantic equivalence of two programs*;

$$s(p_1) = s(p_2) \iff \forall\ in \in I : p_1(in) = p_2(in) \tag{2.1}$$

[1]The original LGP work [10] uses a *steady-state* methodology but in this work, we *use generational-based* methodologies exclusively.



Input	output
0.0	0.00
0.5	1.25
1.0	2.00
1.5	2.25
2.0	2.00

Semantics = [0.00, 1.25, 2.00, 2.25, 2.00]

Figure 2.5: Demonstrating an example of semantics in GP. Given a set of inputs (or fitness cases) for x, semantics can be shown as the vector output of the GP program.

The definition, as presented in Eq. 2.1, produces three important and intuitive properties of semantics:

(1) Every program has only one semantics attributed to it.

(2) Two or more programs may have the same semantics.

(3) Programs that produce different outputs have different semantics.

In Def. 1, we have not given a formal representation of semantics. In what follows, semantics will be represented as a vector of output values which are executed by the program under consideration using an input set of data. For this representation of semantics, we need to define semantics under the assumption of a finite set of fitness cases, where a fitness case is a pair comprised of a program input and its respective program output $I \times O$. This allows us to define the semantics of a program as follows:

Def 2. *The semantics $s(p)$ of a program p is the vector of values from the output set O obtained by computing p on all inputs from the input set I:*

$$s(p) = [p(in_1), p(in_2), ..., p(in_l)] \tag{2.2}$$

where $l = |I|$ is the size of the input set.

Now that we have a formal definition of semantics within the context of GP we can discuss how it relates to phenotypic distance vectors, which in turn can be used in surrogate modelling and neuroevolution in whole, as discussed in Chapter 2.3.

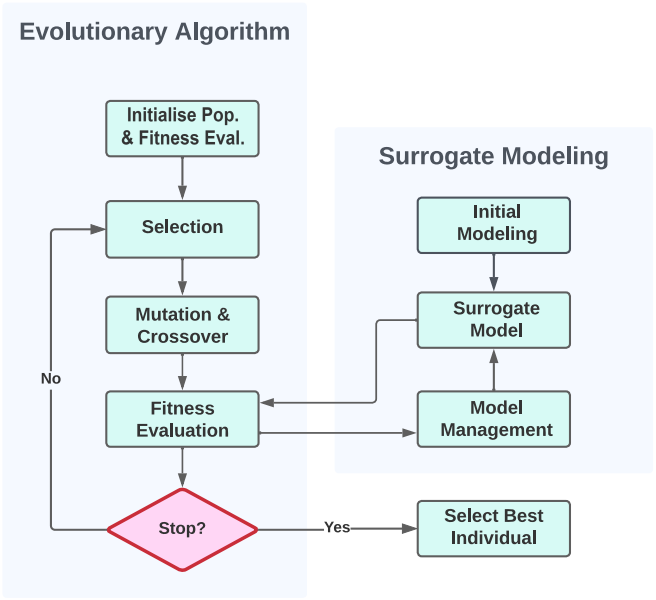


Figure 2.6: Demonstrating the workflow of a typical evolutionary algorithm and the interaction with a surrogate model.

2.3 Surrogate-assisted Evolutionary Algorithm

2.3.1 Surrogate Model-based Optimisation

In this work, the aim of a surrogate model [22], also referred to as a meta-model, is to sufficiently approximate an estimate of the fitness values of a potential solution, while reducing the run time of the evolutionary process, compared to the run time of using the real fitness function alone. This requires that the surrogate model is well-posed and requires a robust model management strategy, otherwise, the EA may converge to a false optimum [22]. The surrogate model management strategy is responsible for adequate sampling of data to ensure the surrogate model is well-informed, using model validation to ensure the surrogate model makes accurate predictions, and model updating to ensure new samples are continually supplied to refine the surrogate model during optimisation. Fig. 2.6 shows the interaction between a typical EA and a surrogate model. The steps are as follows: firstly, some initial modelling is performed based on randomly selected individuals or based on some design of experiment techniques; secondly, after the initial modelling, the surrogate model can be used to estimate the fitness of individuals within the EA; thirdly, we select individuals which we will choose to fully evaluate, based on an acquisition function, to better inform the accuracy of our surrogate model and lastly, we update the surrogate model based on the previously selected individuals as part of the overall model management strategy.

Typically, the surrogate model differs from the parent model, in that, instead of being trained directly on data relating to the problem domain, the surrogate model is trained off the design of the parameter space. Here, the goal is to identify and further explore regions of this design space that will produce preferable parameters. As such, interpolation-based approaches may be used to simulate the regions of the parameter space, such as Gaussian processes, commonly referred to as Kriging. Another benefit to the Kriging approach is that it allows for estimates of the uncertainty of predictions.

2.3.2 Kriging (Gaussian Processes)

Kriging is an interpolation-based technique that assumes spatial correlation exists between known data points, based on the distance, and variation between these points. We aim to use observations $\mathrm{y} = [y(\mathrm{x}^{(1)}), y(\mathrm{x}^{(2)}), ...y(\mathrm{x}^{(n)})]$ to help estimate an unknown function value $\hat{y}(\mathrm{x}^*)$ for the unknown data x^*, where n is the number of known data points in the training data and x is a $1 \times d$ dimensional vector. More specifically, $\mathrm{x}^{(j)} = [\mathrm{x}_1^{(j)}, \ldots, \mathrm{x}_d^{(j)}]$ is the j^{th} training point of a $1 \times d$ vector for $j = 1, \ldots, n$ training points, where X is the matrix $[\mathrm{x}^{(1)T}, \ldots, \mathrm{x}^{(n)T}]$ and is a collection of all training samples for the surrogate model, where T denotes the transpose.

A kernel function $k(\cdot)$ is used to express the spatial correlation between two samples x_i and $\mathrm{x}_i^{'}$ as shown in Eq. 2.3.

$$\begin{aligned} k(\mathrm{x}, \mathrm{x}') &= \sigma^2 \prod_{i=1}^{d} exp(-\theta_i(\mathcal{D}(\mathrm{x}_i, \mathrm{x}_i^{'}))) \\ &= \sigma^2 \prod_{i=1}^{d} exp(-\theta_i(\mathrm{x}_i - \mathrm{x}_i')^2) \end{aligned} \tag{2.3}$$

where the θ hyperparameters control the rate at which the correlation decays to zero between x_i and $\mathrm{x}_i^{'}$ and σ^2 is how much a process varies from expected performance, and $\mathcal{D}$ is the distance between x_i and $\mathrm{x}_i^{'}$. The θ hyperparameters are determined using the Maximum Likelihood (ML) estimator, this however, leads to a significant drawback when considering high-dimensional data, which is discussed in more detail in Chapter 2.3.3.

The θ hyperparameters are crucial in measuring the correlation between the sample points. While it is possible to use isotropic kernels, i.e., a kernel that uses a singular θ hyperparameter, an assumption needs to be made that each element or feature of the training points ought to have the same length scale. On the other hand, anisotropic kernels allow for feature-specific length scales. In other words, for an anisotropic kernel, the θ hyperparameters account for the varying relevance of features within each training point, or, more explicitly, θ_i determines the sensitivity of the kernel for the i-th feature

in the training points. In the context of a classification problem, our features relate to the probability value for a particular class.

From the covariance kernels, we can generate a best linear unbiased predictor for $\hat{y}(\mathrm{x}^*)$, given by the observations y, and is shown in Eq. 2.4,

$$\hat{y}(\mathrm{x}^*) = f(\mathrm{x}^*)^T\hat{\beta} + r_{\mathrm{x}^*X}^T R^{-1}(\mathrm{y} - F\hat{\beta}) \tag{2.4}$$

where $\mathrm{f}(\mathrm{x}^*) = [f_1(\mathrm{x}^*), \ldots, f_m(\mathrm{x}^*)]^T$ is an $1 \times \mathrm{m}$ vector of basis functions, $F = [\mathrm{f}(\mathrm{x}^{(1)}), \ldots, \mathrm{f}(\mathrm{x}^{(n)})]^T$ is an $n \times m$ matrix, β is the vector of generalised least-square estimates of $\beta = [\beta_1, \ldots, \beta_m]^T$, R is an $n \times n$ matrix of of the pairwise covariances of all training samples X such that $R = [\mathrm{r}_{\mathrm{x}^{(1)}X} \ldots \mathrm{r}_{\mathrm{x}^{(n)}X}]$. Notably, $\mathrm{r}_{\mathrm{x}^*X} = [r_{\mathrm{x}^*\mathrm{x}^{(1)}}, \ldots, r_{\mathrm{x}^*\mathrm{x}^{(n)}}]^T$ is a $1 \times n$ vector and can be understood as covariance vector between the new observation and all the training samples X. More generally, $r_{\mathrm{xx}'}$ is the correlation kernel between x and x' and relates back to the kernel function k as previously defined in Eq. 2.3, with Eq. 2.5

$$k(\mathrm{x}, \mathrm{x}') = \sigma^2 r(\mathrm{x}, \mathrm{x}') = r_{\mathrm{xx}'} \tag{2.5}$$

2.3.3 Kriging Partial Least Squares

The popularity of the Kriging method stems from its ability to accurately simulate computationally expensive processes while also giving an estimate of the predictive error, however, a major drawback of Kriging is that for high-dimensional data, the method itself becomes computationally expensive to perform. This is due, firstly, to the size of the covariance matrix becoming large as the number of sample points increases, which subsequently needs to be inverted. Secondly, to further compound the issue, the hyper-parameters θ need to be estimated, which requires the covariance matrix to be inverted several times. Eq. 2.6 shows the ML for θ

$$\begin{aligned} logML(\theta) = -\frac{1}{2}[n \ln \frac{1}{n}(\mathrm{y} - F(F^T R^{-1}(\theta)F)^{-1}F^T R^{-1}(\theta)\mathrm{y})^T \\ \times R^{-1}(\theta)(\mathrm{y} - F(F^T R^{-1}(\theta)F)^{-1}F^T R^{-1}(\theta)\mathrm{y})) + \ln detR(\theta)] \end{aligned} \tag{2.6}$$

For a more comprehensive derivation of the logML equation please refer to the [31]. The key takeaway, however, is that concerning Eq. 2.6, R^{-1} requires multiple inversions and has a cost of $\mathcal{O}(n^3)$ associated with it, where n is the number of sample points (individuals in the training data). Kriging Partial Least Squares (KPLS) seeks to address the issue of high-dimensionality by effectively reducing the number of parameters calculated [31]. It does so using Partial Least Squares (PLS) which projects the high-dimensional data into a lower dimension using principal components.

The PLS approach works by finding a linear relationship between input variables and output variables by projecting the input variable into a new space using principal components. More specifically, the principal components are effectively linear combinations of the input variables and represent the new coordinate system, obtained by rotating the original system with axes, $\mathrm{x}_1, \ldots, \mathrm{x}_d$. There are three main steps for the KPLS approach, i) PLS is used to define weight parameters w, ii) a new covariance kernel is created using the PLS weights that reduce the number of hyper-parameters and iii) the parameters are then optimised. Eq. 2.7 details the KPLS covariance kernel

$$k(x, x') = \sigma^2 \prod_{l=1}^{h} \prod_{i=1}^{d} exp(-\theta_l (w_i^{(l)} \mathrm{x}_i - w_i^{(l)} \mathrm{x}'_i)^2) \tag{2.7}$$

where w are rotated principal directions which maximise the covariance and are a measure of how important each principal component is. The number of principal components h is much less than m which allows for the substantial increase in performance associated with KPLS. For full details of the method, see [31].

2.3.4 Phenotypic Distance Vectors

The use of phenotypic information derived from neural networks, discussed in detail in Chapter 5, can be extremely beneficial in creating distance metrics for use in surrogate modelling [23, 27]. We will discuss in Chapter 4, and have already discussed in Chapter 2.2.3, the importance of semantics in Genetic Programming. In particular, we have learned how diversity can be promoted by carefully adopting semantics in GP. This has led to a considerable increase in performance in GP [1]. We have demonstrated how semantic-based vectors can be successfully used in surrogate models to correctly estimate fitness values of individuals without explicitly evaluating them through the use of a fitness function. We will discuss this in detail in Chapters 5 and 6.

In the context of neuroevolution, we can define a solution sample x as having the semantic or phenotypic behaviour of the i^{th} program such that $\mathrm{x}_i = s(p_i)$, where the semantics $s(p)$ of a program p is the vector of values from the output. From Eq. 2.3, we can now then define our distance $\mathcal{D}$ as a phenotypic distance, represented in Eq. 2.8 as

$$\mathcal{D}(\mathrm{x}_i, \mathrm{x}_j) = \mathcal{D}(s(p_i), s(p_j))) \tag{2.8}$$

where the distance metric is dependent on the outputs of each network. In this work, we use the convention established by Stork et al. [27], where the x is a flattened vector containing the output of the nodes at the final softmax layer for all data instances. As such, our phenotypic distance vector length is given as the number of images of the validation dataset $\times$ the number of classes. Further reduction of the semantic vectors

was not considered in this work, however, it should be noted that since the dense layer outputs two probability values the second value is implicitly known based on the first. As such, it should be possible to reduce the vector by half for a two-class problem and, if we generalise, by (n-1) $\times$ the number of fitness cases for n classes. Importantly, this approach can be extended to any deep learning model architecture that can have its output represented in vectorised form, such as transformers, however, further research would be required to determine the limitations and scalability of applying this approach to other deep learning approaches.

2.4 Multi-objective Optimisation

2.4.1 Multi-objective Optimisation

In a multi-objective optimisation (MO) problem, one optimises with respect to multiple goals or objective functions. Thus, the task of the algorithm is to find acceptable solutions by considering all the criteria simultaneously. This can be achieved in various ways, where keeping the objectives separate is the most common. This form keeps the objectives separate and uses the notion of *Pareto dominance*. In this way, Evolutionary MO (EMO) [40–42] offers an elegant solution to the problem of optimising two or more conflicting objectives. The aim of EMO is to simultaneously evolve a set of the best tradeoff solutions along the objectives in a single run.

2.4.2 Pareto Dominance

In general terms, a multi-objective problem aims to find a solution that either maximises or minimises a number of objectives. In the case of maximisation, this can be represented mathematically as

$$max(f_1(x), f_2(x), ..., f_k(x)) \qquad s.t. \;\; x \in X, \tag{2.9}$$

where X represents the feasible solution set, $f_i(x)$ represents the i^{th} objective function for the feasible solution x and $k \geq 2$. In general, there is no single solution that will fully maximise all objective functions. A candidate solution is Pareto dominant if its fitness is better or equal for all objectives and is strictly preferred by at least one in the search space. This can be formally represented by

$$S_i \succ S_j \leftrightarrow \forall_m[(S_i)_m \geq (S_j)_m] \wedge \exists k[(S_i)_k > (S_j)_k] \tag{2.10}$$

where $(S_i)_m$ is the i^{th} solution for objective m and $S_i \succ S_j$ denotes that solution i is non-dominated by solution j. A candidate solution is considered Pareto optimal if is not

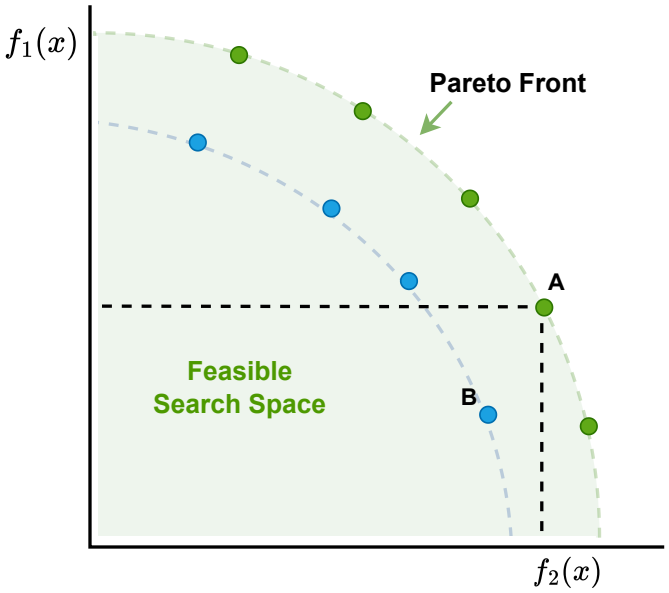


Figure 2.7: Demonstrating dominated (blue) and non-dominated solutions (green) with the aim of maximising objectives. Non-dominated solutions lie along the Pareto front and represent the best set of solutions found so far for a given MO problem.

dominated by any other candidate solution. In other words, if none of the objectives can be improved within the current set of candidate solutions without degrading at least one of the other objectives it can be considered Pareto optimal. For MO problems, there may exist a number of non-dominated solutions. The set of non-dominated candidate solutions for an MO problem is referred to as the first Pareto front when represented in objective space. Fig. 2.7 demonstrates the concept of dominated (blue dashed lines) and non-dominated (green dashed lines) solutions for a maximisation problem. We can see that solution B is dominated by solution A since A has a higher fitness for both $f_1(x)$ and $f_2(x)$ (black dashed line) and as such meets the condition set out in Eq. 2.10. In this example, the set of non-dominated solutions lie on the boundary of the feasible search space (green region). In other words, there exists no other solutions which will dominate this set and as such they are considered Pareto optimal.

In practice it is not always possible to do an exhaustive search for the true Pareto optimal set and as such this is something we seek to approximate instead. Pareto dominance relation is an integral part of MOEAs and has allowed practitioners and researchers to form important metrics in the selection process of these algorithms.

2.4.3 The Non-dominated Sorting Genetic Algorithm II

The Non-dominated Sorting Genetic Algorithm II (NSGA-II) [43] uses the Pareto dominance relationship as its primary mechanism for selecting solutions to be retained into future generations. It does so by sorting non-dominated solutions based on their dominance rank. More formally, the dominance rank $D(S_i)$ is determined by the number of individuals that dominate S_i in the population. In mathematical terms, this is represented in Eq. 2.11 as the cardinality of the set $|.|$ such that,

$$D(S_i) = |\{j | j \in Pop \wedge S_j \succ S_i\}| \tag{2.11}$$

In this sense, each individual may be assigned a rank and individuals of the same rank will belong to the same front F_D. For instance, individuals which have $D(S_i) = 0$ (i.e., are dominated by no other individual) will belong to front F_0, individuals which have $D(S_i) = 1$ will belong to the second front F_1 and so forth. As such, preferable Pareto fronts consist of lower-ranked individuals with the same rank. A new population is then generated from these fronts, where the new population size is the same size as the original parent population size.

Since the original population size is unlikely to be exactly filled by the last front, a secondary sort is required to select individuals from the last front and is based on the crowding distance operator. The crowding distance is the sum of Manhattan distances between the nearest neighbours of an individual under consideration [44]. Crowding distance is discussed in more detail in Chapter 2.4.5. NSGA-II is one of the most popular and still widely used EMO approaches.

2.4.4 The Strength Pareto Evolutionary Algorithm 2

Both Dominance rank and Dominance count are used in the Strength Pareto Evolutionary Algorithm 2 (SPEA-2) [45]. Dominance count calculates how many individuals a candidate solution dominates. The higher the dominance count the better.

$$D_{count}(S_i) = |\{j | j \in Pop \wedge S_i \succ S_j\}| \tag{2.12}$$

Leading on from the Eq. 2.12 we can get a measure of raw fitness $R(i)$, by summing the dominance count of all individuals as seen in Eq. 2.13;

$$R(i) = \sum_{j \in Pop,\ S_j \succ S_i} D_{count}(S_j) \tag{2.13}$$

An $R(i)$ value of 0 corresponds to an individual which is non-dominated and an $R(i)$ value that is high corresponds to an individual which is dominated by many other individuals,

as such the raw fitness should be minimised.

2.4.5 Crowding Distance

Solutions are ranked relative to each other according to a metric known as the crowding distance. The crowding distance is used to compare any pair of solutions in search space and is used in NSGA-II and SPEA2 as Pareto Dominance alone only acts as a partial order of the solutions. The crowding distance calculation is comprised of three parts;

- Initialise the distance d to zero.
- Set the boundary solutions to ∞. These solutions are always selected due to this constraint.
- Calculate the average distance differences for an individual against its two nearest neighbours using the Manhattan distance, shown in Eq. 2.14:

$$d = d + \frac{|f_{r+1}^{(k)} - f_{r-1}^{(k)}|}{|f_{max}^{(k)} - f_{min}^{(k)}|} \tag{2.14}$$

where k denotes the objective in question and r is the index for the current individual, where $r+1$ and $r-1$ reference its two nearest neighbours, assuming individuals are sorted in increasing order of their fitness for their respective objective. Solutions with the highest crowding distance are considered better solutions, in other words, the algorithm preferences localities along the Pareto front which are more sparsely populated with solutions than those which are more dense. In this manner the crowding distance resolves which solutions to retain when programs produce very similar fitness values.

2.4.6 Multi-Objective Evolutionary Algorithm with Decomposition

The Multi-Objective Evolutionary Algorithm with Decomposition (MOEA/D) [46] works by decomposing a multi-objective optimisation problem into a number of scalar optimisation sub-problems, where each sub-problem exploits a different region of the objective space via weights [46]. These scalar optimisation sub-problems are defined by the scalar optimisation function g, which under the canonical approach uses a grid-based distribution of weight vectors λ_j, as seen in Eq. 2.15, where $j = 1, 2...m$ and $\sum \lambda_j = 1$, such that

$$\lambda_j \in \left\{0, \frac{1}{H}, \frac{2}{H} ... \frac{H}{H}\right\} \quad for\ j = 1, 2...m \tag{2.15}$$

where H is an integer value determining the distribution of weights. In other words, for each objective, each element λ^j is one of $\{0, \frac{1}{H}, \frac{2}{H} ... \frac{H}{H}\}$ based on the parameter H. In the two-objective case, from Eq. 2.16 we have

$$\lambda_j^i = \left(\frac{i}{H}, 1 - \frac{i}{H} \right), \quad i = 0, 1, 2, ..., H \tag{2.16}$$

where i is an index of H and helps ensure our weight vectors are equally spaced. In the three-objective case, weight vectors are generated from a simplex lattice. The neighbour solutions are selected based on the Euclidean distance between the weights in terms of the objective space.

A core mechanism of the decompositional approach is the neighbourhood structure, where genetic operations and the selection process occur within neighbourhoods of closely related sub-problems. In this way, the neighbourhood structure controls the exploration and exploitation properties of the algorithm.

Canonical aggregation functions

In the original paper by Zhang et al. [46], three scalarisation decomposition methods were proposed: Weighted sum, Tchebycheff and Penalty-based Boundary Intersection (PBI), each with their corresponding scalar optimisation functions g^{ws}, g^{tch} and g^{pbi}, respectively. We briefly discuss each of these.

Weighted Sum
The scalar optimisation of g^{ws} is given by Eq. 2.17

$$\max(g^{ws}(x|\lambda)) = \sum_{j=1}^{m} \lambda_j f_j(x) \tag{2.17}$$

$$\text{subject to } x \in \Omega$$

where $j = \{1, 2...m\}$ is the dimensions of the objective space, f_i is the objective function, Ω is the decision (variable) space and x is the variable to be optimised.

Tchebycheff
The scalar optimisation of g^{tch} is given by Eq. 2.18

$$\min(g^{tch}(x|\lambda)) = \max_{1 \leq j \leq m} \{\lambda | f_j(x) - z_j|\} \tag{2.18}$$

The ideal point z_j represents the best objective function value for f_j found in the population thus far.

Penalty-based Boundary Intersection (PBI)
The scalar optimisation of g^{pbi} is given by Eq. 2.19

$$\min(g^{pbi}(x|\lambda)) = d_1 + \theta d_2 \tag{2.19}$$

where,

$$d_1 = \frac{||(f(x) - z)^T \lambda||}{||\lambda||},\ d_2 = ||(f(x) - (z - d_1 \frac{\lambda}{||\lambda||})|| \tag{2.20}$$

where θ is a predefined penalty parameter set by the user.

Algorithm

The steps involved in the canonical MOEA/D version are detailed in Alg. 1, as well as the procedure for the standard neighbourhood update approach in Alg. 2. The individual steps involved in these are given below in more detail below.

Step 1: Initialisation.

Step 1.1) Initialise external population $EP = \emptyset$.

Step 1.2) Calculate the Euclidean distance between any two weight vectors and find the T closest weight vectors to each respective weight vector. For each $i = \{1, 2, ..., N\}$, set $B(i) = \{i_1, i_2, ..., i_T\}$, where $B(i)$ can be understood as a neighbourhood reference table of indices and where $\lambda^{i_1}, \lambda^{i_2}, ..., \lambda^{i_T}$ are the T closest weight vectors to i.

Step 1.3) Randomly create the initial population $x_1, x_2 ... x_N$ and calculate initial fitness value $F(x_i)$, where $F(x)$ is the transpose of $(f(x_1), ..., f(x_N))$.

Step 2: Apply genetic operations and search for solutions.

Step 2.1) Select two indices k and l randomly from the neighbourhood reference table $B(i)$ and generate new offspring y from parents x_k and x_l by applying crossover and mutation genetic operations.

Step 2.2) Update z such that for each $j = \{1, 2, ..., m\}$ if $z_j < f_j(y)$, then set $z_j = f_j(y)$. In the case where the objective is to minimise $F(x)$, then this inequality is reversed.

Step 2.3) Update the neighbouring solutions for the j^{th} case such that $j \in B(i)$, if $g(y|\lambda^j, z) \geq g(x^j|\lambda^j, z)$, then let $x^j = y$ and calculate new fitness $F(y)$.

Algorithm 1 Cannonical MOEA/D

1: $\Lambda = \{\lambda^{i_1}, \lambda^{i_2}, ..., \lambda^{i_N}\}$ ▷ Create weight vectors
2: $P = \{x_1, x_2, ..., x_N\}$ ▷ Initialise population
3: **for each** $\lambda^i \in \Lambda$ **do**
4: $B(i) = \{i_1, i_2, ..., i_T\}$ ▷ Define reference table
5: **end for**
6: **repeat**
7: **for each** $i \in \{1, 2, ..., N\}$ **do**
8: $k, l \leftarrow$ Return random parent indices from B(i)
9: $y \leftarrow$ Generate child program from x^k and x^l
10: $P = Standard\ Update(y, i)$, which calls Alg. 2
11: **end for**
12: $EP \leftarrow$ Remove non-dominated solutions based on P
13: **until** Stopping criteria is met
14: **return** EP ▷ The non-dominated solutions from EP

Algorithm 2 Standard Update

1: **for each** $j \in B(i)$ **do**
2: **if** $g(y|\lambda^j, z) \geq g(x^j|\lambda^j, z)$ **then**
3: $x_j = y$ ▷ Allows for multiple replacements to occur
4: **end if**
5: **end for**

Step 3: Fill external population and termination.

Step 3.1) EP is filled with non-dominated solutions across all generations, where newly dominated solutions at each generation are removed. When the stopping criterion is satisfied, the non-dominated solutions of EP are outputted; otherwise, Step 2 is repeated until the stopping criteria is satisfied.

Solution search as determined by each of the decomposition methods controls which population members are updated. In the standard update (Step 2.3), each program from the current population is updated with the child program. Since each index j from the neighbourhood reference table is checked multiple times against the child program this might lead to multiple replacements within the specified neighbourhood. As such under the standard update, duplication of programs can occur at a generational level.

One way to alleviate this duplication is to break out of the loop after the first replacement has occurred. Such an approach would immediately reduce the amount of duplication occurring at each generation, though subsequently there is no guarantee that the most beneficial replacement would have occurred. To facilitate this pitfall, such an approach would need to order the neighbourhood prior to undergoing the solution search.

2.5 Deep Neural Networks

The origins of DNNs [17] stem from the study of Artificial Neural Networks (ANNs), where traditional ANNs were inspired by the neural connections and functionality of animal brains and consist of interconnected nodes, or artificial neurons, which produce real-valued activations that feed to subsequent nodes. The strengths of these values are controlled by weights which are tuned over time, such that the network can learn a specific task. In this type of architecture, rows of adjacent nodes are referred to as layers and any layer between the input and output nodes of a network is referred to as a hidden layer. The idea behind deep learning stems from work like Hinton et al. [47], who sought to improve the learning capabilities of these networks by introducing much greater numbers of artificial neurons and increasing the number of hidden layers to help abstract much deeper underlying representations of the raw data. There are a number of different types of DNN architecture such as Convolutional Neural Networks (CNNs) [48], Deep Belief Networks [47], Autoencoders [49] and Recurrent Neural Networks [50], but in this work we focus primarily on CNNs as discussed in greater detail in Chapter 2.5.1.

2.5.1 Convolutional Neural Networks

CNNs were ground-breaking in helping to establish deep learning architectures and have successfully been applied to numerous domains such as face detection [51], image classification [52], speech recognition [53], natural language processing [54] and recommender systems [55]. CNNs have shown an incredible ability to process data that has been constructed in a grid-like fashion. Typically, the network architecture is represented as a series of layers represented in a grid structure, where each layer is composed of an array of units. Without considering the operations involved, on a basic level, information can then be passed from a neighbourhood of units of one layer to the next. This concept was directly inspired by Hubel and Wiesels [56] work relating to the perceptron and animal visual cortex [57]. Furthering this idea, the basic principle of a CNN is to convolve a series of filters or kernels over a layer to produce feature maps, which extract important feature information from the input. Typically, the greater the number of feature maps the more a model will be able to learn complex representations.

Fig. 2.8 demonstrates this principle, a series of n kernels is used to generate n feature maps to make a convolutional layer. Each kernel is represented by an array of weights. The weights are trained using traditional back-propagation methods such as the gradient descent process [50, 58]. The corresponding layer for each of these kernels is referred to as a convolution layer. In the early stages of the network, the resulting feature maps produced by the convolutional layer primarily capture linear patterns, such as edges and textures. For example, with image data, a feature map may capture linear information

as a result of object borders or changes in colours. One way to add in non-linearity is by using rectifier functions such as ReLu [59] or TanH [58]. Introducing non-linearity allows the network to learn more complex representations and patterns within the data, as such, feature maps deeper in the network will capture more abstract patterns. Also, certain activation functions like ReLu can help mitigate the vanishing gradient problem by avoiding saturated values that can occur with other activation functions, such as sigmoid or tanh. Furthermore, the ReLu function helps make the gradient sparser by converting negative values in a feature map to zero. Additionally, through a process of sub-sampling or down-sampling, it is possible to reduce the dimensionality of the data as it progresses through the various layers, helping to reduce the number of features to learn. This can be controlled by the kernel size or by using specific pooling layers to handle. Some examples of pooling layers are max pooling and average pooling. Furthermore, dropout layers may be employed along with other regularization techniques, to help prevent overfitting during training [60].

2.5.2 Neuroevolution of Deep Neural Networks

In this section, we will detail some of the most important considerations when considering neuroevolution. In this work, we look specifically at evolving Deep Neural Network architectures and use traditional stochastic gradient descent for weight training, a technique which is often referred to as Evolutionary Neural Architecture Search (ENAS). While the nomenclature may often deviate as to when to use neuroevoluion or ENAS, we have chosen to use the more general umbrella term of neuroevolution as the analysis and conditions for weight training are not fully explored in this thesis nor does the surrogate methodology stringently require a specific weight update mechanism.

First, we formally define the goal of the neuroevolution process, based on the definition of a CNN architecture by Sun et al. [25], as seen in Eq. 2.21, such that

$$\arg\max_{\lambda} \mathcal{F}(A_\lambda, \mathcal{D}_{train}, \mathcal{D}_{test}), \quad \lambda \in \Lambda \tag{2.21}$$

where $\mathcal{F}$ represents the fitness of the architecture in terms of its performance, A is the neural network architecture, $\mathcal{D}_{train}$ and $\mathcal{D}_{test}$ are the datasets for the training and test sets respectively, and λ is the set of architecture parameters, such as the layers, hyperparameters and weights from the total parameter space Λ. As such our evolutionary process is concerned with finding the set of λ in order to find the architecture with the maximum fitness. As mentioned, in this work we are only considering layers for the evolutionary process, such as the convolutional and pooling layers discussed in the previous section as well as different hyperparameter settings for these layers, where the weights are trained using back-propagation.

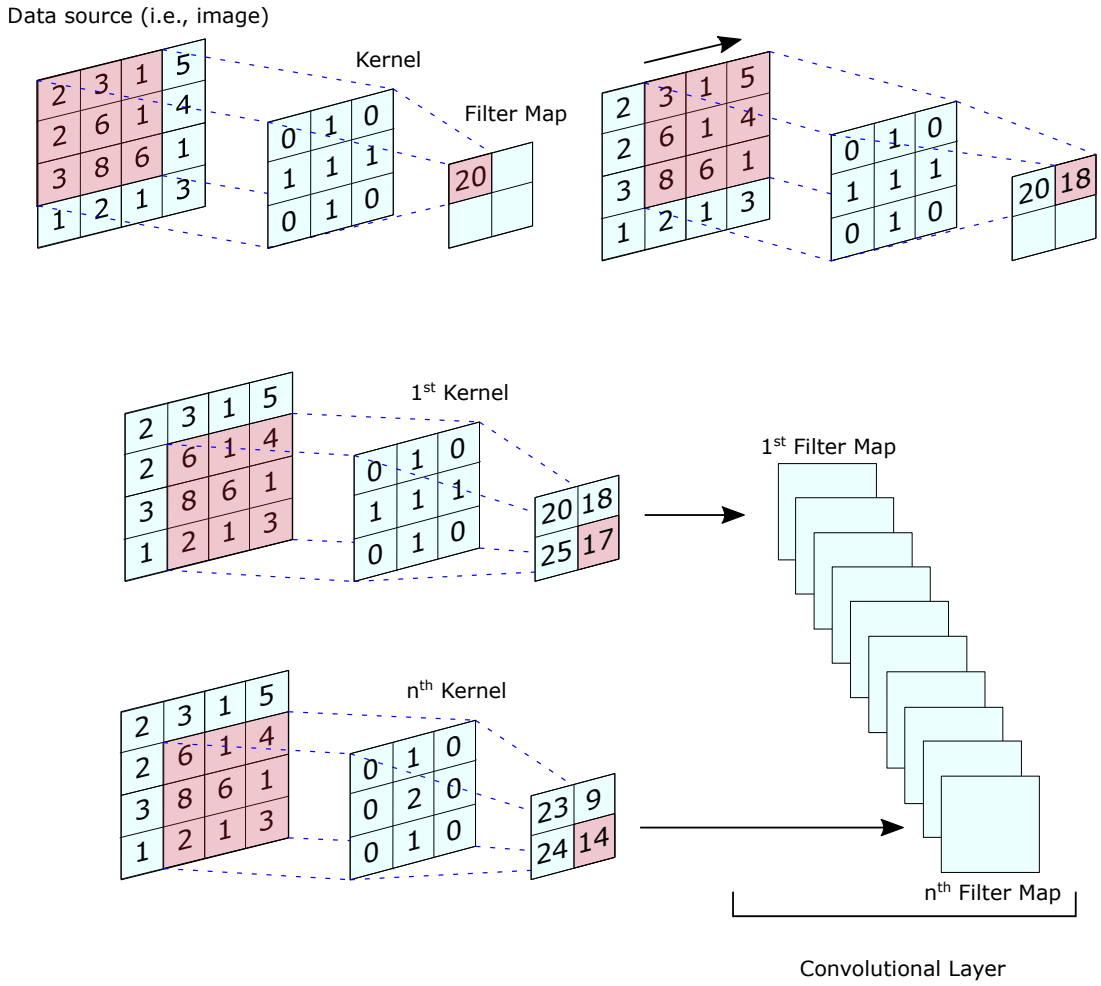


Figure 2.8: Demonstrating the convolution process in a CNN architecture. At the top, a kernel convolves values from a data source (i.e., pixel values of an image), which subsequently, make up a filter map. At the bottom, a series of kernels can be used to generate the convolutional layer. For simplicity's sake, integers have been used throughout this figure but it is important to note that, typically, values would consist of real numbers.

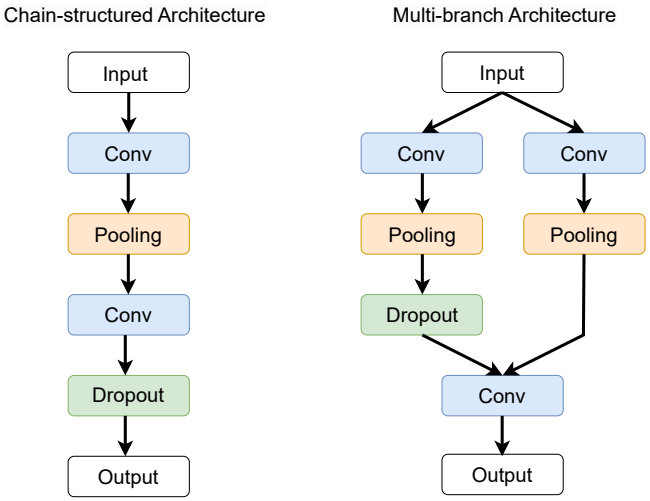


Figure 2.9: Example topologies for chain-structured networks (left) and multi-branch networks (right). Colour coding denotes layers of different types.

Another important consideration is the topological search space of potential architectures. There are two[2] main topologies to consider. The first example is *chain-structured* topologies. In Fig. 2.9, and on the left-hand side we can see an example of this topology. Here the architecture is represented as a sequence of layers, where the output of one layer directly feeds into next. As such, in terms of graph theory, the maximum degree a node (i.e., a layer) can have is two, since its edge will at most connect with a previous layer and the next layer. While the layers in this diagram have been colour-coded by their type it is important to note any layer of the same type (same colour) may have differing hyper-parameters in a real architecture. The second example is that of a *multi-branch* topology as seen on the right-hand side. Even though only a single branch is represented in this diagram there may be numerous branches and skip connections, allowing for many degrees at each node, ultimately resulting in much more complex network topologies compared to chain-structured topologies.

2.5.3 Further Details on Parameters and Layers used in this Work

Below is a list of the evolvable parameters for the neuroevolution approach in Chapters 5 and 6, giving details on their use in each case. The configuration files in Appendix B, shown in Tables B.1 and B.2 detail the specifics for each chapter, where there is a notable increase in potential parameters from Chapter 5 to 6. Certain parameters are not evolved or adapted, for instance, in all our experiments we use Adaptive moment estimation

[2]some publications will also specify cellular, hierarchical and macro search spaces, however, in most cases, these are also multi-branch topologies [61].

(Adam) [62] as an optimiser and He initialisation [63] to initialise the weights. As such, in the below list, we will only focus on parameters that are adaptable.

- **Convolutional Layers:** The mechanism for the convolutional layer [58], is discussed in detail in section 2.5.1, however, important parameters to highlight are the number of filters, which are the number of filters in the convolution and the kernel size which determines the size of the convolution window. As mentioned, activation functions such as ReLu [59] and TanH [58] are used, which help reduce or add in non-linearity. The stride i.e., the number of pixels/values that the convolutional window iterates over. Additionally, an l2-norm kernel regulariser is also available, which adds a penalty factor that can help reduce overfitting.

- **Pooling layers:** Maximum and average pooling are two available layers which can be evolved in this work. These are both reduction techniques, where a pooling window operates over the feature map and creates a down-sampled image based on the pixel/value information within that window. For maximum pooling, the maximum value is taken for all the values within the pooling window and for average pooling the average value is taken. Important parameters here are the stride i.e., the number of pixels/values the pooling window traverses over the feature map by, and the size of the pooling layer.

- **Batch Normalisation:** The batch normalisation layer [64], takes the inputs from one layer and normalises them before passing to the next layer and was originally designed to mitigate internal covariant shift. Momentum is a parameter relating to batch normalisation and helps alleviate noise in gradient updates.

- **Dropout:** The dropout layer [60], is a regularisation technique that masks a random selection of neurons within the network and can help prevent overfitting. The percentage of neurons dropped out is the key parameter of this layer.

3
Literature Review

In this chapter we cover some of the most relevant works relating to this thesis. Firstly, we give a brief history of Evolutionary Algorithms (EAs) and Artificial Neural Networks (ANNs) (Chapter 3.1). This section details some of the major developments in each field, albeit in an independent context. Next, we discuss some of the most popular machine learning approaches for implementing Neural Architecture Search (NAS), looking at Reinforcement Learning (RL), One-shot models and EAs (Chapter 3.2). Following on, we cover some of the major seminal works in neuroevolution, focusing first on early approaches that evolved both the weights and/or architecture of traditional neural networks before discussing more recent works that evolve the architectures of deeper more complex networks (Chapter 3.3). We then discuss surrogate-assisted evolutionary algorithms for neuroevolution with a particular focus on performance prediction approaches (Chapter 3.4). Finally, we discuss some of the major works in semantic and phenotypic distance. The latter two areas are closely related and while the study of semantics within genetic programming is well established, the use phenotypic distance vectors, particularly for their adoption in neuroevolution of DNNs is quite novel (Chapter 3.5).

3.1 A Brief History of Artificial Neural Networks and Evolutionary Algorithms

The study of EAs and ANNs both draw inspiration from nature. In the case of EAs, many of the underlying principles are inspired by biological evolution, while with ANNs inspiration is drawn from neuroscience, specifically the functionality of the animal brain. It is important to note that both fields developed independently from each other with some of the most significant foundational breakthroughs occurring in the 1960s, leading through to the modern day. First, we will discuss some of the major early works of each field independently before then discussing how EAs can be used for evolving ANN architectures later in the chapter.

The groundwork for the mathematical principles of ANNs stems from the study of human cognition, where one of the earliest works, by McCullock and Pitts [65], proposed the idea of an artificial neuron. Following on, Rosenblatt's seminal work relating to the Perceptron was the first physical computational example of an ANN [66]. Many components of this work are still present in modern neural networks, such as the input, hidden and output layers that consist of interconnected neurons and their associated weights. In this early work, an effective method for updating weights had yet to be developed and as such was only suitable for linearly separable data, however, later developments, such as the adoption of backpropagation overcame this issue [50].

In fact, backpropagation would play a critical role in the development of deeper ANNs. LeCun [48] demonstrated how the techniques could be used during the classification of handwritten zip codes, using images rather than the feature vectors of previous work [67]. Furthermore, this seminal work details how feature maps can be convolved over each image to extract important feature information which is then fed into fully connected layers. This type of neural network, known as a Convolutional Neural Network (CNN), had first been developed by Fukushima [68], drawing inspiration from the work of Hubel and Wiesel on the animal visual cortex [56], however the adoption of backpropagation marked a significant step forward for the development of deep neural networks.

A short summary of other major ANN developments include but are not limit to, Recurrent Neural Networks (RNN) [50], which are particularly well suited to sequential data, Long-short Term Memory (LSTM) [69], which are a form of RNN but are capable of learning longer term dependencies within the sequential data, Restricted Boltzman Machines (RBN), which are a form of stochastic neural network designed to learn a probability distribution based on its inputs, Deep Belief Networks [47], which use stacked RBNs within its architectures and, AutoEncoders [49], which have an encoding-decoding architecture, where input sequence data is encoded to a embedded space before being decoded as an output sequence and, Transformers [70], which again used an encoding-decoding architecture but use a multi-headed attention mechanism to encapsulate dependencies within the embedded space and have seen a surge in interest in recent years due to their use in large language models.

Similarly, an early breakthrough in the field of EAs was by Fogel et al. [35], known as Evolutionary Programming (EP). They proposed a method for evolving finite-state machines over time, with the aim of predicting output sequences based on input sequences to the finite-state machine. Closely related are Evolutionary Strategies (ES) [71], where traditionally, the real-valued representations are evolved. In both techniques, mutation is the main operator for altering individuals for future generations, whereas the crossover operator is seldom used in ES and is generally absent in EP.

In the 1970's Holland [72] introduced Genetic Algorithms (GAs) and further popu-

larised by Goldberg [33]. With GAs the candidate solution is now an encoded representation of the problem solution, using binary strings known as the chromosome, where genetic operations alter this encoded structure directly. Crossover operations often play a much more significant role in GAs and this representation is suitable for both discrete and combinatorial search spaces.

While Cramer [73] adapted concepts from GAs in order to evolve short computer programs, in a technique that is known as Genetic Programming (GP), it was not until later, in the 1990s, that the approach was popularised by Koza, who produced his seminal work relating to GP [74]. With GP, the idea is to evolve programs, such that they find candidate solutions to a problem without the need to have been explicitly programmed by a user. In GP the chromosome uses a tree-based encoding as opposed to the binary string typically found in GAs. GP is suitable for complex optimisation problems and has been shown to produce human competitive results [75].

EP, ES, GA and GP constitute the four-main paradigms of EAs. Other variations of EAs techniques or meta-heuristics inspired by EAs, which have been developed over the years, which include, but are not limited to Differential Evolution (DE) [76], a population-based approach that uses the scaled differences of vectors to optimise toward a solution, Particle Swarm Optimisation [77] which is similar to DE in terms of its evolutionary process but the mechanics of which are inspired by swarm behaviour, Grammatical evolution [78] which is similar to GP but uses production rules based on Backus-Naur grammar definition, Cartesian Genetic Programming [79] that derives its name from its grid-based encoding that can be used to represent directed-acyclic graphs and Linear Genetic Programming (LGP) [10] which is used to evolve computer programs, where programs within the population are defined using sequences of register and operands. Both traditional GP and LGP are described in full detail in Chapter 2.2.

3.2 Neural Architecture Search

This chapter section serves as an overview for some of the main machine learning approaches that have been applied to NAS. Thus far, the following fields have seen some of the most interest over the years.

Reinforcement Learning: With RL-based approaches the idea is that the RL method acts as a decision-making agent responsible for sampling high-quality networks from the search space of all possible architectures. In most cases, the architectures are generated as a sequence of actions based on this agent. These actions are responsible for the overall architecture of the network, and a reward signal, which is some measure of the architecture performance can be used to update the agent and subsequently maximise the expected value. For example, Zoph and Le [19] used an RNN as a decision-making agent

responsible for constructing a DNN architecture. With RL, often these architectures are composed of cells or blocks, which are predefined sub-architectures that may be reused throughout the overall architecture [80], though cellular-based search is not limited to reinforcement learning. It has been noted that other approaches have seen more interest in recent years [61].

One Shot Models: These approaches use a single over-parameterised supernet, where sub-graphs represent different candidate architectures. As such, the search is conducted over a predefined search space represented by the supernet. Additionally, one-shot models often use weight sharing to help reduce the training time associated with other approaches [18]. One of the earliest one-shot methods was developed around the NASNet search space [18]. This approach makes use of a cellular-based implementation, consisting of two types of cells: a normal cell and a reduction cell. The normal cell contains layers where the feature size input and output are the same size. The reduction cell is responsible for decreasing the resolution of the input data. This was soon followed by DARTS [81] which made some significant steps forward for one-shot type architecture. Firstly, DARTS relaxes the discrete search space of the supernet, by using softmax, allowing for optimisation in a continuous search space, secondly, this approach uses gradient descent to optimise the weights, discussed in Chapter 2.5.1, and thirdly, operations are performed on the edges of the graphs rather than the nodes. Many of the one-shot architecture search spaces now use benchmarks, for example, NASBench-101 [82], which is compatible with popular datasets like CIFAR-10 [83] and ImageNet [84]. While one-shot models have the advantage of often being many orders of magnitude faster than RL and EA approaches it has been noted in some studies that performance wise they often only perform marginally better or similar compared to random search [85, 86].

Evolutionary Algorithms: EAs can be used to evolve an encoded representation of the underlying neural network architecture over time. Many of these approaches differ in the specifics of how the evolutionary process is implemented [37], but typically offspring architectures are generated from parent architectures where genetic operations will alter their structure, for example, the number of layers, and subsequently, these offspring are retained based on the quality of there fitness. This technique of using EAs is typically referred to as neuroevolution. Originally, interest centred on evolving both network weights as well the network architecture [87, 88] but within the last two decades research in neuroevolution has focused more so on architecture, since evolving weights for DNNs is computationally prohibitive. One of the major transitionary works between weight-focused and architecture-focused neuroevolution, was NeuroEvolution of Augmenting Topologies (NEAT) by Stanley and Miikkulainen, as this work was later adapted to architecture-focused neuroevolution [89–91]. NEAT and its variants will be discussed in greater detail in the next sections.

There are other approaches to NAS to be aware of, such as Bayesian Optimisation [92], Hill Climbing [93], and Monte Carlo Tree Search [94, 95], to name a few, but will not be discussed in detail in this thesis. For the remainder of this chapter, we will focus mainly on evolution-based techniques for NAS but for more details, the 2018 survey paper by Elsken [80] and more recent 2023 paper by White et al. [61] cover some of the major topics and innovations for NAS in recent years.

3.3 Neuroevolution

3.3.1 Artificial Neural Networks

Two of the earliest examples of using EAs to evolve neural network architecture were by Miller et al. [96] and Harp et al. [97], both published in 1989. While earlier works had considered using GAs to evolve weights [98], these works are two of the first to consider evolving the connections between neurons and hence alter the overall topology of the networks. Another early example of applying EAs for neuroevolution was by Whitley et al. [87]. In this work, they investigated two experimental types. Firstly, they investigate the use of GAs rather than the standard back-propagation technique, in order to evolve neural network weight connections using both binary and real-valued representations. Secondly, they use GAs to evolve connectivity patterns to find neural network architectures, in this scenario using back-propagation to update the weights.

Yao and Liu [99] proposed Evolutionary Programming-Net (EPNet) a method which uses EP to evolve both the weights and architecture of a network simultaneously. Notably, an emphasis is put on the network behaviour during evolution, maintaining a behavioural link through parents and offspring through partial training, as well as node-splitting techniques. By applying a systematic approach to mutation they help reduce destructive properties of randomised mutations. Another important aspect of this work is that the learned weights and architectures are inherited into the next generation, akin to Lamarckian evolution.

NeuroEvolution of Augmenting Topologies (NEAT), proposed by Stanley and Miikkulainen, is one of the most popular and groundbreaking approaches for Neuroevolution in ANNs [88]. The approach evolves not only the architecture structure (or topology) but also the connection weights and as such is an example of a direct encoding approach, i.e., all the network parameters are encoded within the genome. NEAT works by a process of complexification where initial architectures are uniformly structurally simple and are gradually built up to become more complex with time. Complexification has a strong foundation in biology where biological genomes experience growth via a process known as gene duplication. Gene duplication copies parental genes into the offspring genome more

than once, resulting in redundant genes that express the same proteins.

3.3.2 Deep Neural Networks

By the mid to late 2000s research in neuroevolution began to shift towards deeper network architectures. For instance, NEAT was further extended where the first major approach to tackle deeper networks was HyperNEAT by Stanley et al. [89], which exploits reoccurring patterns in the data by using connective Compositional Pattern Producing Networks (CPNNs) to produce larger networks [89] [100]. DeepNEAT, proposed by Miikkulainen et al. [90] extends the concept of NEAT where now each gene no longer represent single neurons but individual layers of DNNs (referred to as modules). Each node consists of a table of binary or real values for the hyperparameters and determine the type of layer at that node, as well as its properties. The edges of the graph now represent how layers are connected rather than how individual neurons are connected (and their weights). The final DNN is constructed via traversing the graph, replacing each node with its respective layer. The authors provide a criticism of DeepNEAT, in that the resulting topologies can be unconventional and complex. As such, the same authors provide an extension of this approach known as CoDeepNEAT [90], using the coevolution of population blueprints (network connection templates) and a population of modules. While the original authors do not provide source code, a Keras implementation has recently been developed [100].

Also of note, is the work of Desell [91] who adapted NEAT to work specifically on CNNs using volunteered computing. The author's approach made use of both mutation and crossover, but with specific mutation operators for manipulating edge connections. This work is notable in that it demonstrates the ability to scale neuroevolution to large-scale distributed computing environments with notable results on the MNIST dataset [101]. Some drawbacks of their approach are that it could not incorporate pooling layers and was restricted to two-dimensional convolutional layers.

A well-known neuroevolution approach is GeneticCNN [102] by Xie and Yuille. This approach uses binary encoded strings to represent the connections between a series of convolutional layers. The architecture makes use of what the authors call stages, where the encoded set of CNN layers are followed by fixed pooling layers. These stages are chained together to comprise the overall architecture. As such, only a subset of the architecture is evolved where each stage is responsible for finding interesting micro-architectures within the overall network. This approach therefore is limited by its genotypic representation to a large domain of potential structures. The authors note that other modules such as Maxout and Inception models cannot be found using this approach.

Recently, Z. Shuai et al. [103] proposed an approach that makes use of a type-free search space composed of cells and organs. Cells are composed of a core module with

affiliate modules directly related to that core module and organs are responsible for higher level functionality such as feature extraction or classification. For example, a cell may be composed of convolutional layer with an affiliated ReLu and pooling layer and an organ may be composed of several cells each containing convolutional layers but with differing affiliate modules. They conducted their experiments on a variety of convolutional neural networks (CNN), general adversarial networks and long short-term networks.

Sun et al. [104] proposed the EvoCNN algorithm which uses a variable-length encoding to search for CNN architectures of potentially optimal depth. This approach also incorporates a novel weight encoding strategy, that efficiently searches for initial weight connections using just the mean and standard deviation of a Gaussian distribution, using a low number of epochs, to begin with. Upon completion, the best-fit model is then deeply trained. As such, this approach efficiently initialises the weights, while at the same time also searching for optimal architectures. As mentioned, EvoCNN uses variable-length encoding [104], whereas Genetic-CNN is somewhat fixed by the number of stages selected and the length of the binary string encoding. EvoCNN in a sense uses 'blocks' to generate sequences of layers. The genetic operations of mutation and crossover then operate on these blocks. Interestingly, the crossover operation works by swapping blocks between parents thus producing offspring of the same size and as such is an example of uniform crossover. The architectures instead 'grow' in size based on some predefined probability where a new block is randomly assigned.

Neuroevolution has also been expanded to look at transformer-based architectures. Yang Xiu and Yongjie Ma proposed an evolutionary NAS approach using an improved transformer block as well as being capable of handling multiple branches. The 'batch free normalisation transformer' block avoids problems associated with small batch size and degradation of performance if there is a covariance shift between training and test data. The 'multi-branch' block uses a multi-path topology which allows the architecture to branch to sub architectures with differing scales and complexity. A survey by Chitty-Venkata et al. [105] details some of the recent advancements in NAS for transformers, including RL, One-shot and EA approaches.

To date, there are few works that use Genetic Programming (GP) [106] in the encoding for neuroevolution for DNNs. Suganuma et al. [107] used Cartesian Genetic Programming [108] to design CNN architectures. In CGP a 2-dimensional grid of nodes is used as the genetic representation, where the grid structure can be used to define a directed acyclic graph. The genotype consists of integers where each gene encodes information regarding the type and connections between each of the nodes. The design adopts highly functional modules (i.e., ConvBlock and ResBlock) to search architectures efficiently. Additionally, this representation by design allows for skip connections and multiple branches. The authors note that the representation is fixed-length, but that the size of the networks

varies since not all nodes are connected at once.

Assunção et al. [109] introduced a new approach, Deep Evolutionary Network Structured Representation (DENSER), that makes use of GA to adapt the macro structure of the architecture, represented as a sequence of units, where each unit may be responsible for a layer, training or data-augmentation. Each unit then represents a starting non-terminal symbol for the expansion of a Dynamic Structured Grammatical Evolution (DSGE) level genotype. The DGSE level is therefore responsible for parameters, either represented as ranges or closed sets of different possibilities. Later Assunção et al. [110, 111] further improved this approach, entitled Fast-DENSER, such that is could now incorporate skip connections as well as offering a significant time improvement over the original. They demonstrate that they could get a similar performance to the previous DENSER approach by increasing the rate of mutation while lowering the number of evaluations from 10,000 to just 750. The authors noted an average test accuracy of 91.46% on CIFAR-10 making their approach competitive with state-of-the-art.

3.4 Surrogate-Assisted Evolutionary Algorithms for Neuroevolution

In this section, we will discuss works mainly relating to surrogate-assisted evolutionary algorithms (SAEA) for neuroevolution with a particular focus on performance prediction. In some of the below works [112] [113], an explicit evolution technique may not be present but rather the authors instead focus on the specifics of the surrogate modelling technique. These works have been included regardless, due to their relevance to this work as a whole. Before discussing these works in more detail there are some important surveys on surrogate modelling we would like to draw attention to. An old, but still highly relevant survey on surrogate models by Jin [22], covers some of the fundamental aspects of surrogate-assisted EAs, such as surrogate model management strategies and acquisition functions. A more recent 2023 survey by Khaldi and Draa [114], covers more state-of-the-art concepts, such as considering the computational complexity of a surrogate model when performing neuroevolution. Furthermore, a 2022 survey by Liu et al, [115] considers computationally efficient approaches to NAS while also detailing recent advances specifically for surrogate assisted evolutionary NAS. Next, we will cover some of the most relevant works.

A key advantage of using surrogate models for neuroevolution is that they can be used as performance predictors for ANNs. Performance prediction can be categorised into two branches: (i) approaches that infer the performance of unseen networks based on specific characteristics of previously evaluated networks and which typically have been fully trained, for example end-to-end performance predictors [25, 116, 117], and (ii) approaches that instead used partially trained information to predict the future performance of a

network, for example using learning curve based prediction [112, 118].

A major work that falls into the first category is the End-to-End Performance Predictor (E2EPP) [25] proposed by Sun et al. This approach uses an offline surrogate model based on random forests to search for optimal CNN architectures. The CNN is cleverly encoded such that it maps to a numerical decision variable, which can then be processed by a random forest surrogate. This approach alleviates restrictions found in other approaches such as assuming a smooth learning curve and not requiring large amounts of training. The authors approach was shown to speed up fitness evaluations while also achieving the best performance compared to other peer performance predictors.

Another example is the work of Greenwood and McDonnell [117], who proposed a grammar-based approach which generates a tensor representation of variable-length DNN topologies. An advantage of using formal grammar as a representation is that they can be designed to ensure validity of models by describing the space of allowable topologies. Their approach modifies the DeepNeat algorithm first proposed by Miikkulainen [90]. The modelling strategy of this work is designed around a two-phase approach. Firstly, during the initialisation phase, the DeepNeat algorithm is used to evolve the population of neural networks and these models are used to formulate the surrogate. Secondly, the active learning phase is implemented where new networks are evaluated using the surrogate model, which uses a single LSTM layer with 20 hidden units, followed by a 30 unit linear layer with ReLU activation function. Additionally, a subset of networks that are fully trained and evaluated to further inform and improve the surrogate model. They noted a five times improvement in compute time.

In another example, Fan and Wang [119] used unsupervised learning approach using a network embedding strategy using Graph2vec [120] to build a surrogate-assisted neural architecture search (SAENAS-ES) model. Graph2vec works by training a skip-gram model to get an architecture embedding by predicting if an architecture substructure, or sub-graph, exists and then asserts that architectures with more similar substructures are closer in the embedding space. The resulting embeddings can be then fed to a surrogate model, RankNet [121], to evaluate the architectures. Results for SAENAS-ES on three NASBench and DARTS search spaces demonstrated that the surrogate model with an embedding strategy performed as well or better than other approaches.

Approaches falling into the second category, i.e., methods that use partially trained information, include the Freeze-Thaw Bayesian Optimization (FBO) technique proposed by Swersky et al. [112]. It uses Bayesian optimization to decide if a partially trained network should be trained to completion. The fundamental idea is that, in practice, a human expert is quickly able to assess if a network is likely to result in poor performance and as such can decide to halt training. Based on this notion the authors designed an approach that at its core is a form of performance prediction, where Bayesian Optimisation

is used to determine whether a particular partially trained network will yield preferable results compared to other networks. Unlike E2EPP, this approach does require a smooth learning curve which can be hampered if different learning rates are considered. Of interest, is that the FBO approach relies on the behaviour, rather than the structure of the network in order to decide which networks to evaluate.

Other developments relating to learning curve-based prediction are the work of Baker et al. [113], who used a subset of validation accuracies as time series information, along with network features such as architecture parameters and hyperparameters, to build sequential regression models to predict learning curves. Yan et al. [122] developed a learning curve extrapolation framework by developing NAS-Bench-x11, a surrogate NAS benchmark that outputs the entire training information of each network. This framework is capable of incorporating many single and multi-fidelity NAS approaches, including evolutionary-based algorithms like regularised evolution [123].

Other approaches have taken a more indirect approach for performance prediction, some which use surrogate modelling specifically or other ML approaches for similar effect. For instance, Gaier et al. [24] devised a kernel-based surrogate model to use with NEAT [88] referred to as Surrogate-assisted NEAT. To overcome the limitation of variable-length genotypes, they exploited a distance metric inherent in the NEAT algorithm. This distance metric which is known as the 'compatibility distance' was originally designed to help promote diversity amongst the network topologies through speciation. Within this work they also use it as a distance metric to use for Gaussian Processes. The authors note a motivating factor behind using this distance metric rather than one based solely on the structure of the architecture, is that, for differing topologies, constructing a static or consistent parameter space is difficult and as such using standard metrics like Euclidean distance may not be feasible with Kriging. They demonstrated that they were able to achieve similar performance to NEAT with much fewer evaluations.

Likewise, Yuan et al. [124] employed a random forest-based performance prediction approach to pre-select offspring aiming to select individuals which ought to be high-performing. One major benefit of this method is that since the offspring are fully evaluated, the approach naturally avoids inaccurate predictions. In addition to using performance prediction, the authors incorporate weight inheritance and proposed an efficient backbone structure based on custom blocks inspired by the lightweight MobileNetV3 architecture [125]. While this approach does not refer to the random forest predictor as a surrogate model directly, nevertheless, this approach shows the effectiveness of considering performance prediction at various stages of the evolutionary process.

3.5 Semantics in Genetic Programming and Phenotypic Distance

3.5.1 Semantics in Genetic Programming

The range of problem domains for GP are wide and this form of EA has been found to be beneficial for problems with varying degrees of complexity, for example, problems with multiple local optima [126]. Furthermore, EAs are well suited for highly complex problems including the automatic configuration of deep neural networks' architectures and their training (an in-depth recent literature review in this emerging research area can be found in [127]). However, despite the well-documented effectiveness of canonical GP, there are well-known limitations of these methods, as found through the study of properties of encodings, [128, 129], and research is ongoing into finding and developing approaches to improve their overall performance. One such area is the study of semantics which has seen a dramatic increase in interest over the last years given that it has been consistently reported to be beneficial in GP search, ranging from the study of geometric operators [130], including the analysis of indirect semantics [28,131]. While most of the following works are directly related to Chapter 4, they also offer a significant understanding of the potential impacts of considering phenotypic distances in neuroevolution.

Even though researchers have proposed a variety of mechanisms to use the semantics of GP programs to guide a search, it is commonly accepted that semantics refers to the output of a GP program once it is executed on a data set (also known as fitness cases in the specialised GP literature). Semantics can be classified into two main categories: indirect or direct. Indirect semantic approaches refer to approaches that indirectly increase semantic diversity by acting on the syntax of GP individuals. Direct semantic approaches adapt genetic operators to work directly on the semantics of GP individuals. The semantic methods detailed in this thesis focus exclusively on indirect semantic methods.

The work conducted by McPhee et al. [132] paved the way for the proliferation of indirect semantics works. In their research, the authors studied the semantics of subtrees and the semantics of context (the remainder of a tree after the removal of a subtree). In their studies, the authors pointed out how a high proportion of individuals created by the widely used 90-10 crossover operator (i.e., 90%-10% internal-external node selection policy) are semantically equivalent. That is, the crossover operator does not have any useful impact on the semantic GP space, which in consequence leads to a lack of performance increase as evolution continues.

Uy et al. [133] proposed four different forms of applying semantic crossover operators on real-valued problems (e.g., symbolic regression problems). To this end, the authors

measured the semantic equivalence of two expressions by measuring them against a random set of points sampled from the domain. If the resulting outputs of these two expressions were close to each other, subject to a threshold value called semantic sensitivity, these expressions were regarded as semantically equivalent. In their experimental design, the authors proposed four scenarios. In their first two scenarios, Uy et al. focused their attention on the semantics of subtrees. More specifically, for Scenario I, they tried to encourage semantic diversity by repeating crossover for a number of trials if two subtrees were semantically equivalent. Scenario II explored the opposite idea of Scenario I. For the last two scenarios, the authors focused their attention on the entire trees. That is, for Scenario III Uy et al. checked if offspring and parents were semantically equivalent. If so, the parents were transmitted into the following generation and the offspring were discarded. The authors explored the opposite of this idea in Scenario IV (children semantically different from their parents). They showed, for a number of symbolic regression problems, that Scenario I produced better results compared to the other three scenarios.

The major drawback with the Uy et al. [133] approach is that it can be computational expensive, since it relies on a trial mechanism that attempts to find semantically different individuals via the execution of the crossover operator multiple times. To overcome this limitation, Galván et al. [134] proposed a cost-effective mechanism based on the tournament selection operator to promote semantic diversity. More specifically, the tournament selection of the first parent is done as usual. That is, the fittest individual is chosen from a pool of individuals randomly picked from the population. The second parent is chosen from a pool of individuals that are semantically different from the first parent and but is also the fittest individual from that pool. If there is no individual semantically different from the first parent, then the tournament selection of the second parent is performed as usual. The proposed approach resulted in similar, and in some cases better, results compared to those reported by Uy et al. [28, 133] without the need of a trial and error (expensive) mechanism.

Forstenlechner et al. [135] proposed two semantic operators, Effective Semantic (ES) Crossover for Program Synthesis and ES Mutation for Program Synthesis. Program synthesis operates on a range of different data types as opposed to those that work in a single data type such as real-valued cases [136] and Boolean values [134]. The main elements considered by the authors were the metrics used to determine semantic similarity, named partial change, used in the first instance, and any change, used only if the first failed to be satisfied to avoid using standard crossover. In their results, the authors reported that a semantic-based GP system achieved better results in 4 out of 8 problems used in their studies.

Alternative to indirect semantic approaches, direct semantic approaches have also garnered attention within the GP community. Based on strong theoretical foundations [137],

Moraglio et al. [130] proposed new semantic-based geometric operators, which rather than only considering the syntactic relationship between parent and offspring, instead directly consider the semantic relationship. This relationship is captured using semantic distance and allows offspring to inherit desirable properties from their parents. Furthermore, the mapped semantic fitness landscape results in a conal landscape, which is beneficial to convergence for the evolutionary process.

3.5.2 Semantics and its Relevance to Surrogate-Assisted Neuroevolution

Despite an abundance of literature detailing the benefit of semantics in genetic programming 3.5.1, to date, there have been relatively few works that have been implemented or made a direct connection between semantics and phenotypic distance for neuroevolution. Nevertheless, we will demonstrate in greater detail how semantics can be used to inform surrogate models in Chapters 4 - 6.

Santos et al. [138] proposed a novel approach that makes use of semantics in neuroevolution. They do so by using Geometric Semantic Genetic Programming, in conjunction with a neuroevolutionary approach called Semantic Learning Machines (SLM) [139]. With SLM, mutation works by adding a random network to a parent network. Barring the input layer, connections can only flow from the parent to the random network, as such allowing the parent network to retain its semantic information. Their approach is also notable for its efficiency, capable of finding reasonable performing networks on the scale of GPU minutes as opposed to GPU hours or days.

Hagg et al. [23], made a connection between semantic distances in GP and phenotypic distances within the context of surrogate-assisted evolutionary algorithms for neuroevolution, in that, semantics distance can be understood as a distance of the outputs of GP individuals, determined with the same measure that is used in the fitness function, while phenotypic distances may also derive from the fitness function but not necessarily. Similarly, Stork et al. [27] extended CGP to use a surrogate-assisted neuroevolution approach that uses phenotyic distance vectors to estimate fitness. A limitation of Stork's work is the scalability of using Kriging on high-dimensional data. For instance, our recent work highlighted that traditional approaches such as the Kriging approach suffer from high-dimensionality [5] and may not be suitable for DNN architectures. In Chapter 5, we discuss this limitation at length, proposing a more suitable surrogate model management strategy for handling larger phenotypic distance vectors, and propose a new surrogate model variant entitled NeuroLGP-SM [6, 7]

4

Semantic-based Metrics in Multi-objective Genetic Programming

---

The following papers summarise the work outlined in this chapter.

- Edgar Galván and Fergal Stapleton. Semantic-based distance approaches in multi-objective genetic programming. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 149–156. IEEE, 2020

- Edgar Galván, Leonardo Trujillo, and Fergal Stapleton. Semantics in multi-objective genetic programming. *Applied Soft Computing*, 115:108143, 2022

- Edgar Galván, Leonardo Trujillo, and Fergal Stapleton. Highlights of semantics in multi-objective genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 19–20, 2022

- Fergal Stapleton and Edgar Galván. Semantic neighborhood ordering in multi-objective genetic programming based on decomposition. In *2021 IEEE Congress on Evolutionary Computation (CEC)*, pages 580–587. IEEE, 2021

Semantic diversity in Genetic Programming (GP) is a diversity-preserving measure based on the outputs (behaviour) of a GP program and has proven to be highly beneficial in evolutionary search. We have witnessed a surge in the number of scientific works in the area, starting first in discrete spaces [132, 140, 141] and moving then to continuous spaces [133]. The vast majority of these works, however, have focused on single-objective genetic programming paradigms. Therefore, the study of semantics in multi-objective (MO) GP has been limited, an issue we address in this chapter, though it is not the central focus. In its origins, MO started with naïve approaches such as using a sum of elements in the fitness function, very often with associated weights [142]. Nowadays, however, one can say that there are two major approaches for conducting MO optimisation. Firstly, one can use the Pareto dominance relationship to directly search for the Pareto optimal front or secondly, one can decompose the multi-objective problem into multiple single-objective sub-problems. As such, the focus of this chapter is to demonstrate the adaptability and robustness of semantics for when we later consider its use in surrogate-assisted neuroevolution, using multiple different semantic-based distance metrics, while considering its novel application to two inherently different EA workflows: i) *Pareto-based* and ii) *Dominance-based*.

i) *Pareto-based:* We conduct a comparison of three different forms of semantics in Pareto-based MOGP. Firstly, Semantic Similarity-based Crossover (SSC), is borrowed from single-objective GP [28], where the method has consistently been reported beneficial in evolutionary search. We also study two other methods, dubbed Semantic-based Distance as an additional criteriOn (SDO) [38] and Pivot similarity SDO (PSDO) [2]. We empirically and consistently show how by naturally handling semantic distance as an additional criterion to be optimised in MOGP leads to better performance when compared to canonical methods and SSC. Both semantic distance-based approaches make use of a pivot, which is a reference point from the sparsest region of the search space and it was found that both individuals which were semantically similar, and those which were semantically dissimilar to this pivot were beneficial in promoting diversity. Moreover, we also show how the semantics defined in single-objective optimisation does not necessarily lead to a better performance when adopted in MOGP.

ii) *Dominance-based:* First, we discuss the limitations of using the above semantic distance-based approaches in dominance-based MOEA/D, which suffer as a result of the localised neighbourhood structure that is particular to MOEA/D. Then, we show, for the first time, how we can naturally promote semantic diversity in MOEA/D in GP using semantic neighbourhood ordering (SNO), an approach that is designed specifically to tackle these limitations. The SNO approach helps select semantically preferable individuals within each MOEA/D neighbourhood by ordering them based on their semantic distance from individuals from the sparsest region of the archived Pareto front [3].

While semantics in the context of GP, is often associated with promoting diversity, it is important to highlight that program semantics itself is often a vector representation of the output of the program tree, given a set of inputs (or fitness cases), and signifies the behaviour of the program tree rather than its syntactic structure. In this chapter, program semantics are used to create distance metrics to help promote diversity. How these distance metrics are used depends greatly on the framework being employed or the specific nature in which semantics will be used to promote diversity. In the following chapters, inspired by the use of semantic distance-based metrics, we again use the notion of semantics to build distance vectors (in some literature this is synonymous to phenotypic distance vectors [27]) but in this context, these vectors will be used to impute the performance of partially trained networks for use within a surrogate-assisted framework for neuroevolution. In this work, this method of imputing performance based on these vectors is referred to as semantic-based surrogate-assisted neuroevolution, as discussed in Chapters 5 and 6.

4.1 Introduction

Genetic Programming (GP) [36] is a powerful Evolutionary Algorithm (EA) [20] that has been successfully applied in a variety of challenging problems including finding the automatic configuration of modern neural networks [37, 127] and energy-based problems [143, 144]. Despite the popularity of GP and its proven effectiveness in the face of challenging problem features such as deceptiveness and multiple local optima [126], it is also well-known that canonical GP has some limitations and researchers have developed new approaches to make it more reliable. To this end, researchers have focused their attention on neutrality [37,145–149], locality [128,129,150], reuse of code [151], to mention some research topics.

One of the most popular elements studied by GP researchers during recent years is semantics, with many works reporting substantial improvements in GP performance achieved by encouraging semantic diversity during evolutionary runs. While multiple definitions of semantics have been proposed in the GP community, it is commonly accepted that semantics refers to the output of a GP program when it is executed on a (training) data set (also known as fitness cases).

Semantics can be classified into one of two main categories: indirect or direct. Indirect semantic approaches refer to those that act on the syntax of GP individuals to indirectly increase semantic diversity. On the other hand, direct semantic approaches refer to those mechanisms that adapt genetic operators to work directly on the semantics of GP individuals. Both types of approaches have benefits and limitations. This work focuses on indirect semantic-based approaches. A formal treatment and more comprehensive

overview of semantics is given in Chapter 2.2.3.

To date, the use of semantics has been limited to a single-objective context. As such, we will outline four semantic approaches that can be used in a multi-objective context. A summary of the four semantics approaches used in this chapter is as follows:

Semantic Similarity-based Crossover (SSC). This method was first proposed by Uy et al. [28] in the context of single-optimisation GP. This method uses a computationally expensive procedure by applying crossover between two parents multiple times using semantic diversity as a criteria in the selection process.

Semantic-based Distance as an additional criteriOn (SDO). This is a method which originates as an improvement to the crowding distance operation as proposed by Galván et al. [38]. This method uses a reference point or pivot from the sparsely populated region of the search space and computes the semantic distance between this pivot and every individual. This distance is optimised as an additional criterion in Evolutionary Multiobjective Optimisation (EMO).

Pivot similarity Semantic-based Distance as an additional criteriOn (PSDO). It is a method which is a variation of SDO but instead prefers solutions that are semantically similar to the pivot as proposed by Galván et al. [2].

Semantic Neighbourhood Ordering (SNO). This method first proposed by Stapleton et al. [3], again uses a reference point or pivot from the sparsely populated region of the external archive, which represents the best solutions found so far. Solutions are ordered based on their semantic similarity to these sparse regions and single replacement is incorporated in the neighbourhood update to help retain diverse candidate solutions that explore discontinuous regions of the Pareto front.

Broadly speaking, there are two major framework implementations of MO: Pareto-based approaches, which use the dominance relationship to find non-dominated solutions that represent the Pareto front (see Chapters 2.4.2, 2.4.3 and 2.4.4 for details) and decomposition-based which decomposes the multi-objective problem into multiple scalar objective sub-problems which are solved in tandem (see Chapter 2.4.6 for details). Table 4.1 offers a brief taxonomy of the semantic methods and their corresponding MOGP frameworks and optimisation strategies. It should be noted that SDO is also applied to MOEA/D to demonstrate the limitations of this method.

Based on these two implementations we will conduct an in-depth analysis of semantic approaches in MO optimisation. Firstly, a comparison is performed using various semantic approaches specifically for two Pareto-based approaches: the Non-dominated Sorting Genetic Algorithm II (NSGA-II) [152] and the Strength Pareto Evolutionary Algorithm

Semantic Approach	MOGP Framework	Optimisation Strategy
SSC	NSGA-II & SPEA2	Pareto-based
SDO	NSGA-II & SPEA2	Pareto-based
PSDO	NSGA-II & SPEA2	Pareto-based
SNO	MOEA/D	Dominance-based

Table 4.1: Summary of the approaches and where they are used. In Chapter 4.3.2 SDO is also applied to MOEA/D to demonstrate the limitations of this method.

(SPEA2) [45]. For the Pareto-based implementations, the semantics approaches are SSC, SDO and PSDO, the results and analysis of which are discussed in Chapter 4.3.1.

Next, we will turn our attention to a decomposition approach: Multi-objective Evolutionary Algorithm with Decomposition (MOEA/D) [46]. When discussing MOEA/D specifically, we highlight a limitation found in Pareto-based distance metrics, using SDO, and propose a novel approach, SNO, for using semantic distance in the context of decomposition as discussed in greater detail Chapter 4.1.2 and the analysis and results of which are discussed in greater detail in Chapter 4.3.2.

For the Pareto-based implementations, we demonstrate that it is possible to naturally incorporate semantics in MOGP leading to a better performance. Furthermore, we demonstrate the robustness of the proposed method by using two different forms of computing semantic distance, used in SDO and PSDO, leading to similar results by these two methods, and demonstrates how a widely successful form of semantics used in single-objective GP does not necessarily yield good results in MOGP.

For the decomposition-based implementation, we perform similar experiments with the SDO approach. Our analysis reveals that these metrics perform poorly compared to the Pareto-based approach. An in-depth discussion offers a strong motivating factor as to why these approaches fail for decomposition-based approaches, a limitation which derives from how the internal mechanism of MOEA/D works. Based on this limitation, we propose a new way to naturally incorporate semantic distances into MOEA/D methods in GP, specifically using the Weighted Sum, Tchebycheff and Penalty-based Boundary Intersection methods. Our proposed semantic-based method yields as good or better results on the average hypervolume of evolved Pareto-approximated fronts, Pareto optimal fronts as well as reducing dramatically the same number of solutions in the objective space. To the best of our knowledge, this is the first study of semantics in MOEA/D.

4.1.1 Semantics in NSGA-II and SPEA-II

Semantic Similarity-based Crossover MOGP

To incorporate semantics, in a MOGP paradigm, we first use the Semantic Similarity-based Crossover (SSC) originally proposed by Uy et al. [28] which, to the best of our knowledge, has been exclusively used in single-objective GP.

To use SSC in single-objective GP, a semantic distance must be computed first. Using Def. 2 from Chapter 2.2.3, this distance is obtained by computing the average of the absolute difference of values for every $in \in I$ between parent and offspring. If the distance value lies within a range, defined by one or two threshold values, then crossover is used to generate offspring. Because this condition may be hard to satisfy, the authors tried to encourage semantic diversity by repeatedly applying crossover up to 20 times. If after this, the condition is not satisfied, then crossover is executed as usual. The crossover operation is explained in Chapter 2.2.1.

SSC made a notable impact in GP, showing, for the first time, how semantic diversity can be promoted in continuous search spaces, with several subsequent papers following along this line [38, 134, 153]. We incorporate SSC in NSGA-II and SPEA2. However, the performance increase reported when adopting SSC in single-objective optimisation is not observed in MOGP, as we shall see in Chapter 4.3.1.

Semantic Distance as an additional criteriOn (SDO)

In this method, the crowding distance as discussed in Chapter 2.4.5 is replaced by a semantic-based crowding distance. A pivot p is selected as the individual in the first Pareto front which is furthest away from all other individuals v in that front. This distance is calculated using the crowding distance as discussed in Chapter 2.4.5. Once we have the pivot, we can compute the semantic differences of the pivot against all the other individuals in the population. Upper and lower semantic similarity bounds denoted as UBSS and LBSS, respectively, are used to promote semantic diversity within a range as shown in Eq. 4.1 or via a single bound as like in Eq. 4.2.

$$d(p, v_j) = \sum_{i=1}^{l} 1 \; if \; LBSS \leq |p(in_i) - v_j(in_i)| \leq UBSS \tag{4.1}$$

$$d(p, v_j) = \sum_{i=1}^{l} 1 \; if \; |p(in_i) - v_j(in_i)| \geq UBSS \tag{4.2}$$

The semantic-based crowding distance can also be used as an additional criterion to optimise. With the majority and minority classes (these benchmark problems are discussed

later in this chapter) serving as the first two objectives to optimise semantic distance is also treated as a third objective.

Pivot similarity Semantic-based Distance as an additional criteriOn (PSDO)

In Chapter 4.1.1 we discussed SDO which calculates the semantic distance between each individual and a pivot. To this effect, Equations 4.1 and 4.2 were used to determine if the semantic difference between a pivot and the individuals fall within a predefined range. This calculation naturally preferences programs which are semantically dissimilar to the pivot. However, given that the pivot is picked from the most sparse region of the search space this individual ought to be the most diverse as such an update a proposed update in the distance calculation is given in Equations 4.3 and 4.4, where the summed values is now taken away from the number of fitness cases. This method is referred to as Pivot similarity Semantic-based Distance as an additional criteriOn (PSDO) and uses Equations 4.3 and 4.4 in lieu of distances shown in Equations 4.1 and 4.2.

$$d(p, v_j) = l - \sum_{i=1}^{l} 1 \; if \; LBSS \leq |p(in_i) - v_j(in_i)| \leq UBSS \tag{4.3}$$

$$d(p, v_j) = l - \sum_{i=1}^{l} 1 \; if \; |p(in_i) - v_j(in_i)| \geq UBSS \tag{4.4}$$

4.1.2 Semantics in MOEA/D

Semantic Neighbourhood Ordering (SNO)

Similar to how previous semantic distance-based approaches have calculated semantic distance, this proposed method also makes use of a *pivot* [2,38]. Alg. 3 largely follows the steps outlined in Chapter 2.4.6 as seen in Alg. 1, however, two additional lines have been added (Alg. 3: Line 7-8). First, the crowding distance is calculated from the external population (Alg. 3: Line 7). The pivot is selected as an individual from the sparsest region from the non-dominated solutions of the external population EP (Alg. 3: Line 8). The external population EP represents non-dominated solutions across all generations, where newly dominated solutions at each generation are removed, as discussed in Chapter 2.4.6. The semantic distance is calculated as the absolute difference between the pivot p as selected from the external population EP and every individual v from the standard population P using Eq. 4.5

$$d(p, v) = \sum_{i=1} 1 \; if \; |p(in_i) - v(in_i)| < \text{UBSS} \tag{4.5}$$

As such, individuals that are semantically similar to the pivot will have a larger distance than those which are dissimilar. For the sake of simplicity, a single upper bound is chosen, referred to as Upper Bound Semantic Similarity (UBSS) and is used as a threshold value for the semantic similarity between the pivot and the individual under consideration. The use of bounds in continuous spaces is common to quantify semantic diversity as profusely used in the specialized literature [2, 38, 136].

The pseudocode for this approach, entitled Semantically Ordered Update (Alg. 4) replaces the standard update (Step 2.3 in Chapter 2.4.6) of the original MOEA/D approach and is called in Line 12 of Alg. 3. More specifically, the population is first subsetted using the neighbourhood reference table $B(i)$ and the population P (Alg. 4: Line 2). The semantic distances are computed between the pivot and each individual in the neighbourhood and these distances are assigned to each individual in the subsetted population NP (Alg. 4: Line 3). The neighbourhood is then sorted such that the most semantically dissimilar individuals are checked first (Alg. 4: Line 4). Once a preferable update occurs (Alg. 4: Line 7), the individual is replaced by the child program (Alg. 4: Line 8), the subroutine then exits (Alg. 4: Line 9) and the next child program is evaluated.

This process is visually represented in Fig 4.1, broken into six stages. Stages 1 - 3, show the typical selection of parents and the subsequent generation of an offspring individual (blue). Stages 4 - 6, demonstrate Alg. 4, showing how the neighbourhood (solutions close to A) are first ordered by the semantic similarity to the pivot (red), as selected from the external archive (green), and shows how the more preferable individual is replaced in a single replacement strategy.

4.2 Implementation Details

The use of benchmark problems has allowed the research community to test, validate and explain a plethora of evolutionary algorithms. In this work, we also adopt well-known, robust and tested benchmark problems used in other studies [154, 155] that will allow us to (i) test the algorithms used in this work, (ii) to use well-defined metrics that allow us to compare one method against another one, (iii) to allow us to explain why one particular method behaves better than others, (iv) to draw sound conclusions on the results reported in the following sections. Thus, for this study, the impact of semantics in MOGP are analysed using several unbalanced binary classification problems, taken from the UCI Machine Learning repository [156]. Table 4.2, adapted from [154], gives the details of all datasets used in this work. These classification problems have various degrees

Algorithm 3 Semantically Ordered MOEA/D

$\Lambda = \{\lambda^{i_1}, \lambda^{i_2}, ..., \lambda^{i_N}\}$ ▷ Create weight vectors
$P = \{x_1, x_2, ..., x_N\}$ ▷ Initialize population
for each $\lambda^i \in \Lambda$ **do**
    $B(i) = \{i_1, i_2, ..., i_T\}$ ▷ Define reference table
end for
repeat
    $CD_1 \leftarrow crowding\_distance(EP)$
    $pivot \leftarrow furthest\_point(CD_1)$
    for each $i \in \{1, 2, ..., N\}$ **do**
        $k, l \leftarrow$ Return random parent indices from B(i)
        $y \leftarrow$ Generate child program from x^k and x^l
        $P = update(y, i, pivot)$
    end for
    $EP \leftarrow$ Remove non-dominated solutions based on P
until Stopping criteria is met
return EP ▷ The non-dominated solutions from EP

Algorithm 4 Semantically Ordered Update

for each $i \in T$ **do**
    $NP \leftarrow$ Subset population using B(i) and P
    $NP \leftarrow compute\_semantics(pivot, NP)$
    $B(k) \leftarrow sort(B(i), NP)$
end for
for each $j \in B(k)$ **do**
    if $g(y|\lambda^j, z) > g(x^j|\lambda^j, z)$ **then**
        $x_j = y$ ▷ Single replacement
        return ▷ Exit subroutine
    end if
end for

of class imbalance, from 1:3 to 1:130, for the Ion and the $Abal_2$ data set, respectively. The 50-50% training/test sets also vary in size, with some having substantially more instances ($Abal_2$ has around 2,100 instances) to relatively low numbers of instances (Spect has around 133 instances, where around 27 are from the minority class). Moreover, these data sets range from low dimensionality ($Yeast_1$ has 8 features) to high dimensionality (Ion has 34 features). Finally, our data sets include binary and real-valued features. Thus, these data sets represent class imbalance problems of various degrees of difficulty, size, dimensionality and types of features reasonably well. Moreover, we carefully choose these benchmark problems so that our evaluations on the four semantic-based methods (SSC, SCD, SDO and SNO) and the three EMO approaches (NSGA-II, SPEA2 and MOEA/D) are not problem dependent.

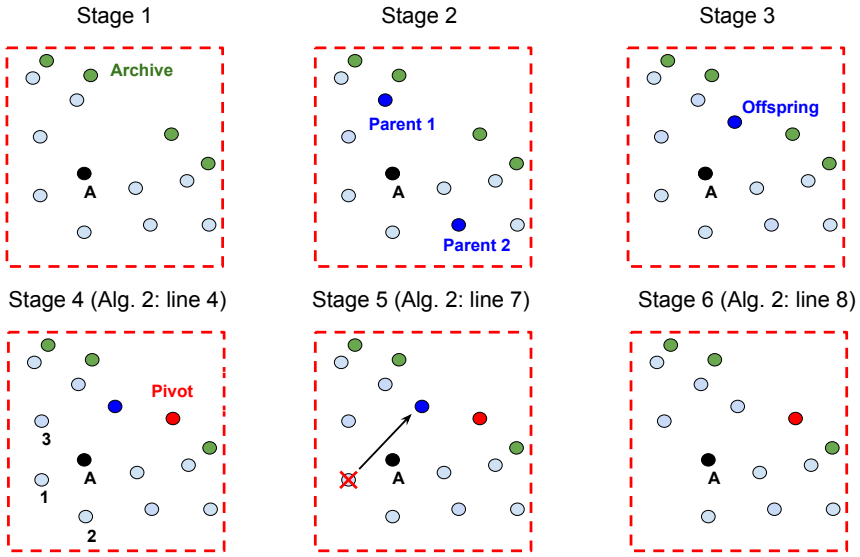


Figure 4.1: Demonstrating the semantic neighbourhood update. Stage 1 - 3 shows a typical selection of parents and the subsequent generation of an offspring individual (blue). Stages 4 - 6 show how the neighbourhood (solutions close to A) is first ordered by the semantic similarity to the pivot (red), as selected from the external archive (green), and how the more preferable individual is replaced in a single replacement strategy.

The terminal and function sets used in this work are as follows. The terminals are the problem features. The function set contains the most common arithmetic operators, namely $\mathcal{F} = \{+, -, *, /\}$, where the division operator is protected by returning the numerator when the denominator has a value of zero. The models evolved by GP map each input pattern in a dataset to a single output value. When the output of a GP model is greater than, or equal to, zero the pattern is labelled as part of the minority class, and it is labelled as a majority class pattern, otherwise.

The common way to measure fitness in a classification task is to use the overall classification accuracy: for binary classification, the four possible cases are shown in Table 4.3. Assuming the minority class is the positive class, the accuracy is given by Acc $= \frac{TP+TN}{TP+TN+FP+FN}$, where TP are the true positives, TN are the true negatives, FP are the false positives and FN are the false negatives. The drawback of using Acc alone is that it rapidly biases the evolutionary search towards the majority class [154] and other metrics like F1 score can bias optimization by failing to balance the trade-off of precision and recall. A better approach is to treat each class 'separately' using a MO approach. Two objectives considered are thus the true positive rate, given by TPR $= \frac{TP}{TP+FN}$, and the true negative rate given by TNR $= \frac{TN}{TN+FP}$. They measure the distinct accuracy for the minority (TPR) and majority class (TNR), respectively.

Table 4.2: Details on binary imbalanced classification data sets used in our research

Data set	Number of examples			Imb.	Features	
	Total	Positive	Negative	Ratio	No.	Type
Ion	351	126 (35.8%)	225 (64.2%)	1:3	34	Real
Spect	267	55 (20.6%)	212 (79.4%)	1:4	22	Binary
$Yeast_1$	1482	244 (16.5%)	1238 (83.5%)	1:6	8	Real
$Yeast_2$	1482	163 (10.9%)	1319 (89.1%)	1:9	8	Real
$Abal_1$	731	42 (5.75%)	689 (94.25%)	1:17	8	Real
$Abal_2$	4177	32 (0.77%)	4145 (99.23%)	1:130	8	Real
Climate	540	46 (8.5%)	494 (91.5%)	1:12	18	Real
Glass	214	70 (32.7%)	144 (67.3%)	1:3	10	Real
Parkinson's	197	48 (24.6%)	147 (75.4%)	1:4	22	Real
Wine	178	59 (33.1%)	119 (66.9%)	1:3	12	Real

Table 4.3: Confusion Matrix

	Predicted positive	Predicted negative
Actual positive	True Positive (TP)	False Negative (FN)
Actual negative	False Positive (FP)	True Negative (TN)

The experiments were conducted using a generational approach. Tree size was controlled by using a maximum number of nodes or a maximum final depth, whatever happens first. When a child tree exceeds any of these, the offspring is generated again until these conditions are satisfied. The parameters used are shown in Table 4.4. These include the use of different bounds, defined as Upper Bound Semantic Similarity (UBSS) = {0.25, 0.50, 0.75, 1.0} values and Lower Bound Semantic Similarity (LBSS) = {0.001, 0.01, 0.1} values, to compute the semantic distances defined in Eqs. 4.1, 4.2, 4.3 and 4.4. This results in conducting a thorough analysis: for each of the semantic-based approaches and for each of the datasets used, we have 16[1] independent results, each being the result of 50 independent runs. To obtain meaningful results, we carried out an extensive empirical experimentation (29,400 independent runs in total)[2]. On the other hand, only a single UBSS value is used for the semantic neighbourhood approach leading to less independent runs being tested (this gives a total of 1,440 independent runs)[3].

[1] 4 values for UBSS and 4 cases for LBBS: 3 values and 1 case where LBSS is not defined.

[2] 50 independent runs, 6 datasets, 3 semantic-based MOGP approaches (SSC, SCD, SDO), 16 different combinations of values for UBSS and LBSS, 2 canonical EMO methods (NSGA-II, SPEA2).

[3] 30 independent runs, 8 data sets, 3 semantic-based MOEA/D approaches and 3 canonical MOEA/D methods (the Weighted Sum, Tchebycheff and Penalty-based Boundary Intersection.

Table 4.4: Summary of parameters used in our experiments.

Parameter	*Value*
Population Size	500
Generations	50
Type of Crossover	90% internal nodes, 10% leaves
Crossover Rate	0.60
Type of Mutation	Subtree
Mutation Rate	0.40
Initialisation Method	Ramped half-and-half
Initialisation Depths:	
Initial Depth	1 (Root = 0)
Final Depth	5
Maximum Length	800 nodes
Maximum Final Depth	8
Independent Runs	50 (Ch. 4.3.1), 30 (Ch. 4.3.2)
Neighbourhood size	20
Scalar Optimisation	{ Weighted Sum, Tchebycheff & PBI }
PBI Theta Value	0.1
Semantic Thresholds	UBSS = {0.25, 0.5, 0.75, 1.0}
(Ch. 4.3.1)	LBSS = {-, 0.001, 0.01, 0.1}
Semantic Thresholds (Ch. 4.3.2)	UBSS = 0.5

4.3 Results

4.3.1 Semantic Approaches for Pareto-based Optimisation

SSC, SDO and PSDO: Comparison

In order to compare the approaches used in this work, we use the hypervolume of the evolved Pareto approximations as a performance measure [157]. For bi-objective problems, as the binary highly unbalanced classifications problems used in this work are, the hypervolume of a set of points in objective space, using reference point $(0, 0)$, is easily computed as the sum of the areas of all non-overlapping rectangles fitted under each point. The reference point $(0, 0)$ is required for the hypervolume calculation and as we wish to maximise each of the objectives we select this point as the lowest value that either the TPR or TNR can take. The hypervolume was chosen because it is one of the most widely used performance indicators in the EMO literature and it allows us to compare the performance of EMO approaches. Furthermore, we also computed the accumulated Pareto-optimal (PO) front with respect to 50 runs: the set of non-dominated solutions after merging all 50 Pareto-approximated fronts.

In our work, we use payoff tables to demonstrate which semantic approach (SSC, SDO or PSDO) is significantly better than its canonical form. The motivation behind using these payoff tables is to easily summarize the vast amounts of results obtained. To demonstrate this, Table 4.5 which contains just a single dataset for NSGA-II (Ion

Table 4.5: Average hypervolume (± std. deviation) and last run Pareto Front for NSGA-II SDO, NSGA-II PSDO and NSGA-II SSC methods for Ion dataset only. Each is compared against NSGA-II, results of which are found in Table 4.6

	Hypervolume							
	Average UBSS				PO Front UBSS			
LBSS	0.25	0.5	0.75	1.0	0.25	0.5	0.75	1.0
NSGA-II SDO								
–	**0.860 ± 0.033**+	**0.869 ± 0.037**+	**0.869 ± 0.033**+	**0.845 ± 0.057**+	0.948	0.958	0.962	0.950
0.001	**0.817 ± 0.087**+	**0.819 ± 0.104**+	**0.857 ± 0.057**+	**0.861 ± 0.047**+	0.942	0.957	0.954	0.958
0.01	**0.825 ± 0.084**+	**0.843 ± 0.073**+	**0.861 ± 0.045**+	**0.861 ± 0.038**+	0.946	0.956	0.957	0.944
0.1	**0.846 ± 0.070**+	**0.848 ± 0.068**+	**0.844 ± 0.075**+	**0.864 ± 0.044**+	0.950	0.956	0.953	0.960
NSGA-II PSDO								
–	0.794 ± 0.100	**0.811 ± 0.084** +	0.823 ± 0.091	0.795 ± 0.105	0.904	0.932	0.945	0.939
0.001	**0.867 ± 0.035**+	**0.874 ± 0.029**+	**0.880 ± 0.036**+	**0.873 ± 0.045**+	0.959	0.952	0.965	0.945
0.01	**0.852 ± 0.050**+	**0.867 ± 0.051**+	**0.880 ± 0.031**+	**0.867 ± 0.050**+	0.947	0.950	0.944	0.949
0.1	**0.853 ± 0.062**+	**0.869 ± 0.048**+	**0.875 ± 0.051**+	**0.872 ± 0.049**+	0.941	0.951	0.956	0.938
NSGA-II SSC								
–	0.761 ± 0.108	0.749 ± 0.161	0.763 ± 0.152	0.744 ± 0.137	0.941	0.937	0.951	0.949
0.001	0.765 ± 0.134	0.753 ± 0.124	0.699 ± 0.188	0.803 ± 0.103	0.954	0.935	0.928	0.946
0.01	0.760 ± 0.125	0.751 ± 0.123	0.710 ± 0.161	0.802 ± 0.104	0.947	0.929	0.928	0.947
0.1	0.775 ± 0.095	0.738 ± 0.184	0.746 ± 0.141	0.778 ± 0.099	0.957	0.951	0.945	0.936

Table 4.6: Average hypervolume (± std. deviation) and last run Pareto Front for NSGA-II and SPEA2 for 50 independent runs.

Dataset	NSGA-II Hypervolume		SPEA2 Hypervolume	
	Average	PO Front	Average	PO Front
Ion	0.766 ± 0.114	0.938	0.786 ± 0.094	0.948
Spect	0.534 ± 0.024	0.647	0.544 ± 0.032	0.659
$Yeast_1$	0.838 ± 0.011	0.876	0.838 ± 0.008	0.877
$Yeast_2$	0.950 ± 0.009	0.976	0.946 ± 0.015	0.978
$Abal_1$	0.847 ± 0.058	0.961	0.832 ± 0.078	0.960
$Abal_2$	0.576 ± 0.122	0.842	0.544 ± 0.147	0.834

Table 4.7: Payoff tables for canonical NSGA-II, NSGA-II SDO, NSGA-II PSDO and NSGA-II SSC for each of the 6 data sets

Ion

	NSGA-II	SDO	PSDO	SSC
NSGA-II	-	0	0	0
SDO	16	-	2	15
PSDO	13	3	-	14
SSC	0	0	0	-

Spect

	NSGA-II	SDO	PSDO	SSC
NSGA-II	-	0	0	0
SDO	16	-	8	16
PSDO	16	3	-	15
SSC	0	0	0	-

Yeast_1

	NSGA-II	SDO	PSDO	SSC
NSGA-II	-	0	0	0
SDO	16	-	2	16
PSDO	16	1	-	16
SSC	0	0	0	-

Yeast_2

	NSGA-II	SDO	PSDO	SSC
NSGA-II	-	0	0	0
SDO	16	-	2	16
PSDO	16	4	-	16
SSC	0	0	0	-

Abal_1

	NSGA-II	SDO	PSDO	SSC
NSGA-II	-	0	0	0
SDO	12	-	10	12
PSDO	6	1	-	7
SSC	0	0	0	-

Abal_2

	NSGA-II	SDO	PSDO	SSC
NSGA-II	-	0	0	2
SDO	12	-	1	15
PSDO	16	7	-	16
SSC	0	0	0	-

dataset), will be used to fully explain the process, but in the Appendix, the full tables, containing six datasets (Ion, Spect, Yeast_1, Yeast_2, Abal_1 and Abal_2) for both NSGA-II and SPEA2, have been reproduced (Tables A.2 and A.3). Before discussing the payoff tables we will first discuss how to interpret the results as shown in Tables 4.6 (results yielded by canonical EMO approaches) and 4.5 (results obtained by semantic-based EMO approaches on the Ion dataset).

As such, Table 4.5 reports both, the average hypervolume over 50 runs (columns 2-5, from left to right) and also the hypervolume of the accumulated PO front with respect to all 50 runs (columns 6-9). To obtain a statistically sound conclusion, a series of Wilcoxon rank-sum tests were run on the average hypervolume results. To account for the problem of multiple comparisons that arose from testing the canonical method 16 times for each data set, a Bonferroni correction $\frac{\alpha}{m} = 3.125 \times 10^{-3}$ was used where $\alpha = 0.05$. These statistically significant differences are highlighted in boldface, to easily distinguish from non-significant results. Moreover, in this table, the symbols "+" and "–", indicate that the results of a given semantic-based approach are significantly better or worse than those found by the canonical NSGA-II (Table 4.6), using the Wilcoxon rank-sum test.

We can now compare the semantic-based methods, introduced in Section 4.1.1, against their corresponding canonical EMO algorithms. As previously discussed, due to the significant amount of results obtained and shown in the appendix, we summarised our findings in a series of payoff matrices to show which strategies have significantly better values or 'wins' *vs.* the other methods. These payoff matrices are shown in Tables 4.7 and 4.8 for semantic-based methods using NSGA-II and SPEA2, respectively.

The payoff tables can be read as follows: the strategies of the row index are compared

Table 4.8: Payoff tables for canonical SPEA2, SPEA2 SDO, SPEA2 PSDO and SPEA2 SSC for each of the 6 data sets.

Ion

	SPEA2	SDO	PSDO	SSC
SPEA2	-	0	0	0
SDO	16	-	0	16
PSDO	16	0	-	16
SSC	0	0	0	-

Spect

	SPEA2	SDO	PSDO	SSC
SPEA2	-	0	0	0
SDO	14	-	0	16
PSDO	16	0	-	16
SSC	0	0	0	-

Yeast_1

	SPEA2	SDO	PSDO	SSC
SPEA2	-	0	0	0
SDO	16	-	1	16
PSDO	16	0	-	16
SSC	0	0	0	-

Yeast_2

	SPEA2	SDO	PSDO	SSC
SPEA2	-	0	0	0
SDO	16	-	0	16
PSDO	16	0	-	16
SSC	0	0	0	-

Abal_1

	SPEA2	SDO	PSDO	SSC
SPEA2	-	0	0	0
SDO	16	-	0	16
PSDO	13	0	-	13
SSC	0	0	0	-

Abal_2

	SPEA2	SDO	PSDO	SSC
SPEA2	-	0	0	0
SDO	15	-	0	16
PSDO	16	0	-	16
SSC	0	0	0	-

against the strategies of the column index and for each LBSS and UBSS setting which is significantly better for the column strategy counts as one 'win' towards the count. For example, SDO *vs.* NSGA-II for the Ion data set is significantly better for all settings of LBSS and UBSS (as seen in Table 4.5 originally, columns 2-5, highlighted in bold) and as such has 16 'wins' overall (top left-hand side of Table 4.7). Likewise, PSDO *vs.* NSGA-II for the Ion data set is significantly better for 13 settings of LBSS and UBSS (columns 2-5, highlighted in bold), and SSC *vs.* NSGA-II is not significantly better for any (columns 2-5, here no values have been highlighted in bold). The payoff tables have been colour-coded as such with solid black denoting that the strategy is the best overall in terms of the number of 'wins' and light grey being the worst overall.

Given our understanding of the payoff tables, we can explain the results in full. The two methods that use semantic distance as a criteria strategy, SDO and PSDO, outperformed their respective canonical counterparts (NSGA-II and SPEA2) with the exception of Abal_1 which had a large number of settings that produced no significant difference in the hypervolume averages for canonical NSGA-II. Additionally for Abal_1, NSGA-II SDO had the greatest number of wins over other strategies and results for NSGA-II PSDO were comparatively mixed. We can see that SSC did not provide any 'wins' over NSGA-II.

If we look at SPEA2 we see the same general trend; both SDO and PSDO outperform the canonical SPEA2 approach and SCC does not produce any wins over the canonical SPEA2 approach. In Appendix A, we provide a detailed comparison, looking specifically at comparing the three semantic approaches to each other, as seen in the payoff tables, however, the main takeaway is that the semantic distance-based approaches, SDO and PSDO, produce better results when compared to their canonical counterparts.

4.3.2 Semantic Approaches for Decomposition-based Optimisation

SDO in MOEA/D: Limitations

To highlight some of the limitations of the SDO approach we also look at a decomposition approach known as Multi-Objective Evolutionary Algorithm with Decomposition (MOEA/D) [46]. As previously discussed, SDO uses a pivot which is selected from the sparsest region of the Pareto front and preferences individuals that have the greater semantic distance from that point. With MOEA/D we decompose the optimisation problem into a set of scalar optimisation problems (see Chapter 2.4.6 for details). A scalar optimisation function g, along with a uniform distribution of weight vectors λ_i are used to define each sub-problem. It is important to note that each sub-problem relies only on neighbouring sub-problems for evolution, that is crossover, mutation and selection only occur for individuals within a given neighbourhood. In the original MOEA/D approach, three aggregation approaches are discussed, namely the Weighted sum, Tchebycheff and Penalty-based Boundary Intersection (PBI) approach. To demonstrate that our approach is best suited to Pareto dominance-based approaches, we use one of these three MOEA/D methods. As such, the Tchebycheff approach[4] was selected as (i) it does not require any additional parameters, unlike the PBI approach and (ii) since it tended to perform as well as or better than the Weighted Sum approach in preliminary baseline tests. The scalar optimisation of the Tchebycheff approach is explained in Chapter 2.4.6.

Only one additional step is required to implement the SDO approach in MOEA/D. The inclusion of this step can occur at any stage prior to implementing the aggregation and selection process but after mutation and crossover operations have been performed. Since the dominance relation is not naturally used in the MOEA/D algorithm, we instead create a dummy population. That is, the parent and offspring populations are joined together and the pivot is derived from this joint population as is the case of the framework used in our NSGA-II and SPEA2 approaches. Once the semantic distances have been calculated, this joint population is discarded and the MOEA/D algorithm works as expected.

Using the same six datasets from Chapter 4.3.1, the average and accumulated hypervolumes results for the canonical MOEA/D approach are produced in Table 4.9 and results for the SDO approach are produced in Table 4.10. A fixed neighbourhood size of 20 was used for all experiments. A Friedman test is used to test the null hypothesis that the median performance of all groups in the same block are the same. We reject the null hypothesis at the $\alpha = 0.01$ significance level[5], concluding that the median performance

[4]Later, in Chapter 4.3.2 all three methods are compared when analysing the semantic neighbourhood ordering approach.

[5]The use of a Friedman test and different significance levels for alpha stems from this analysis coming from the journal article in Applied Soft Computing [2]. The SDO data used is the same in both papers. Using a Wilcoxon text with higher significance for alpha leads to similar analysis.

for all groups differs. When we perform multiple comparisons against the baseline results (using MOEA/D only), from Table 4.9, and each of the SDO configurations, from Table 4.10, we found that the baseline method outperformed the SDO method.

In Table 4.10 when looking at the average and accumulated hypervolumes, barring Abal_2, we can see that none of the datasets produces better results for SDO when compared to their respective canonical results, shown in Table 4.9. In fact, it was found that for these data sets the canonical approach performed better. The results for Abal_2, highlighted in boldface, are performing better for all UBSS and LBSS values. The seemingly conflicting results are due to the high level of variance in the canonical results, where the standard deviation is $\pm$ 0.100. This unexplained variance is accounted for in the Friedman test.

To understand why MOEA/D fails to produce similar results to NSGA-II and SPEA2 when using SDO, it is important to highlight how these algorithms differ in terms of searching for solutions in objective space. Both NSGA-II and SPEA2 rely on the dominance relationship to sort preferential solutions to be retained, MOEA/D instead decomposes the multi-objective problem into sub-problems and each sub-problem is assigned a weight vector which represents a specific direction or location for that particular sub-problem to be optimised to. The weight vectors are typically initialised so that they are evenly distributed, with the ultimate aim of producing a representative spread of solutions in objective space. Fig. 4.2 illustrates the distinction between decomposition (left-hand side diagram) and Pareto dominance (right-hand side diagram) based approaches. Both diagrams have three numbered regions separated by a dashed line. Region 1 represents the initially randomised solutions, Region 2 represents the solutions midway through optimisation and Region 3 represents solutions towards the end of the optimisation process when maximising a potential solution.

On the left diagram, solutions have been colour-coded to highlight how they optimise towards particular localities in the objective space. For simplicity's sake, only 3 neighbourhoods have been represented and we will assume these three neighbourhoods are from a subset of a much larger set of neighbourhoods such that none of the individuals in each neighbourhood overlap. Each neighbourhood has been colour-coded with blue, yellow and red circles. Even though the solutions are initially mixed in objective space (left diagram, Region 1), as the optimisation progresses, newly produced offspring will converge towards their respective locations in objective space based on the initial weighting (left diagram, Region 2). In the final stages, the offspring will likely converge on a single point, leading to duplication of solutions (left diagram, Region 3).

In the Pareto dominance approach (right diagram, Region 1), the selection process is determined by the dominance relation. For simplicity's sake, only non-dominated solutions with dominance Rank 0 have been highlighted (purple circle) along with the

Table 4.9: Average (± standard deviation) hypervolume of evolved Pareto-approximated fronts and PO fronts for MOEA/D over 50 independent runs.

Dataset	MOEA/D Hypervolume	
	Average	PO Front
Ion	0.823 ± 0.031	0.932
Spect	0.537 ± 0.024	0.652
$Yeast_1$	0.837 ± 0.008	0.877
$Yeast_2$	0.945 ± 0.013	0.978
$Abal_1$	0.808 ± 0.097	0.959
$Abal_2$	0.602 ± 0.100	0.798

Table 4.10: Average (± standard deviation) hypervolume of evolved Pareto-approximated fronts and PO fronts for the MOEA/D semantic-based method for SDO with over 50 independent runs. Bold indicates better performance compared to the baseline MOEA/D results reported in Table 4.9.

		Hypervolume							
		Average UBSS				PO Front UBSS			
	LBSS	0.25	0.5	0.75	1.0	0.25	0.5	0.75	1.0
		MOEA/D SDO							
Ion	–	0.689 ± 0.039	0.702 ± 0.038	0.708 ± 0.032	0.691 ± 0.030	0.848	0.847	0.850	0.850
	0.001	0.753 ± 0.032	0.738 ± 0.038	0.727 ± 0.033	0.729 ± 0.031	0.885	0.879	0.866	0.860
	0.01	0.747 ± 0.039	0.725 ± 0.035	0.735 ± 0.036	0.727 ± 0.031	0.906	0.880	0.872	0.859
	0.1	0.749 ± 0.033	0.749 ± 0.035	0.734 ± 0.033	0.734 ± 0.032	0.885	0.892	0.865	0.869
Spect	–	0.472 ± 0.028	0.474 ± 0.027	0.474 ± 0.024	0.473 ± 0.024	0.601	0.604	0.599	0.585
	0.001	0.439 ± 0.047	0.441 ± 0.038	0.461 ± 0.040	0.458 ± 0.030	0.597	0.589	0.612	0.575
	0.01	0.456 ± 0.040	0.436 ± 0.050	0.447 ± 0.037	0.447 ± 0.035	0.588	0.582	0.611	0.609
	0.1	0.451 ± 0.051	0.444 ± 0.043	0.453 ± 0.033	0.449 ± 0.028	0.603	0.588	0.606	0.578
$Yeast_1$	–	0.748 ± 0.036	0.743 ± 0.037	0.746 ± 0.030	0.749 ± 0.032	0.848	0.845	0.846	0.843
	0.001	0.764 ± 0.030	0.767 ± 0.035	0.776 ± 0.028	0.781 ± 0.027	0.850	0.848	0.850	0.852
	0.01	0.755 ± 0.040	0.765 ± 0.034	0.764 ± 0.028	0.768 ± 0.031	0.847	0.847	0.848	0.849
	0.1	0.758 ± 0.032	0.754 ± 0.041	0.743 ± 0.037	0.755 ± 0.037	0.846	0.846	0.840	0.844
$Yeast_2$	–	0.833 ± 0.075	0.846 ± 0.064	0.825 ± 0.067	0.825 ± 0.060	0.968	0.959	0.958	0.956
	0.001	0.772 ± 0.092	0.800 ± 0.080	0.833 ± 0.077	0.859 ± 0.075	0.960	0.959	0.968	0.969
	0.01	0.778 ± 0.091	0.771 ± 0.073	0.794 ± 0.087	0.801 ± 0.083	0.957	0.952	0.963	0.960
	0.1	0.770 ± 0.083	0.778 ± 0.081	0.755 ± 0.086	0.766 ± 0.076	0.950	0.948	0.953	0.953
$Abal_1$	–	0.705 ± 0.087	0.707 ± 0.099	0.700 ± 0.100	0.698 ± 0.110	0.938	0.927	0.932	0.941
	0.001	0.719 ± 0.072	0.736 ± 0.089	0.765 ± 0.068	0.794 ± 0.066	0.920	0.928	0.931	0.944
	0.01	0.678 ± 0.078	0.722 ± 0.073	0.738 ± 0.076	0.769 ± 0.068	0.882	0.908	0.899	0.936
	0.1	0.678 ± 0.086	0.672 ± 0.086	0.692 ± 0.066	0.720 ± 0.080	0.898	0.894	0.884	0.895
$Abal_2$	–	**0.672 ± 0.061**	**0.665 ± 0.046**	**0.666 ± 0.053**	**0.658 ± 0.075**	**0.844**	**0.840**	**0.836**	**0.851**
	0.001	**0.666 ± 0.042**	**0.660 ± 0.041**	**0.676 ± 0.043**	**0.676 ± 0.034**	**0.851**	**0.829**	**0.852**	**0.838**
	0.01	**0.666 ± 0.035**	**0.664 ± 0.035**	**0.676 ± 0.044**	**0.663 ± 0.039**	**0.846**	**0.834**	**0.849**	**0.815**
	0.1	**0.672 ± 0.046**	**0.661 ± 0.036**	**0.652 ± 0.040**	**0.656 ± 0.046**	**0.850**	**0.831**	**0.830**	**0.828**

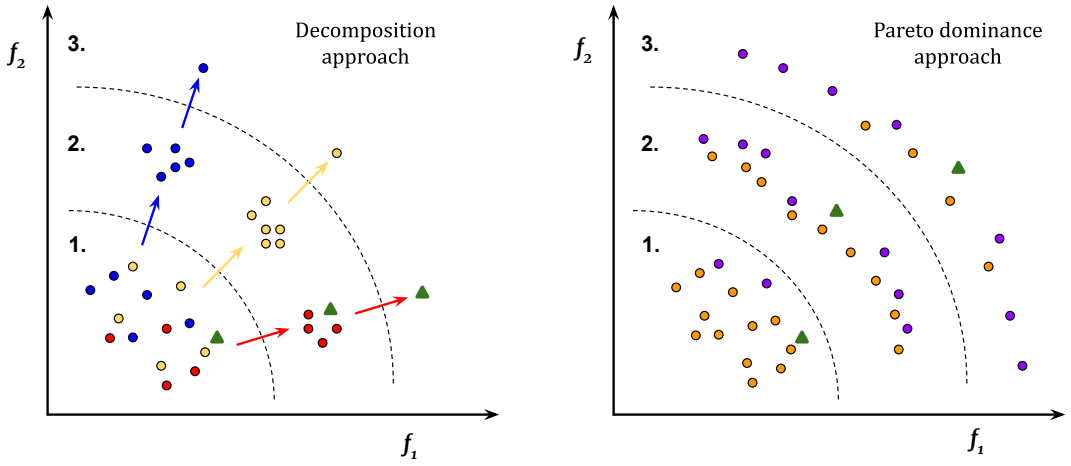


Figure 4.2: Diagram illustrating search behaviour for both decomposition and Pareto dominance-based approaches when maximising a solution. Numbers 1, 2 and 3 denote the typical spread of solutions in objective space at initial, intermediate and latter generations respectively. The green triangle represents a typical pivot selection.

dominated solutions with dominance rank greater than 0 (orange circle). Under the assumption that diversity is actively being promoted, as the optimisation progresses, the solutions will become more evenly spread with the dominated solutions reducing over time and subsequently more non-dominated solutions being assigned within the population (right diagram, Regions 2 and 3).

This illustrated behaviour allows for an intuitive distinction between the search mechanism in the Pareto dominance-based algorithms and the decomposition algorithm. In the Pareto dominance-based approach new solutions are free to occupy any location in the objective space as long as they satisfy the dominance relationship and crowding distance criteria, and therefore these algorithms search the objective space globally. However, in the MOEA/D algorithm, the search is localised, for a given sub-problem in conjunction with its weights and once the algorithm converges the diversity of solutions are constrained by the initialised weights.

Given that the pivot behaves as a point of semantic attraction, it is possible to show how this is problematic: the semantic relationship between individuals in a specific neighbourhood and the pivot may not provide beneficial updates as the selection process is determined by the aggregation function (Tchebycheff approach in this example), which behaves as a localised search mechanism when the neighbourhood size is small relative to the overall population size. In other words, assuming the neighbourhood is far away from the pivot, the direction a particular sub-problem in objective space is being optimised towards, will likely not correspond to the direction of attraction to the pivot in semantic

space and as such the aggregation function negates any benefit from drawing a semantic relationship to the pivot. To get an intuitive understanding of this, if we look at the left diagram of Fig. 4.2, we can see that if the pivot (green triangle) is always selected in the bottom left neighbourhood (red circles), then it will not be possible to draw individuals to this neighbourhood from the other neighbourhoods (blue and yellow circles) and subsequently using the semantic relationship between the pivot and these individuals is redundant. Furthermore, as the algorithm converges, the number of duplicates in each neighbourhood increase significantly meaning the semantic distance will be the same for all individuals in a particular neighbourhood.

While each individual sub-problem engages in a localised search of the objective space, when we consider all sub-problems in unison MOEA/D is effectively searching globally. The real issue arises from the trade-off of exploitation versus exploration when selecting the neighbourhood size relative to the population size and specifically its incompatibility when using a pivot in this manner. Selecting a neighbourhood size of 20 was required in the baseline models to garner reasonable results (i.e., preferencing local search rather than global exploration) and when we compare Tables 4.6 (results using NSGA-II and SPEA2) and 4.9 (results using MOEA/D), we can see these results are comparatively similar. If we increased the neighbourhood size so that it was substantially larger, this would likely improve exploration and make the pivot mechanism more effective, however, doing so would come at the cost of the exploitative aspects of the algorithm. It is, however, important to note, that using semantic distance can still be useful in decomposition approaches when we operate internally within the neighbourhood structure.

SNO in MOEA/D: Hypervolume

Next, we look at the results for the semantic neighbourhood ordering approach in MOEA/D, which overcomes the limitation as discussed previously in this chapter. In this chapter, we use four of the same datasets as before, but aba1$_l$ and abal$_2$ have been dropped due to i) the run time associated with these datasets and ii) the higher level of variance found in the canonical MOEA/D made drawing sound conclusions somewhat more difficult in the SDO comparison. These two datasets have been replaced with four new datasets; Climate, Glass, Parkinson's and Wine, shown in Table 4.2.

Tables 4.11 and 4.12 report the average hypervolume over 30 independent runs for the external population. We also computed the accumulated Pareto optimal (PO) front for 30 runs from the external population, that is the set of non-dominated solutions after merging all 30 Pareto-approximated fronts. These results were gathered for both the canonical MOEA/D framework (Table 4.11) and for the framework which incorporates the semantic neighbourhood ordering (Table 4.12), as outlined in Chapter 4.1.2. The

Weighted Sum (WGT), Tchebycheff (TCH) and Penalty-based Boundary Intersection (PBI) scalar optimisation functions were tested and for each framework, with only the majority and minority classes being considered for objectives. In order to obtain a statistically sound conclusion, the Wilcoxon rank-sum test was run with a significance level of $\alpha = 0.05$ on the average hypervolume results. The statistically significant differences between the two frameworks are highlighted in boldface in each of the respective tables. A row-wise comparison for each of the data sets shows that the semantically ordered approach for Ion, Climate and Glass produced significantly better results when compared to the canonical approach for the hypervolume averages, barring the WGT approach for Glass, which while larger, was not significant. Wine had no statistical change in this regard. The canonical approach was universally better only in the case of the Spect dataset when compared against the other three methods. We will explain these results in more detail in the following section. The results for Yeast_1, Yeast_2 and Parkinson's datasets, are mixed, however, when a result was statistically significant for these datasets, the difference in hypervolume between the canonical and semantic approaches was typically less than 0.01, whereas the results for the other datasets the difference in hypervolume was generally greater.

At a more granular level, when comparing the semantic ordered approach against its canonical form, there are 3 out of 8 (Ion, Climate, Glass) instances where WGT is significantly better, there are 3 out of 8 instances (Ion, Climate, Glass) where TCH is significantly better, and 4 out of 8 instances (Ion, Yeast_2, Climate, Glass) where PBI is significantly better. Let's turn our attention, to when the canonical approach outperforms the semantic ordered approach. We can see that there are 3 out of 8 instances where WGT is significantly better, 1 out of 8 instances (Spect) where TCH is significantly better, and 3 out of 8 (Spect, Yeast_1 and Parkinson's) where PBI was significantly better in the canonical form. The accumulated PO front largely conforms with the average hypervolume results, having as good or better performance for the semantic method, although some notable decreases were observed for Climate (TCH and PBI). In general, TCH performs as well or better for the semantic ordering approach with WGT and PBI showing mixed results depending on the datasets.

SNO in MOEA/D: Analysis Of Objective Space

We have seen some promising results when promoting diversity in MOEA/D algorithms. In other cases, the results are mixed, where canonical MOEA/D methods perform better in a few data sets. To better understand this, we perform a detailed analysis in the objective space. This is shown in Fig. 4.3 and Fig. 4.4.

The left-hand plots show the canonical approach while the right-hand plots show

Table 4.11: Average (± standard deviation) hypervolume of evolved Pareto-approximated fronts and PO fronts for MOEA/D with WGT, TCH and PBI methods of decomposition for 30 runs.

Data set	WGT Hypervolume		TCH Hypervolume		PBI Hypervolume	
	Average	PO Front	Average	PO Front	Average	PO Front
Ion	0.819 ± 0.032	0.926	0.840 ± 0.034	0.934	0.817 ± 0.038	0.921
Spect	**0.537 ± 0.029**	0.641	**0.538 ± 0.028**	0.635	**0.536 ± 0.025**	0.648
$Yeast_1$	**0.835 ± 0.007**	0.867	0.838 ± 0.010	0.874	**0.836 ± 0.006**	0.873
$Yeast_2$	**0.951 ± 0.009**	0.973	0.948 ± 0.008	0.975	0.939 ± 0.024	0.976
Climate	0.664 ± 0.082	0.898	0.645 ± 0.084	0.866	0.603 ± 0.094	0.788
Glass	0.781 ± 0.051	0.903	0.807 ± 0.051	0.925	0.810 ± 0.048	0.917
Parkinson's	0.810 ± 0.041	0.941	0.788 ± 0.054	0.932	**0.779 ± 0.04**8	0.942
Wine	0.960 ± 0.026	1.000	0.959 ± 0.031	1.000	0.954 ± 0.037	0.999

Table 4.12: Average (± standard deviation) hypervolume of evolved Pareto-approximated fronts and PO fronts for **semantically ordered neighbourhood** MOEA/D with WGT, TCH and PBI methods of decomposition for 30 runs.

Data set	WGT Hypervolume		TCH Hypervolume		PBI Hypervolume	
	Average	PO Front	Average	PO Front	Average	PO Front
Ion	**0.840 ± 0.026**	0.932	**0.854 ± 0.018**	0.927	**0.849 ± 0.020**	0.937
Spect	0.523 ± 0.020	0.605	0.517 ± 0.021	0.605	0.520 ± 0.021	0.615
$Yeast_1$	0.829 ± 0.006	0.863	0.832 ± 0.006	0.874	0.834 ± 0.005	0.873
$Yeast_2$	0.946 ± 0.009	0.972	0.946 ± 0.009	0.972	**0.949 ± 0.009**	0.972
Climate	**0.774 ± 0.056**	0.906	**0.730 ± 0.088**	0.909	**0.763 ± 0.068**	0.908
Glass	0.799 ± 0.044	0.907	**0.831 ± 0.037**	0.916	**0.834 ± 0.046**	0.923
Parkinson's	0.803 ± 0.035	0.917	0.802 ± 0.040	0.916	0.814 ± 0.034	0.928
Wine	0.961 ± 0.019	0.999	0.965 ± 0.021	0.997	0.958 ± 0.023	0.998

the semantic approach. For the semantic approach, the selected pivots in the external population for every generation are displayed and denoted with red cross symbols ('x'). All plots are from the same single-seeded run and have been selected randomly to avoid having a biased analysis of results. We focus our attention on the Ion (Fig. 4.3) and $Yeast_1$ (Fig. 4.4) data sets. The rest of the images associated with this analysis have been produced in full in Appendix A, with Fig. A.1 - A.8. In general, the additional datasets conform to this analysis, although, with the Climate and Glass dataset, for this particular run, it is less visually clear.

SNO in MOEA/D: Analysis Of Duplication

We can see by comparing the canonical method for the Ion data set (Fig. 4.3, left-hand column) that there are a number of discontinuous regions along the border of the feasible space. For instance, a notably large region can be observed in the bottom-right-hand corner of the TCH approach (left-middle). These areas of sparsity correspond to the pivot selection, with the pivot being selected at either the beginning or end of where

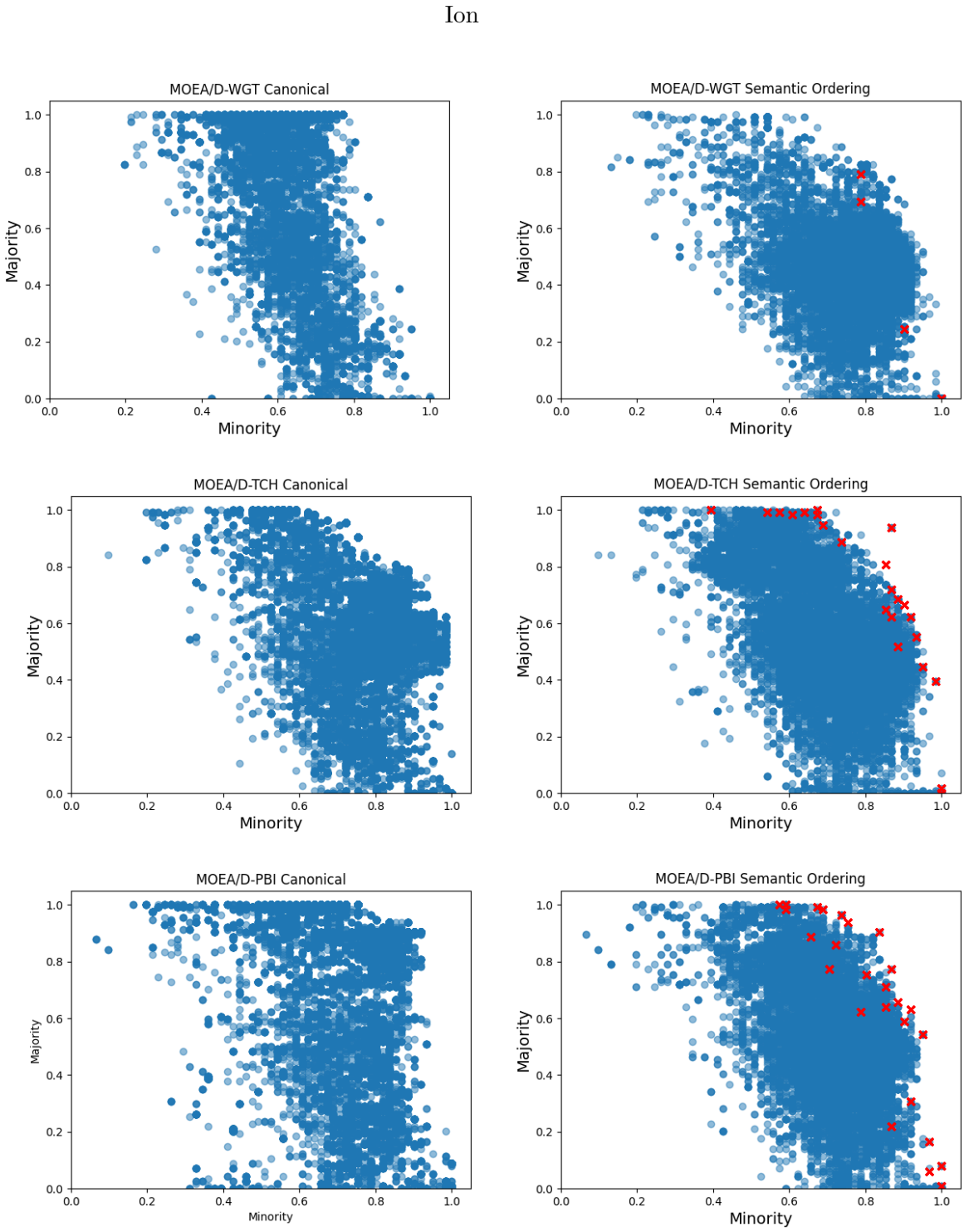


Figure 4.3: Solutions for every generation for the Ion dataset for WGT, TCH and PBI approaches. Red cross symbols ('x') denote pivot selections from the external population for all generations for a single run.

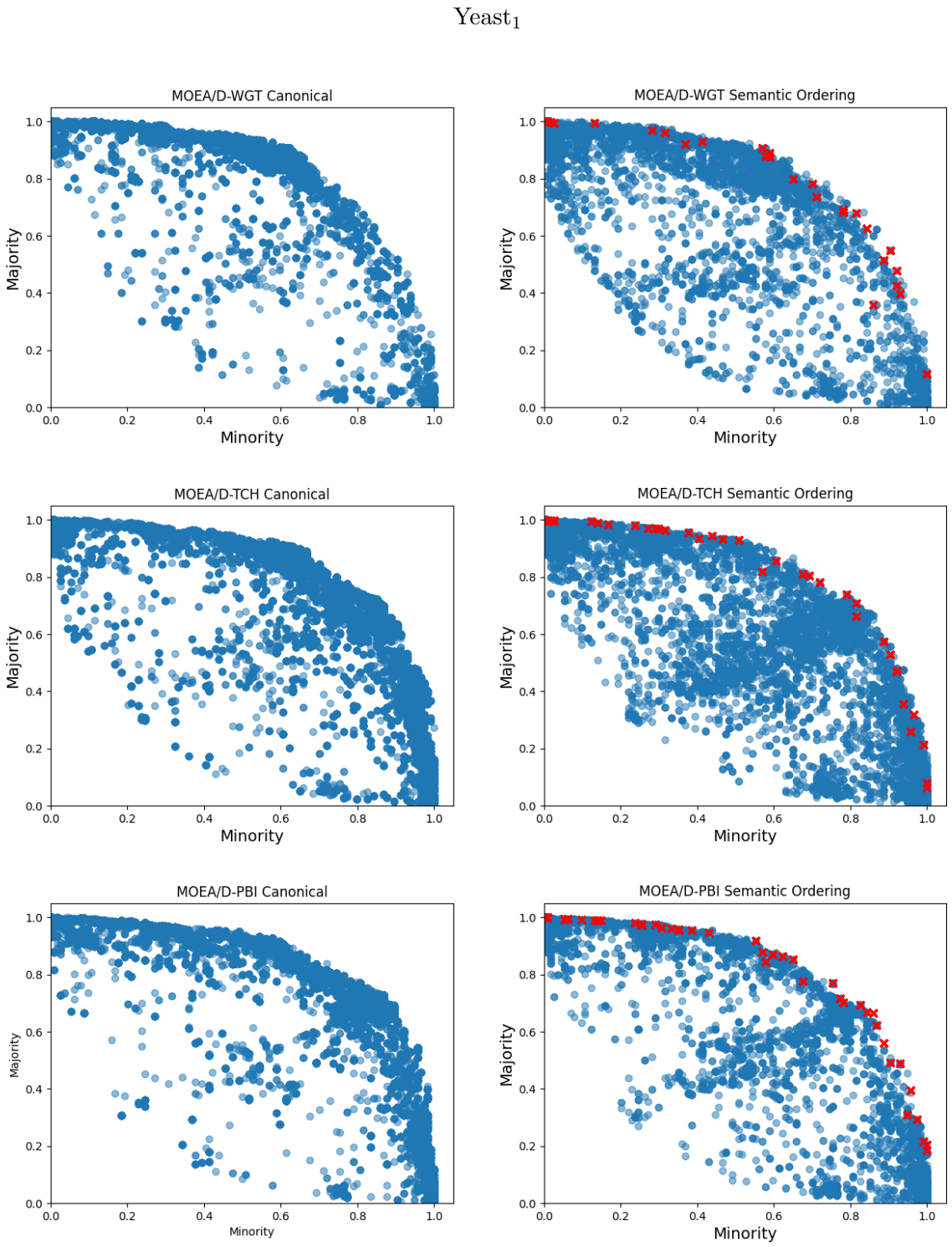


Figure 4.4: Solutions for every generation for the Yeast$_1$ dataset for WGT, TCH and PBI approaches. Red cross symbols ('x') denote pivot selections from the external population for all generations for a single run.

these discontinuities occur. When we look at the solutions produced for the same data set but now using the semantic-based method and using the same MOEA/D algorithm, right-hand column, we can see that discontinuities observed in the canonical method are no longer as prevalent. In contrast, if we look at the Yeast_1 (Fig. 4.4) data set no such region are observed.

This analysis allows for an intuitive understanding of why the semantic ordering method produced better results for some data sets which make use of an ideal point over others. Data sets that had a relatively smooth boundary along the feasible search space tended to perform better for the canonical method, whereas the data sets that tended to have irregular spacing or discontinuities along the feasible search space tended to perform better for the semantic method. In short, based on the datasets we have tested, areas of the objective space which demonstrate discontinuities, tend to have individuals selected for the pivot and in turn these regions appear to be better explored in the semantic approach.

To demonstrate how the semantic ordering method maintains a lower level of duplication compared to canonical MOEA/D, the external population has been output for Yeast_2 data sets at generations 1, 10, 20, 30, 40 and 50 as seen in Fig. 4.5. It is important to note that typically these duplicates are removed from canonical MOEA/D as part of Step 3.1 as described in Chapter 2.4.6. The chosen data set is indicative of the general duplication pattern exhibited by all data sets, the rest of which have been included in Appendix A, Fig A.9- A.16. The left-hand column of plots shows results yielded by the canonical method for WGT, TCH and PBI respectively and the right-hand column of plots shows results yielded by the semantic-based method for WGT, TCH and PBI, respectively. The size of each marker represents the number of candidate solutions found at that point in the objective space and as such is an indication of the frequency of duplication. For the semantic ordered approach, duplication does not occur as readily but for the canonical approach, there is significant duplication. This detrimental effect is, however, nicely handled by our proposed semantic-based method that encourages diversity.

4.4 Summary

4.4.1 Summary of Results

This work showcases the benefits of incorporating semantics in MOGP, namely, its ability to help promote diversity of solutions. Firstly, we looked at semantics in the context of Pareto-based approaches NSGA-II and SPEA2. Here, we found that two semantic-based distance approaches (SDO and PSDO) proved to be significantly better than canonical NSGA-II and SPEA2 methods and that SSC, a method continuously reported to be

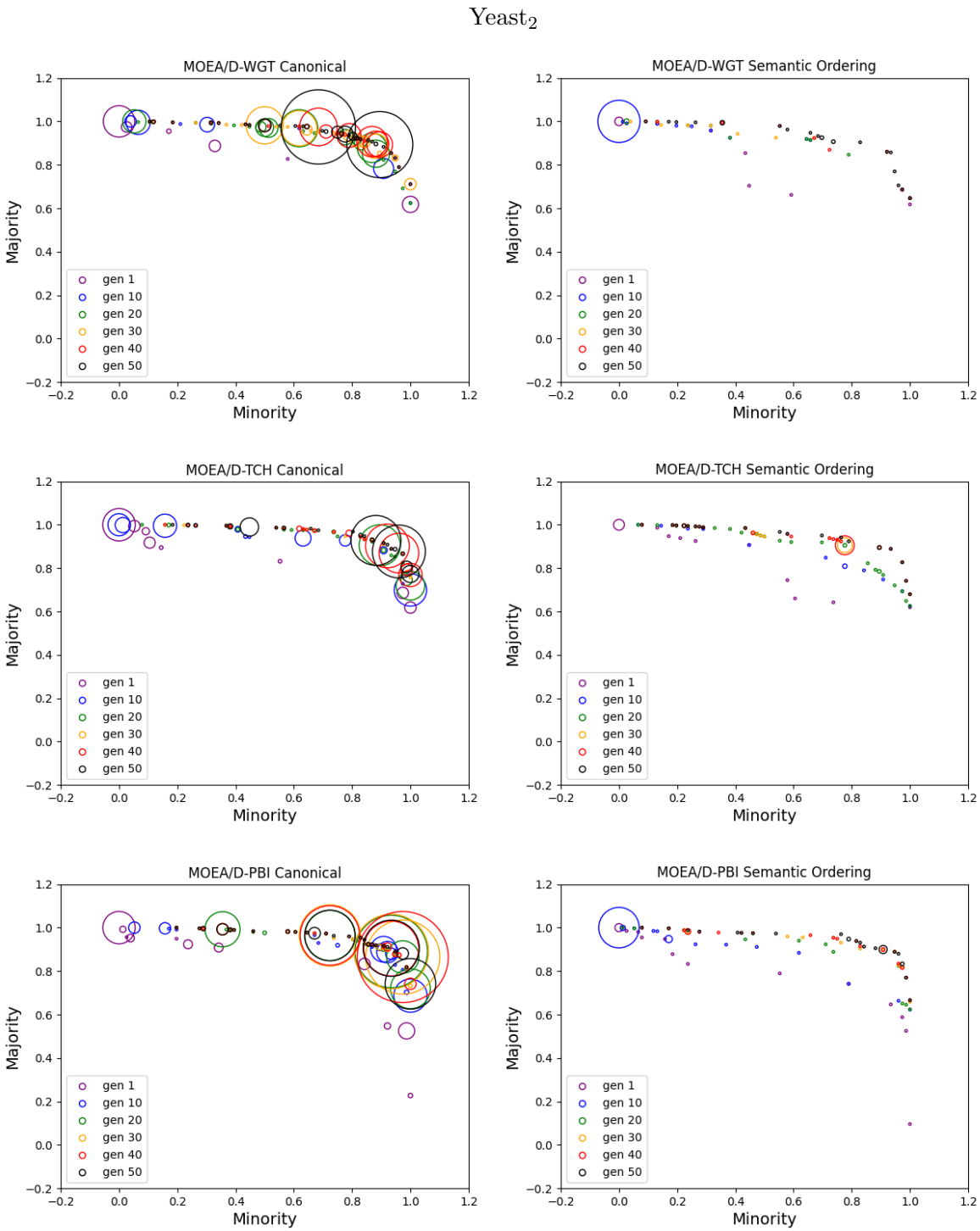


Figure 4.5: Duplicate frequency of individuals at first Pareto Front for Yeast$_2$ dataset for WGT, TCH and PBI approaches for generations 1, 10, 20, 30, 40 and 50, for a single run. Frequency is represented by the size of circles e.g., a large circle denotes a large number of duplicates.

beneficial in single-objective GP, was not able to outperform the canonical approaches for the challenging imbalance datasets tested that we tested.

This led to an important insight into semantic distance-based approaches, in that our analysis shows there is a tendency for these approaches to preference programs that were semantically very similar and also semantically very dissimilar relative to the pivot. The pivot represents an individual from the most sparse region of the search space and can be considered the most divergent relative to the other programs in the population, as such it makes sense that these programs would be preferenced. Additionally, the more semantically diverse programs will produce significantly different outputs, which in turn, ought to also be preferenced.

Next, we demonstrated that the SDO approach which was found to be beneficial in Pareto-based MO optimisation did not translate over to decomposition-based MO optimisation. More specifically, we highlighted key differences between the Pareto-based approaches (NSGA-II and SPEA2) and a decomposition-based approach (MOEA/D), that makes Pareto-based approaches fundamentally incompatible with the semantic-distance metrics, as they were first proposed.

From this, we developed a new approach, Semantic Neighbourhood Ordering, adapting the notion of semantic distance metrics to be used specifically within a decomposition-based context. We have demonstrated for the first time how semantics can be naturally incorporated into MOEA/D, specifically using the scalar optimisation techniques of Weighted Sum, Tchebycheff and Penalty-based Boundary Intersect methods. The approach reduces the duplication of solutions in objective space, while simultaneously allowing a systematic approach for offspring to compete with neighbourhood individuals based on semantics. For the datasets tested, it was found that for data sets with discontinuous or irregular boundaries along the feasible search space the semantic-based method produced significantly better results in terms of the hypervolume averages for the Pareto-approximated fronts. Again, for the datasets tested, for the boundaries of smooth search spaces, the methods performed as well or better for the TCH approach with only the WGT and PBI methods producing significantly mixed results. Further studies are necessary to determine additional ways in which to incorporate semantics into decomposition approaches, but this work offers the first natural approach to implementing semantics in MOEA/D.

4.4.2 Discussion on Semantics and its Role in Neuroevolution

The focus of this chapter was to demonstrate the effectiveness of using semantics in GP, serving as a test case for the robustness of semantic-based distance methods on a whole, since we will adapt semantic-based distance metrics for their use in surrogate-assisted

neuroevolution in Chapters 5 and 6. To be more specific, we used semantic-based distance metrics to promote diversity, in a MO context, where previous works have focused almost exclusively in a single objective domain. We have demonstrated how beneficial it can be to use semantics during the evolutionary process for both Pareto-based and decomposition-based MO optimisation. It is important to note that the underlying mechanisms of how these optimisation techniques work are inherently different but regardless, semantic distances could be adapted for use in both scenarios. In general, the use of semantics in different single and multi-objective frameworks demonstrates the adaptability of incorporating semantics into different EA workflows.

Beyond this, there are a number of other beneficial properties of semantics we will discuss in more detail. For instance, even though we looked exclusively at tree-based GP, it is possible to use semantic information in other GP paradigms such as LGP [158] and grammar-guided GP [159]. Furthermore, the benefits of semantics are not just limited to only promoting diversity but can be extended to other aspects of the evolutionary process such as guiding search [130]. Another very important property of semantics, as was the case in this chapter, is that the size of the semantic vectors can be made equal to the number of fitness cases. In the case of binary classification, this will mean the vector is the size of the partition of the dataset. This is important for two reasons: (i) the vector size is invariant to the structure of the GP programs, where relatively short program trees will still produce an output vector the same size as longer program trees. (ii) having a consistent semantic vector size allows us to use these vectors with standard Euclidean distance metrics. The ramifications for the above aspects is that not only can we consider using semantics for surrogate-assisted neuroevolution that use distance-based imputation but also if we use a GP-based representation for our architectures, then we can impute the fitness of variable-length architectures.

To summarise, with neuroevolution, one of the major drawbacks is the computational expense associated with training a single individual network. To address this issue, in the following chapters, we will show how semantic vectors can be used to build distance metrics within surrogate-assisted evolutionary algorithms. More specifically, we will use these vectors to help impute the fitness for partially trained networks based on phenotypic distance vectors, which are a form of semantic distance vector. Ultimately, the insights gained from using semantics in GP play a significant role in the motivation for thinking about neuroevolution within a semantic-based context.

5

NeuroLGP-SM: A surrogate-assisted approach to Neuroevolution using Linear Genetic Programming

---

The following papers summarises the work outlined in this chapter.

- Fergal Stapleton and Edgar Galván. Initial steps towards tackling high-dimensional surrogate modeling for neuroevolution using kriging partial least squares. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, GECCO '23 Companion, page 83–84, New York, NY, USA, 2023. Association for Computing Machinery
- Fergal Stapleton, Brendan Cody-Kenny, and Edgar Galván. Neurolgp-sm: A surrogate-assisted neuroevolution approach using linear genetic programming. In *International Conference on Optimization and Learning (OLA)*, 2024
- Fergal Stapleton and Edgar Galván. Neurolgp-sm: Scalable surrogate-assisted neuroevolution for deep neural networks. In *2024 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, 2024

Evolutionary Algorithms (EAs) can play a crucial role in the architectural configuration and training of Artificial Deep Neural Networks (DNNs), a process which is known as neuroevolution. However, neuroevolution is hindered by its inherent computational expense. The most computationally intensive aspect lies in evaluating the fitness function of a single EA candidate solution. One way to address this challenge is to employ Surrogate-assisted EAs (SAEAs). While a few SAEAs approaches have been proposed in neuroevolution, none have used semantics distances at their core for truly large DNNs. Drawing inspiration from Genetic Programming semantics, as discussed in Chapter 4, we use phenotypic distance vectors, which are vectors consisting of the final layer outputs of a DNN. Previously, we demonstrated how semantics could be used to improve the diversity of solutions during evolution, however, now we will focus on using semantic information to impute the fitness of partially trained DNNs within a SAEA.

To this end, these vectors are used to train a surrogate modelling technique known as Kriging Partial Least Squares (KPLS), an approach that is effective in handling these large vectors in the context of estimating θ hyperparameters, making them suitable for search. Our proposed approach, named Neuro-Linear Genetic Programming surrogate model (NeuroLGP-SM), efficiently and with a high degree of accuracy estimates DNN fitness without the need for complete evaluations. NeuroLGP-SM demonstrates competitive or superior results compared to 12 other methods, including NeuroLGP without SM, convolutional neural networks, support vector machines, and autoencoders. For comparison purposes, we also code and use a baseline approach incorporating a repair mechanism, a common practice in neuroevolution. Notably, the baseline approach surpasses the renowned VGG-16 model in accuracy. Given the computational intensity inherent in DNN operations, a singular run is typically the norm. To evaluate the efficacy of our proposed approach, we conducted 96 independent runs spanning a duration of 4 weeks. Significantly, our methodologies consistently outperform the baseline, with the SM model demonstrating superior accuracy or comparable results to the NeuroLGP approach. Additionally, it is worth noting that NeuroLGP-SM exhibits a 25% reduction in computational requirements and is also 25% more energy-efficient than its NeuroLGP counterpart. This efficiency advantage adds to the overall appeal of our proposed NeuroLGP-SM in optimising the configuration of large DNNs.

5.1 Introduction

Evolutionary Algorithms (EAs) [20] have proven to be effective in both the crafting of architectures and hyperparameter optimisation of Deep Neural Networks (DDNs) [57]. This application is commonly known as neuroevolution, a widely explored field highlighted by the abundance of scientific publications and impactful outcomes [37], and have

been applied to numerous problem domains, such as autonomous vehicles [8] and face recognition [160]. The pursuit of optimal DNN architectures has led to the use of various methodologies, including EAs [37], reinforcement learning [19], and more, however, a significant challenge persists across these methods: the substantial computational resources required to identify high-performing networks.

The rise of GPU-accelerated hardware has helped alleviate some of this computational cost, however, a significant proportion of research in DNNs is based on incremental improvements on DNN algorithms for benchmark problems [161], where there is a significant correlation between network complexity for incremental gains in terms of additional performance. In fact, when looking at very large models of hundreds of billions of parameters, it can cost millions of dollars for a single iteration [21]. This energy consumption is further compounded when considering population-based neuroevolutionary techniques which require many networks to be trained and evaluated in order to find suitable architectures.

One way to address this significant issue is with the use of surrogate-assisted evolutionary algorithms (SAEAs). SAEAs can be used to estimate the fitness of DNNs without the need to fully train each network. In particular, surrogate modelling strategies that employ Bayesian optimisation have shown much promise [21]. However, a major challenge remains in how best to deal with the surrogate representation. For instance, using genotype information to build surrogates often requires complex encoding strategies [25], and in some instances, constructing adequate distance metrics to compare different network topologies is not feasible [27]. Using phenotypic information on the other hand has shown some promise [23, 27], but a challenge remains in scaling to deeper and more complex networks which inherently requires a high-dimensional representation [5].

In this work, we analyse a novel population-based Neuroevolutionary technique, referred to as Neuro-Linear Genetic Programming (NeuroLGP), and its surrogate model variant NeuroLGP-SM [6]. Using a robust model management strategy, we use phenotypic distance vectors to estimate the performance of partially trained DNNs. These vectors are comparatively large for the optimisation problem at hand [5] and, as such, we incorporate an approach that is designed for handling high-dimensional data, known as Kriging Partial Least Squares (KPLS). This approach allows for a novel and scalable surrogate-assisted technique that is skilfully adept at handling neuroevolution of DNNs and to the the best of our knowledge, this method of surrogate-assisted neuroevolution of DNNs has not been studied before.

The aim of this study is to apply Surrogate-assisted Evolutionary Algorithms (SAEA) in neuroevolution, using Kriging Partial Least Squares (KPLS) on phenotypic distance vectors inspired by Genetic Programming Semantics [28]. The core aspects of this study have been summarised below:

(1) **Baseline model:** We implement and validate a baseline model employing a repair mechanism, a strategy frequently utilised in neuroevolutionary methods. This model surpasses the well-established VGG-16 model, setting a high-performance benchmark for comparison with our proposed approaches.

(2) **Representation:** We introduce an innovative representation based on Linear Genetic Programming (LGP) [10], termed NeuroLGP, facilitating the automatic discovery of well-performing DNNs that outperform the baseline model. The NeuroLGP approach is employed to compute the fitness of the entire population and serves as an excellent method to compare against a surrogate model (SM).

(3) **Model management strategy:** This approach remains invariant to varying network topologies and robust to data augmentation techniques. Consequently, we can train our networks with a significantly reduced number of instances while maintaining the ability to generalise effectively to unseen data.

(4) **High-dimensionality:** To address the challenge of high-dimensional data and enable the use of a SM to reduce computational demands in neuroevolution, we employ Kriging Partial Least Squares [31]. The fusion of this technique with NeuroLGP results in our second proposed approach referred to as NeuroLGP-SM. This approach consistently identifies DNNs that perform similarly to those discovered by NeuroLGP, simultaneously reducing fitness evaluations and training time required for these DNNs. In our previous work [5], we identified that it was not possible to use the original Kriging approach to this end.

(5) **Performance Metrics:** We employ three well-defined metrics to assess the predictive capabilities of NeuroLGP-SM in terms of model fitness. These results align with the competitive or superior performance of our NeuroLGP-SM approach compared to 12 popular techniques, including convolutional neural networks, autoencoders, and support vector machines, across four challenging classification tasks.

(6) **Efficient Fitness Estimation:** We demonstrate the accurate estimation of DNN fitness values without full evaluations through our proposed approach, NeuroLGP-SM, employing KPLS on phenotypic distance vectors.

(7) **Energy and Time Consumption Analysis:** We provide a reliable formula to gauge the energy consumption of our approaches, revealing that NeuroLGP-SM is $\sim$25% more efficient compared to its counterpart that does not use surrogate models. Additionally, we show that similarly run time is reduced by $\sim$25%.

(8) **Encoding for Analysis:** Through clever encoding, we allow easy access to analyse

the internal structures of the architectures, enabling us to conduct an in-depth analysis of the networks discovered by our proposed approaches.

In the following chapter, we provide a motivation for why we consider LGP as an approach to integrate into neuroevolution. For instance, LGP allows for variable topology, where a number of neuroevolutionary approaches are fixed. Furthermore, we provide a short preliminary analysis that highlights the limitations of traditional Kriging for surrogate-assisted neuroevolution, namely, that the traditional Kriging approach is computationally expensive when considering high-dimensional phenotypic distance vectors.

5.2 Motivation

5.2.1 Properties of Linear Genetic Programming

In evolutionary algorithms, the genotype of an individual plays a significant role in encoding all the relevant properties to evolve. An ill-posed representation may lack sufficient information to effectively solve a problem. Furthermore, an overly complex representation may in itself be wasteful or fail to solve the problem entirely. Some common representations come in the form of binary strings like in the case of Genetic Algorithms (GA) or tree-like representations in Genetic Programming (GP) [36]. The suitability of the representation is dependent on the problem domain, for instance, the problem domain of symbolic regression is particularly well encapsulated by the tree-like representation of GP. To date, the use of GP for neuroevolution has been underexplored. A 2021 survey [106], highlighted that less than 7% of encodings use GP for representation in neuroevolution, while the majority of encodings, with over 50%, belong to GAs.

Prior to this work, to the best of our knowledge, Linear Genetic Programming (LGP) is an approach that has not been used for neuroevolution. Representation in LGP is based upon a sequence of instructions for an imperative programming language or machine code, where the linear program structure is evolved as opposed to the tree-like structure of canonical GP [10]. In its original form, there are two motivating factors for why researchers would want to use LGP. Firstly, most computer architectures execute programs as a linear sequence of instructions being executed at regular time-steps and secondly, computers are not capable of naturally running tree-based representations unless aided by interpreters or compilers [162]. The former point is interesting to note since the construction of DNN architectures often involves the stacking of layers in a linear and sequential fashion. In particular, the output of a given layer is intended as the input of the next. Therefore, the stacking of layers can be considered as a step-by-step construction that can follow an imperative paradigm.

When considering encoding for neuroevolution, there are a number of important considerations. To start, we can broadly categorise the type of encoding into two main categories [163]; direct and indirect encoding. A direct encoding approach encodes all properties of the network, such as the nodes and connections into the genetic representation, doing so explicitly. Definitions may differ slightly, but indirect encoding instead either reuses genetic information multiple times within the genome or the genome contains generational rules [164] [104]. It has been noted [165], in nature the genome likely encodes general rules for connectivity, rather than the genome encoding the wiring explicitly, as a result of a genomic bottleneck, or limitation in the size of what can be encoded into the animal genome [165]. To some extent, indirect approaches mimic this concept as nodes and connections are not explicitly encoded, as such, there may be a strong biological justification for considering indirect methods.

Also of interest, is whether the encoding allows for fixed or variable-length architectures. Variable-length encoding is an important concept in neural architecture search as it allows the evolutionary process to search for shallower or deeper networks. The LGP approach allows for variable-length encoding through the crossover operation. With a fixed topology the network length cannot grow beyond a certain depth. While this approach can be beneficial in ensuring architectures remain compact, it can also be sub-optimal if deeper networks are desired.

Another important consideration is whether the approach is capable of encoding skip-connections and branching connections, which are often beneficial for ANN performance. In GAs, this typically requires embedding rules within the genotype to handle but with graph-based GP methods like Cartesian Genetic Programming (CGP) [166] and LGP this can be handled explicitly by the representation. However, a comparison between graph-based CGP and graph-based LGP highlighted a major difference between the two approaches [167]. In CGP the connectivity to previous nodes in the layer is controlled by a level back parameter which defined *a priori* and as such is not controlled by the evolutionary process. While this may be suitable for scenarios where a general architecture is understood or preferred, when exploring variable-length or multi-branch architectures this can be limiting. LGP, on the other hand, does not have this parameter and connectivity is controlled by the evolutionary process itself. In Chapter 6, we detail a multi-branch and skip-connection representation in more detail.

Table 5.1 summarises some key encodings and representations for Neuroevolution. Two survey papers [37, 163], offer a more comprehensive analysis of neuroevolutionary encodings and representations. To summarise, the LGP approach has the following aspects which are desirable for neuroevolution:

- The proposed approach uses indirect coding which has a strong biological inspira-

Table 5.1: Summary of encodings, representations and skip connections of some of the most notable Neuroevolutionary approaches.

Approach	Encoding	Representation	Representation Type	Skip/Branching
CGP-CNN [107]	Direct	GP	Fixed-length	Yes
EvoCNN [104]	Indirect	GA	Variable-length	No
GeneticCNN [102]	Direct	GA	Fixed-length	Yes
DeepNEAT [90]	Direct	GA	Variable-length	Yes
CoDeepNEAT [90]	Indirect[2]	GA	Variable-length	Yes
HyperNEAT [89]	Indirect	GA	Variable-length	Yes
NeuroLGP	Indirect	GP	Variable-length	Yes[3]

[1] This has been defined as indirect due to using mean and std to update weights.
[2] Although the general encoding is direct, the co-evolved population of templates adds modularity, as such we have defined this as indirect.
[3] This has not been implemented in this chapter but is inherently possible based on the representation.

tion and allows us to encapsulate a complex representation of the network without explicitly encoding every parameter.

- LGP is implicitly variable-length in design. This can allow the neuroevolutionary process to find deeper architectures if required.
- LGP has examples using both a typical linear crossover (which can lead to increased program length) and homologous crossover (which does not increase program length). Carefully balancing the type of crossover applied can help alleviate bloat [162].
- Some representations (for example CGP-CNN [107]) drop crossover altogether and only consider mutation, therefore the role of crossover in GP-based neuroevolution is yet to be explored fully.
- The data flow of LGP allows for a directed graph-like structure. This should allow LGP to be exploited to not only encode skip connections but also multi-branch connections (discussed in Chapter 6).

5.2.2 Limitation of Traditional Kriging Approach

A major impediment in the use of Surrogate-Assisted Evolutionary Algorithms (SAEAs) in neuroevolution is the computational power still required when using well-known SAEA approaches, such as the use of the Kriging method. In Chapter 2.3.3, we discuss this limitation in detail but in short, the computational cost of estimating the θ hyperparameters is $O(m^3)$ where m is the surrogate model sample size using the traditional Kriging approach. A novel workaround to this limitation is to subset the surrogate model samples

Table 5.2: Details of the classification data sets used in our research as well as preliminary results for an initial surrogate model of 100 networks. A '-' indicates the experiment did not complete within 48 hrs.

Dataset	Brief description	Pheno. dist. vector	KPLS (HH:MM:SS)	Kriging (HH:MM:SS)
Iris	3 classes (Type of Iris plant)	336	00:00:25	01:52:38
Yeast	2 classes (Protein sequence)	1338	00:02:24	-
Ecoli	8 classes (Localization site)	2016	00:01:09	-
Abalone	2 classes (Number of rings)	6264	00:10:38	-

from a larger archive, as proposed in [27]. The authors' results were promising for subset sizes of 25 to 200. However, it is likely for more complex networks such as DNNs, with larger search spaces, the surrogate may require larger sample sizes in order to be well informed.

As a preliminary analysis, we focused on traditional ANN architectures of just a few layers to highlight the limitations of the Kriging approach. An important part of the surrogate model management strategy is the model initialisation as highlighted in Fig 2.6. On initialisation, each individual representing an ANN is trained for a limited number of epochs[4]. This results in using the behaviours of the networks. That is, we generate a phenotypic distance vector, similar to the work by Stork et al. [27], where the vector contains the output of the nodes at the final layer for all data instances. We use the Kriging and Kriging Partial Least Squares (KPLS) method, by employing the Surrogate Modelling Toolbox (SMT) [170], to estimate the fitness values of the subset of the population by feeding these vectors.

To test the efficiency of our methods, we use data sets from the UCI machine learning repository [171]. All experiments were run on Intel Xeon Gold 6148. Table 5.2 summarises the preliminary results, showing the time taken to construct an initial population of 100 fully trained networks that are then used to construct the surrogate model. The third column, in Table 5.2, denotes the phenotype vector length. Larger phenotype vectors result in higher dimensional data for the surrogate model to train on. A dash symbol (-) indicates the experiment did not complete within 48 hours. We can see that based on the last two columns it is infeasible to use the Kriging approach for these datasets while for the KPLS approach, the times would be more than sufficient. This preliminary analysis shows the use of common SAEA techniques such as Kriging cannot be used in neuroevolution due to the high dimensional data normally required for neuroevolution. We have taken the initial steps to show that this can be addressed by using the KPLS method.

[4] A CGPANN variant, called dCGPANN [168,169] was used to initialise the individuals, creating random architectures and allows stochastic gradient descent to train our networks, but beyond this initial step, individuals were not further evolved. For the rest of this work, the NeuroLGP approach is employed.

```
1  def neuroLGP(...)
2  {
3      r[0] := Conv(r[1])
4      // r[4] := BatchNorm(r[3])
5      r[5] := MaxPool(r[0])
6      r[0] := BatchNorm(r[5])
7
8      ...
9  }
```

```
input = tf.keras.Input(shape=(imput_dim))

x = tf.keras.layers.Conv2D(32, 3, ...)(input)
# x = tf.keras.layers.BatchNormalization()(x)
x = tf.keras.layers.MaxPooling2D(3)(x)
x = tf.keras.layers.BatchNormalization()(x)

output = tf.keras.layers.Dense(x)
model = tf.keras.Model(output)
```

Figure 5.1: *left:* NeuroLGP psuedocode for python. *right:* Functional API example in TensorFlow [11].

5.3 Methodology

5.3.1 NeuroLGP

The original LGP encoding is based on the concept of using registers, which are units of computer memory storage for manipulating data while executing instructions, in a low-level programming context. The content of these registers are altered using instruction operations. There are three main components to instruction; an *operand* which performs a specific function on one or more *registers* which store the result in a *destination register*. In the case of 2-register instruction encoding the operand operates on a single instruction and for a 3-register instruction encoding operates on two instructions. The left of Fig. 5.1 highlights an example of LGP written in C code, where `r[i]` denotes the i^{th} register. This example contains both effective and non-effective lines of code, where the non-effective lines of code are commented out and subsequently not compiled. Each line of code is executed imperatively. The register `r[0]` is a specially designated register for the final output of the program. With the NeuroLGP approach, instead of using registers to store small amounts of data, we instead use the idea of registers as pointers to much larger amounts of data. As such, the registers instead control the flow of the initial and intermediary data from each outputted layer of our evolvable DNN. The right of Fig. 5.1, demonstrates how the expected representation would look in TensorFlow, where the general form of this code is similar to the figure on the left.

To understand how this representation can be useful for the case of neuroevolution, some example code in TensorFlow is given to demonstrate the imperative nature of defining models as seen in the right of Fig. 5.1. This code demonstrates an example of a model definition using the functional API from TensorFlow. On Lines 3, 5 and 6, to the left of the assignment, a variable x will hold the outputs of each statement and for Lines 5 – 6, to the right of the assignment, x is passed to each layer. As such, the variable x holds transient data which updates as each line is executed. The aim is to replace x using abstract or virtual registers (i.e r[0], r[1], r[2] ...etc), where the x variables to the

left of the assignment are represented using a *destination register* and values to the right represent *registers* to be operated on. Line 4 is not executed. It is important to note the dense layer, as demonstrated on the right, is not part of the genotype and as such is not evolved. As such, only the feature extraction portion of the architecture is evolved.

Traditionally, in LGP, registers can be of different types (input registers, calculation registers and constant registers). In this work, each register is referencing the intermediary inputs of each neural network layer, i.e. the information that is fed to each network operation and so, in this regard, each register is treated as input registers (though it may be useful to have special registers specifically for handling skip connections and multiple branches). While this may seem wasteful in terms of memory, multiple registers are only used in the genotypic representation and when calculating the fitness, the number of effective registers can be reduced as part of a repair process. Likewise, the non-effective code can also be removed at the same time. As such, the phenotypic representation of the network will not only be much simpler than the genotypic representation, but also much smaller. Fig.5.2 demonstrates the genotype-to-phenotype mapping.

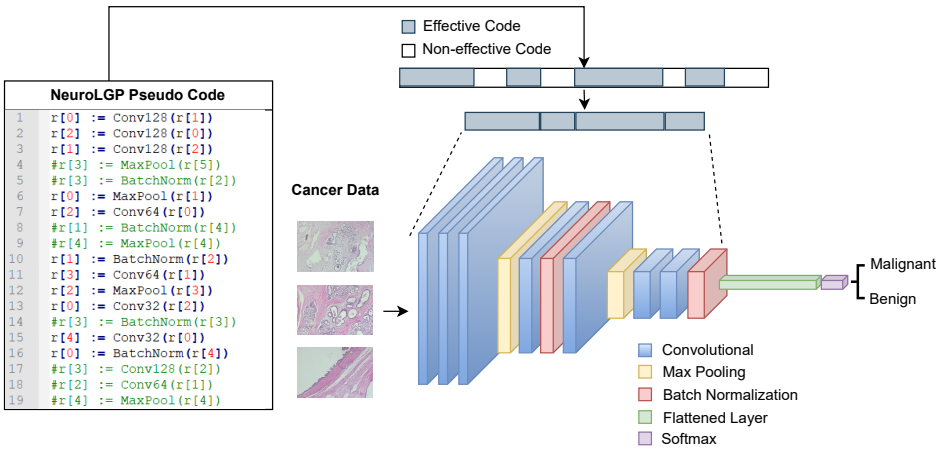


Figure 5.2: Diagram of the NeuroLGP genotype-to-phenotype mapping. The pseudocode for the set of instructions (left-hand side) can be represented as the genotype with effective and non-effective code (top) and produces the resulting phenotype (bottom right-hand side) as a specific neural network architecture. Note that the non-effective coding is not present in the phenotype.

5.3.2 NeuroLGP with Surrogate Model (NeuroLGP-SM)

To identify individuals requiring full evaluation, we use the Expected Improvement (EI) criteria [172]. EI guides the selection of candidate solutions for evaluation by estimating the improvement over the current best solution. This enables us to prioritise solutions

from areas in the search space expected to exhibit the most significant improvement. The calculation of EI is shown in Eq. 5.1,

$$\begin{aligned} \text{EI} &= \begin{cases} (\hat{f}(x) - f(x^*))\Phi(Z) + \sigma(x)\phi(Z) & \text{if } \sigma(x) > 0 \\ 0 & \text{if } \sigma(x) = 0 \end{cases} \\ Z &= \frac{\hat{f}(x) - f(x^*)}{\sigma(x)} \end{aligned} \tag{5.1}$$

where $\hat{f}(x)$ is the model's predicted performance of the surrogate for the phenotypic distance vector x, where $f(x^*)$ is the best-known value of the objective function so far (in this case maximum) and Φ and ϕ are the cumulative distribution function (CDF) and probability density function (PDF) of the standard normal distribution, respectively.

Fig. 5.3 summarises the surrogate model management strategy. The left-hand side of the figure shows the interplay between the evolutionary algorithm and the surrogate model as previously shown in Fig. 2.6, however, we have highlighted the steps controlled by the model management strategy (dark blue dashed-line). The right side of the plot looks at the model management strategy at a more granular level. To simplify the diagram the initial modelling section has been dropped, but is important to note that we inform our surrogate model with randomly generated individuals prior to running through the EA loop. The diagram has been annotated with five key steps (dark blue text). The annotations are: (i) *Split:* first, the population is split with a 40/60 split for individuals to be fully evaluated *vs.* the partially trained individuals, respectively, (ii) *Estimate fitness:* the fitness is estimated using the KPLS approach as informed by the previous generation, (iii) *Add data:* the phenotypic distance vector for the fully evaluated portion of the population is added to the surrogate training data (note: the phenotypic distance vector is taken before the full fitness evaluation), (iv) *Extract data and train:* the phenotypic distance vectors are collectively used to train the KPLS approach as detailed in Chapter 2.3.3, and (v) *Calculate EI:* the EI criteria is calculated for the incoming population of individuals.

A 40/60 split was chosen for the expensive portion of the surrogate model (expensive proportion = 0.4 in Table 5.5), i.e., 40% of individuals are used to re-inform the surrogate model each generation (including the previous surrogate training data). The ratio of full to partial evaluations would be roughly 50% (the initial generation is fully evaluated), which allows for hypothetical best potential time saving of ∼50%, however, given that the neural networks require compiling and initial training of 10 epochs, we expect this to be lower. Also, there is a notable tradeoff that needs to be considered when selecting an appropriate split, firstly, that we need enough full evaluations to ensure the surrogate model is well-informed but also that enough time is saved to warrant justification for using the surrogate model.

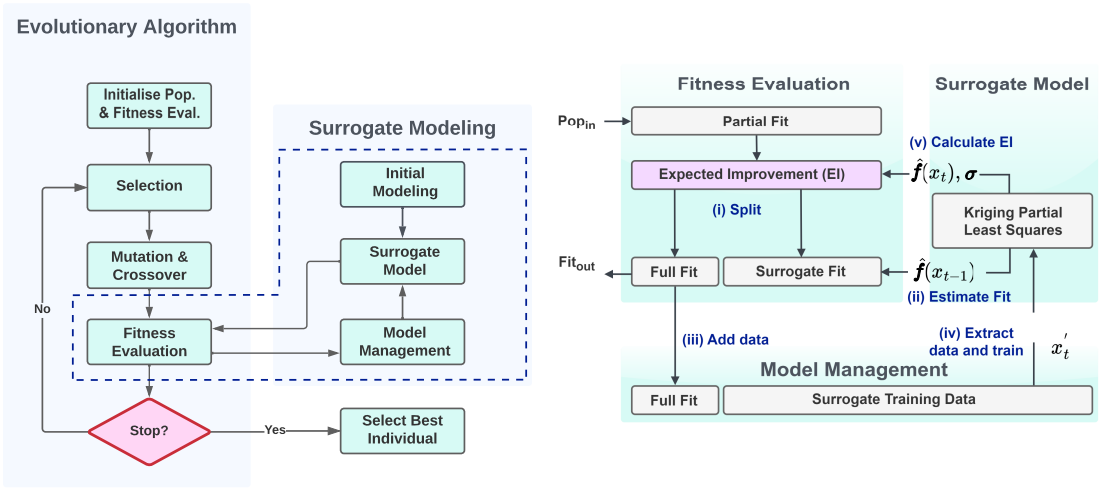


Figure 5.3: *Left:* Diagram showing the interplay between a typical evolutionary algorithm and a surrogate model approach. *Right:* The surrogate model management strategy is shown on a more granular level.

5.3.3 Genetic Operations and Repair Mechanism

The mutation operator we use in this work mutates either a single input or output register or the operand. The mutation operator works on both effective code and non-effective code. Additionally, we make use of a novel effective crossover operator. This operator selects two crossover points from the effective sections of the parent code and then transfers segments to create offspring. While the ends of each segment contain effective code, the code within the segment may be non-effective. A further point, mutation repair is applied at either segment end to ensure input and output registers match after crossover has been applied. In other words, the output register of the parent code will be the same as the input register of the crossover segment at the first crossover point and then the output register of the segment will be the same as the next register of the remaining portion of the parent at the second crossover point, ensuring that the newly transferred segment is effective and that there are no downstream changes to the effective code of the remaining parent segment. Ultimately, this results in always producing a valid network.

A repair mechanism is incorporated when performing the genotype-to-phenotype mapping to ensure models are compilable. There are several conditions for incorporating a repair mechanism, but it is important to note that in each case, the repair is only performed on the effective code by either inserting or deleting a specific layer.

- When there is no effective code in the genotype, we perform a single effective mutation inserting a single convolutional layer from our feature list.
- It is possible for the program to compile without a convolutional layer at the start

Flattened Size Condition	Dropout	Dense Layer
$< 25,000$	0.2	512 units (ReLU)
$< 50,000$	0.2	256 units (ReLU)
$< 100,000$	0.2	128 units (ReLU)
$> 1,500,000$	-	-

Table 5.3: Flattened size conditions for fully connected layer. Additional handling is done to remove layers if the flattened layer exceeds 1,500,000.

(i.e., Max pooling, dropout, etc). We add an effective insertion of a convolutional layer before any layer that does not match this criteria.

- When the input data dimensions are too low or if the number of data-reducing operations exceeds the dimensions of the data, the neural network will fail to compile, we remove layers iteratively until the model is valid.

- If the input data dimensions are too high or if the number of data-reducing operations are too few, our flattened layer can be composed of an arbitrarily large number of nodes. Furthermore, adding further fully-connected layers can cause the models to become prohibitively expensive to evaluate, as such, we put additional conditions in to check and add a fully-connected layer before the final softmax layer if certain conditions have been met The size of the fully connected layer is dependent on the number of parameters from the CNN portion of the architecture and has been summarised in Table 5.3. It is important to note that unlike the previous repairs, this is not directly encoded in the genotype as the fully connected layer is not evolved.

5.4 Implementation Details

The Breast Cancer Histopathological Image Classification (BreakHis) [173] is a binary classification dataset consisting of microscopic images containing 2,480 benign and 5,429 malignant tumours, where images were obtained as four different magnifications ($\times$40, $\times$100, $\times$200 and $\times$400) and are split into four individual datasets for this work. Each image consists of 64x64 pixels (down-scaled resolution from 700x460 pixels) and 3-channel RGB with 8-bit depth in each channel. The split for training, validation, test_1 and test_2 is approximately (63.5%, 12.5%, 12.5%, 12.5%), where the first three splits conform to a ratio of (70:15:15), conforming to the general training/test split as seen in other approaches [174]. The first three splits are used purely for training and evaluating networks and the last split is retained for an unbiased analysis of the evolutionary process. In other words, test_1 is withheld from the context of the individual neural networks but is used for fitness evaluation for the evolutionary process and test_2 is the withheld dataset for

Parameter	Value	Description
Rescale	1./255	Scale pixel values
Rotation range	30	Degree range for random rotations
Width shift range	0.1	Fraction shift of total width
Height shift range	0.1	Fraction shift of total height
Shear range	0.2	Change perception
Zoom range	0.2	Image zoom
Horizontal flip	True	Randomly flip images horizontally
Fill mode	nearest	Pixel fill criteria

Table 5.4: List of data augmentation techniques and parameters used. See [12], for more details on each augmentation technique.

the entire evolutionary process. The dataset is imbalanced and the synthetic minority oversampling technique [175] is used to up-sample the minority class. Additionally, a number of data augmentation techniques have been employed as shown in Table 5.4.

The genotype consists of both effective and non-effective code, where the feature extraction portion of the architecture is evolved. Table 5.5 details the mutation and crossover rates, segment lengths for crossover operation, as well as minimum and maximum chromosome lengths (defined as program length in table). Distances/lengths are based on effective length except for the minimum and maximum chromosome lengths which consider the full chromosome length. Only the most salient information has been included in this table, full details of the configuration files, including additional details on the setup and a full list of evolvable layers have been included in Appendix B in Tables B.1.

Redundancy was not explicitly handled during evolution, i.e., where many genotypes produce the same network architecture (phenotype). However, a relatively high mutation rate is used in this work and operates on both effective and non-effective code. Approximately 1/5th of the code is effective by the end of a run (15 genes are effective for a max genotype of 70) and due to the low number of registers used, mutation on non-effective code had a relatively high chance of becoming effective as a result. No significant amount of duplicate networks were found by the end of each run.

Initial epoch and epoch numbers were selected based on network convergence. Fig. 5.4 shows an example of the convergence plots for training and validation accuracy for BreakHis $\times 400$ dataset for 30 epochs across 8 runs. The general trend of training and validation converging was seen across all datasets by 30 epochs. An initial epoch number of 10 was chosen as it was found that in general the networks had not converged by this point, but also that there should be some level of correlation between the networks' performance at this epoch and the final epoch. Something that is not considered in this work is the use of a stopping criteria and for consistency, a set epoch number was considered

Category	Attribute	Value
SURROGATE		
	METHOD	KPLS
	INITIAL EPOCH NUM	10
	INITIAL SURROGATE POPSIZE	30
	EXPENSIVE PROPORTION	0.4
EXPERIMENTAL SETUP		
	GEN SIZE	15
	POP SIZE	50
	ELITE PERCENTAGE	0.2
	SELECTION	TYPE: tournament, SIZE: 5, NUMBER OF TOURNAMENTS: 1
	CROSSOVER	RATE: 0.3 TYPE: linear, DISTANCE OF CROSSOVER POINTS: 5, MINIMUM PROGRAM LENGTH: 2, MAXIMUM PROGRAM LENGTH: 70, MAXIMUM SEGMENT LENGTH: 5, MAXIMUM DIFFERENCE IN SEGMENT LENGTH: 3
	MUTATION RATE	0.9
	MAX REGISTER	6
	EPOCH NUM	30

Table 5.5: Details of the experimental setup for NeuroLGP method.

across all datasets. Future work could consider using stopping criteria or using an adaptive measure for considering initial epoch number. Later, in Chapter 6.4.6 an analysis of varying epoch number is considered with an updated version of the NeuroLGP approach.

We employ three approaches to investigate the validity and quality of our proposed surrogate-assisted neuroevolution model. These are:

(1) A *baseline* approach which is similar to a random search approach, in that, it structures the network architecture layers in a random order. These networks are not evolved but are repaired in line with the repair mechanism as discussed in Section 5.3.3. All networks are fully trained to the full number of epochs (30 epochs).

(2) The *expensive* approach, which uses a novel encoding to evolve the structure of our networks, training to the full number of epochs as outlined in Chapter 5.3.1 (Pop. size = 50, Gen. size =15).

(3) The *surrogate* approach where we integrate the surrogate modelling strategy, as outlined in Chapter 5.3.2, into the *expensive* approach. The surrogate model is informed using partially trained networks (10 epochs) for 60% of the population size.

Experiments were conducted using Kay supercomputer provided by the Irish Centre for High-End Computing (ICHEC). Experiments were run in parallel, with each run

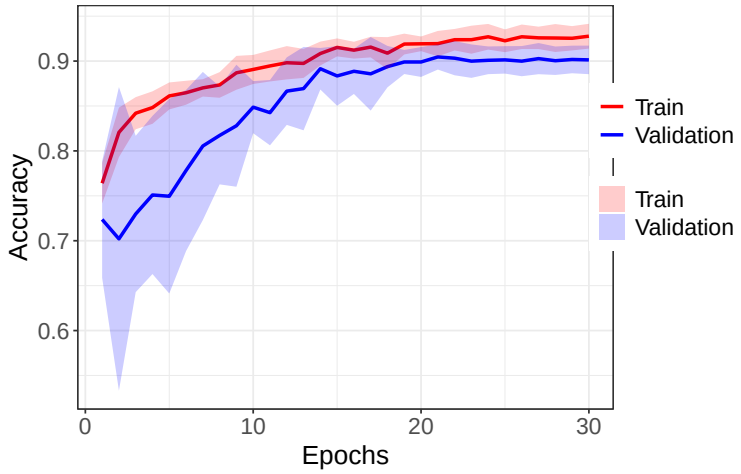


Figure 5.4: Convergence plot for BreakHis ×400 for 8 runs and 30 epoches. Solid lines represent mean training (red) and validation (blue) accuracies.

assigned to a single Nvidia Tesla V100 GPU with 16GB RAM. Additionally, each run has access to a 20-core 2.4 GHz Intel Xeon Gold 6148 (Skylake) CPU processor which is used during training of the surrogate model portion of the surrogate approach. Overall, the expensive approach took ∼28 GPU days and the surrogate approach ∼21 GPU days, for 8 runs across the 4 datasets.

5.5 Results

5.5.1 Preliminary Analysis of the Baseline Model

The baseline in our approach makes use of a random search over the feature extraction portion of our network architecture with a set of repair operations to ensure the architecture can be compiled. Typical network sizes from this approach ranged from ∼20K to ∼12.5M parameters with the baseline. For example of VGG-16 has ∼40M parameters [176]. Fig. 5.5 shows the distribution of accuracy values for the baseline, for magnification ×200, for all networks across 8 runs (750 individuals per run, 6000 individuals overall). It is important to point out that across all 8 runs, every model found by the baseline was valid and compilable. Slight peaks at 0.33 and 0.66 represent poor-performing networks that classify all the data as a single class. We can see that despite a tailed distribution to the left, the vast majority of individuals are centred around the median peak of 0.83. Additionally, the performance of VGG-16 is also shown in the figure with an accuracy of 0.87, which was verified in our own experiments and documented

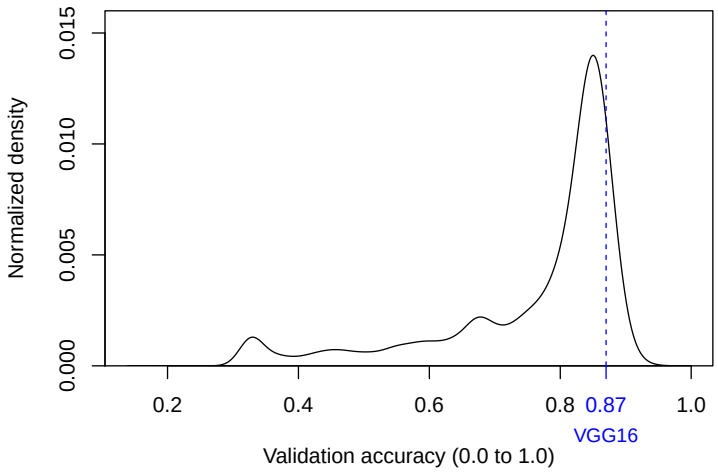


Figure 5.5: Accuracy of 6000 individuals (read text).

in other work [177]. The main takeaway is the baseline approach can find competitive architectures to compare against the surrogate and expensive approaches.

5.5.2 Comparison of Baseline, Surrogate and Expensive

A comparison is conducted between the three experiments as outlined in Section 5.4 for the BreakHis dataset. The violin plots in Fig. 5.6(a-d) plot the accuracies of the best individual in terms of fitness, for each run, for the $\times$40, $\times$100, $\times$200 and $\times$400 magnifications, respectively. For reference, if we were to look at baseline as an example (left-hand side in each plot of Fig. 5.6), then the individuals shown in these plots represent individuals found at the far right of the density plot in Fig. 5.5. Initially, we ran the three models for 4 runs only, which took approximately 40 GPU days in total. It should be noted that in neuroevolution the norm is usually to do a single run. The initial results showed a tendency for the surrogate and expensive models to outperform the baseline model for the $\times$40, $\times$100 and $\times$400 models, however for $\times$200 the surrogate and expensive underperformed. It was decided to extend this, to 8 runs totalling 80 GPU days in all. The updated results are represented here in the violin plots, where again we can see that in the case of $\times$40 and $\times$100 magnification, the surrogate (green) and expensive (blue) models tend to outperform the baseline (red) model. Results are more mixed when we look at $\times$200 but are markedly improved over the initial set of 4 runs. The $\times$400 magnification appears somewhat mixed, however, with more runs, $\times$400 may have more separation for surrogate and expensive models over the baseline given there are few better-performing individuals above the 92% mark. Comparing surrogate and expensive models in particular, an important observation can be made, in that for all magnifications

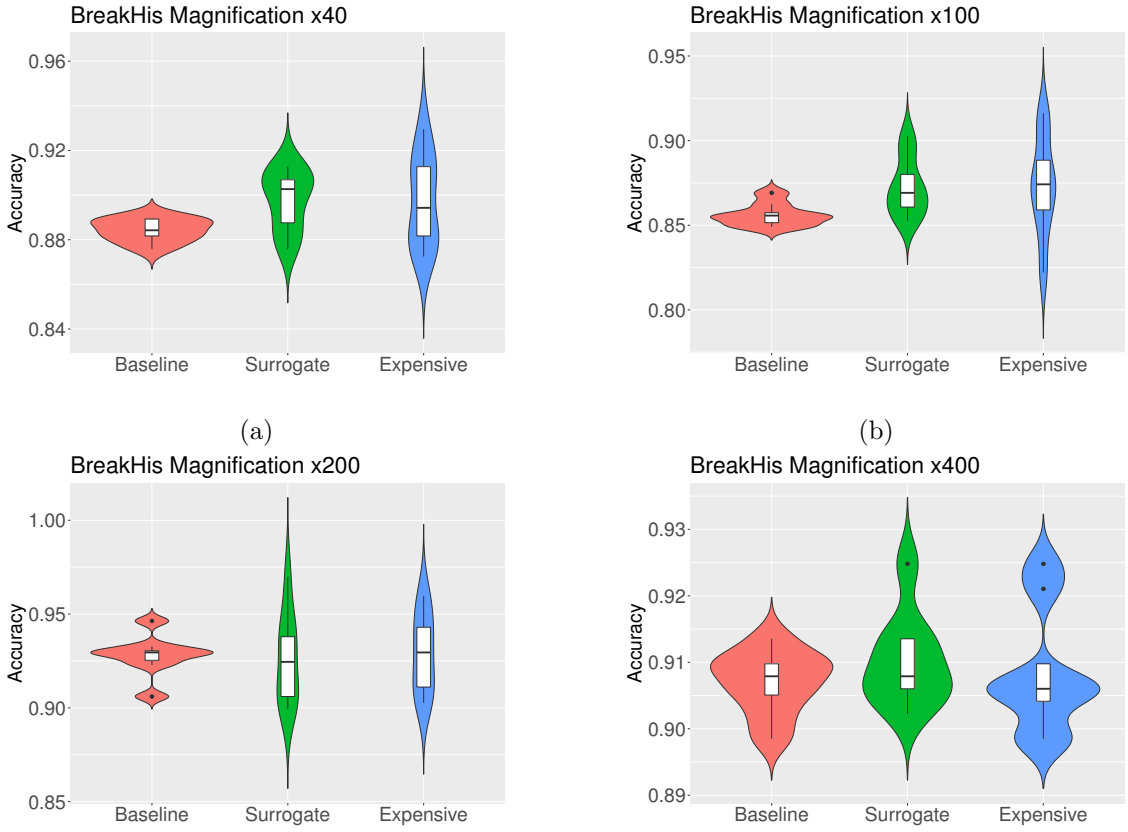


Figure 5.6: Fig. 5.6a through 5.6d plot violin plots, showing the distribution of accuracies of the architectures with the best fitness for each of the four data sets across 8 runs.

they have similar accuracies. Given the distribution of the models are not fully separable, we will focus on analysing the best individuals across all runs, before providing a limited statistical analysis focusing on the surrogate and expensive models in section 5.5.3.

The accuracies for the best-performing individuals across all runs for the baseline, surrogate and expensive models are listed respectively as such; for the $\times 40$ magnification the accuracies are 0.889, 0.913 and 0.930; for the $\times 100$ magnification are 0.869, 0.903 and 0.916; for the $\times 200$ magnification are 0.946, 0.970 and 0.960 and finally for the $\times 400$ magnification are 0.914, 0.925 and 0.925. In no instance did the baseline produce the best-performing individual. For the $\times 40$ and $\times 100$ magnifications, the expensive model produced the best-performing individual. For the $\times 200$ magnification the surrogate model produced the best-performing individual and the $\times 400$ magnification has a tie for the best-performing individual. In summary, while on average our surrogate and expensive models are as good as or better than the baseline, when we consider the best-performing individuals across all runs the surrogate and expensive models are better.

5.5.3 Comparing our Results against the State-of-the-art

Table 5.6 summarises approaches that have used the BreakHis data set from the last few years, including our proposed approaches. Most of these works have been taken from a comprehensive 2020 review paper [174]. For fair comparison, we have selected works that also do not use transfer learning as these represent manually crafted networks that have zero pre-training. Furthermore, many of these approaches use specialised feature extraction and pre-processing steps for histopathological data. As such, our goal is not necessarily to outperform previous state-of-the-art works, but rather to show that our neuroevolutionary technique can achieve similar results to hand-crafted architectures, even without specialised knowledge of the problem domain.

Looking at the last four columns from the right, we list the accuracies for the various magnifications. The last four rows correspond to our proposed approaches: NeuroLGP and its surrogate-assisted variant (NeuroLGP-SM). We can see that our results are in good accordance with the other works, in some cases outperforming other approaches. Notably, the results yielded by our two approaches on the $\times 200$ magnification are very competitive in terms of accuracy.

If we turn our attention to specifically comparing the surrogate and expensive models, we can see that each has very similar performances. For instance, the mean values (third and fourth last lines from bottom), the surrogate and expensive models are typically within 0.2% of each other and for the best (last two lines) the surrogate and expensive models are within 1-2%. For each dataset, a Wilcoxon rank-sum test was run with a significance level of $\alpha = 0.05$ to determine if the distributions of the surrogate and expensive differ and we concluded that they do not. In other words, this indicates the surrogate and expensive models are performing as well as each other, however, it should be noted our analysis is somewhat limited by the low run number.

5.5.4 Analysis of the Surrogate Model

Model Fitness

We use three metrics to determine how well our surrogate model performs in terms of predicting the fitness of our partially trained models. Firstly, we use the Mean Squared Error (MSE) between the predicted fitness and actual fitness. Values closer to 0 indicate our surrogate model is accurate in predicting fitness. Secondly, Kendall's Tau is used to measure the correlation between the predicted fitness and the actual fitness [104] and accounts for the monotonic relationship between these fitnesses, i.e., a non-monotonic trend will lower Kendall's Tau. A coefficient value of -1 indicates a perfect negative correlation, while a coefficient of +1 indicates a perfect positive correlation. Values close to 0 would indicate no discernible correlation. When considering all runs, a mean value

Table 5.6: Accuracy results for approaches using BreakHis dataset (×40, ×100, ×200 and ×400). Results from this work are highlighted in boldface and are presented in the last four rows (Where μ stands for mean and B stands for best). Aug: Data augmentation, Ens: Ensemble, WSI: wholes slide image, CNN: convolutional neural network, SVM: support vector machine, AE: auto-encoder, DBN: deep-belief network.

Work	Preprocessing	Patch/slide	Model	Training/Test	×40	×100	×200	×400
Gupta [178]	None	WSI	Ens.	70% / 30%	88.9	88.9	88.9	88.9
Sharma [179]	Mixed	WSI	Ens.	Not specified	81.7±2.8	81.2±2.7	80.7±3.4	81.5±3.1
Nahid [180]	None	WSI	CNN	Not specified	94.4	95.93	97.19	96
Nahid [181]	K-Mean Clustering	WSI	CNN	Not specified	85	90	90	90
Karthiga [182]	K-Mean Clustering	WSI	SVM	Not specified	93.3	93.3	93.3	93.3
Pratiher [183]	Gray Scale, Aug.	WSI	AE	Not specified	96.8	98.1	98.2	97.6
Badejo [184]	None	WSI	SVM	70% / 30%	91.1	90.7	86.2	84.3
Nahid [185]	Contrast Enhance	WSI	DBN	70% / 30%	88.7	89.06	88.84	87.67
Das [186]	Resize (370x230)	(224x224)	CNN	80% / 20%	89.52	85.3	88.6	88.4
Kumar [187]	Stain Normalisation	(64x64, 32x32)	CNN	70% / 30%	82±2.8	86.2±4.6	84.6±3.0	84±4.0
Aswathy [188]	Mixed	WSI	SVM	90% / 10%	89.1	89.1	89.1	89.1
NeuroLGP (μ)	Resize (64x64), Aug.	WSI	CNN	See Section 5.4	89.7±2.1	0.872±3.0	92.6±2.1	90.8±1.0
NeuroLGP-SM (μ)	Resize (64x64), Aug.	WSI	CNN	See Section 5.4	89.8±1.4	0.873±1.8	92.8±2.4	91.0±0.7
NeuroLGP (B)	Resize (64x64), Aug.	WSI	CNN	See Section 5.4	93	91.6	96	92.5
NeuroLGP-SM (B)	Resize (64x64), Aug.	WSI	CNN	See Section 5.4	91.3	90.3	97	92.5

Table 5.7: Average MSE and Kendall's Tau for different dataset magnifications.

Metric	Magnification Size			
	×40	×100	×200	×400
MSE	0.0037	0.0017	0.0014	0.0009
Kendall's Tau	0.6019	0.6791	0.6225	0.5647
R^2	0.5026	0.6665	0.7079	0.7786

close to 0 would indicate our surrogate model is performing poorly while a mean value closer to 1 would indicate an ideal-performing surrogate model for Kendall's Tau. Thirdly, the R^2 score [189], again measures the correlation between the predicted fitness and the actual fitness and ranges from -∞ to 1, but is insensitive to monotonic relationships and is a measure of how much variance in the data is explained by the prediction model. R^2 scores close to 1 would represent a model which perfectly fits the data.

Table 5.7 summarises the effectiveness of the surrogate model for each of the datasets. The low MSE values, as seen in the first row, show a relatively low error between the predicted and actual fitness across the four datasets. The Kendall's Tau values range from 0.5647 to 0.6791, indicating a strong positive correlation between the actual and predicted fitness. We can see that for ×40 we have R^2 of 0.5026 indicating a moderate level of fit being captured by the surrogate model while for ×100, ×200 and ×400, we have R^2 values of 0.6665, 0.7079 and 0.7786. There is also a general trend showing the R^2 score increasing with higher magnifications. This may be a result of more noise/artefacts present in the lower magnification images, making it more difficult for the surrogate model to predict based on the network outputs but further studies would be required to confirm this. Overall, the quality of fit and performance of the surrogate models based on the

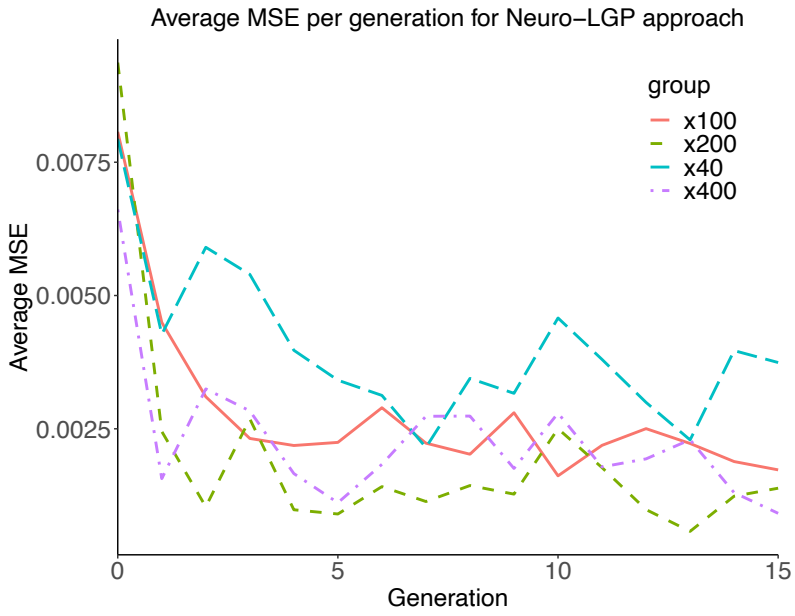


Figure 5.7: Avg. MSE between predicted *vs.* actual fitness over 15 generations for the surrogate approach. The relative stability shows the robustness of the surrogate approach.

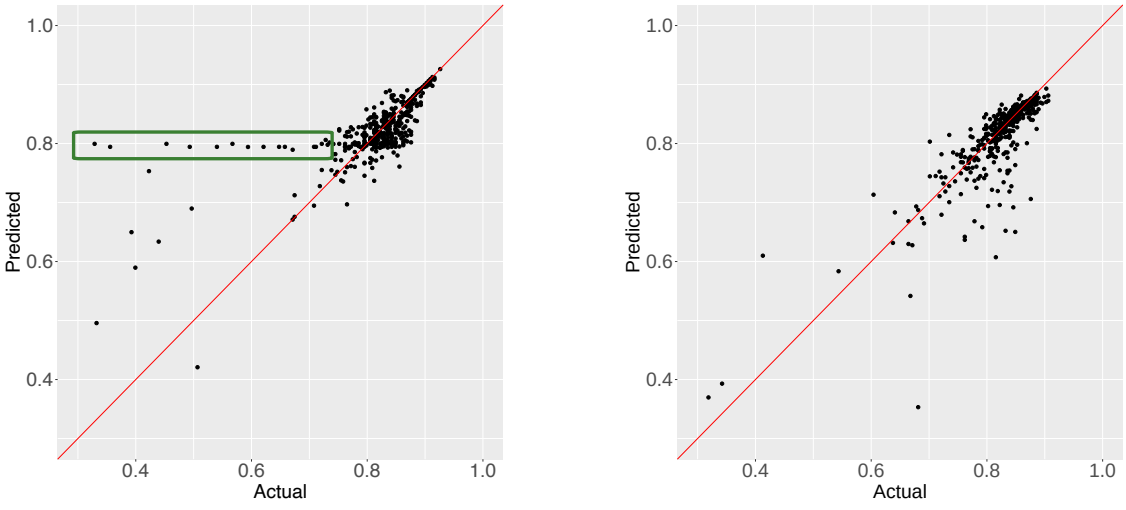
three metrics is very strong for each dataset.

Next, we plot the average MSE per generation to get an idea of how the predictive capability of the surrogate approach changes over time as seen in Fig. 5.7. MSE values that either decrease or remain stable are preferable, as increasing MSE values would indicate our surrogate model is losing its predictive capability as new individuals are introduced. We can see that after an initial drop in average MSE from the first couple of generations (x-axis), while there are some fluctuations in the average MSE, in general values remain relatively stable, demonstrating the robustness of our surrogate.

Fig. 5.8 details the quality of fit in terms of how well the predicted accuracies relate to the actual accuracies. In each figure, values closer to the red diagonal line, represent individuals that have a better quality of fit. We can see that the higher the accuracy the closer our predicted values are to the red line. This is a by-product of the surrogate model management strategy we employ. In essence, we are less concerned with the accuracies of poorer-performing models, instead, we have taken a more greedy approach favouring models that are better performing.

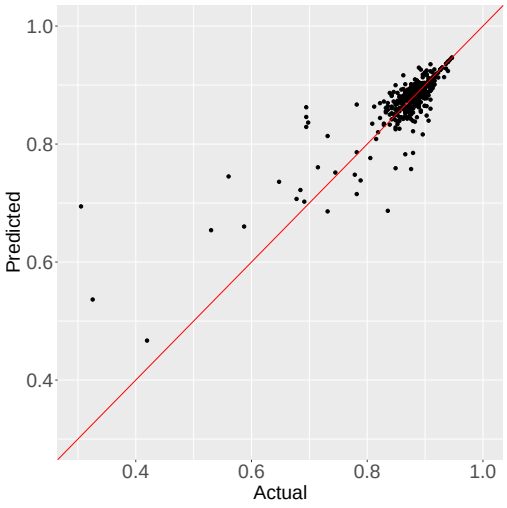
Fig. 5.8a informs us why the $\times 40$ magnification slightly under-performed in terms of the metrics as shown in Table 5.7. We can see that for the predicted values, on the y-axis, there are quite several individuals predicted to have $\sim$80% accuracy when the actual accuracies for these individuals range from 30-80%, as shown on the x-axis (green rectangle). A deeper dive into the results data (not shown here) revealed that this was a

result of one particularly bad-performing run and was not present in the other 7 runs.

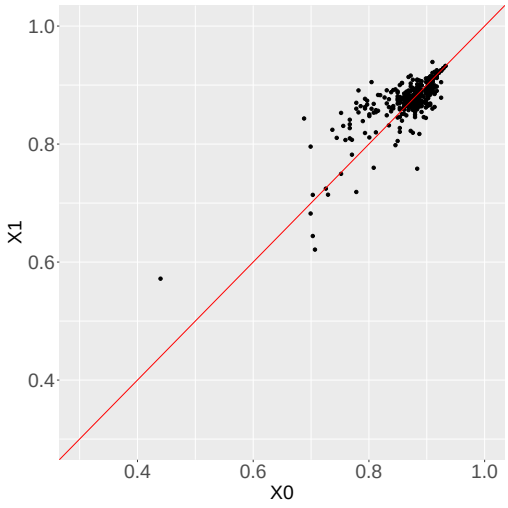


(a) ×40

(b) ×100



(c) ×200



(d) ×400

Figure 5.8: Predicted *vs.* actual accuracies (black points), for ×40, ×100, ×200, and ×400 datasets, shown in (a) – (d), respectively, across 8 independent runs for the surrogate-assisted approach (NeuroLGP-SM). The red line denotes where the accuracy for both predicted and actual are the same, where points closer to this line are preferential.

Time Analysis

Table 5.8 shows a comparison between the expensive and surrogate model in average runtime per GPU hour, in the second and third columns respectively. Of note is that the surrogate model typically is an order of magnitude higher for standard deviation, meaning there is greater variance associated with runtime for the surrogate model approach. The

Table 5.8: Average number of GPU hours per run for expensive and surrogate model. The last column denotes the reduction in time for each of the 4 datasets.

	Average GPU hours per run				
Mag.	**Expensive (hrs)**		**Surrogate (hrs)**		**Reduction**
	Mean	**Std**	**Mean**	**Std**	**Time**
×40	21.3	± 0.2	15.9	± 0.7	25.3%
×100	22.7	± 0.1	16.6	± 0.7	26.7%
×200	22.4	± 0.1	16.8	± 0.8	24.9%
×400	20.0	± 0.1	15.3	± 0.6	23.8%

last column shows the percentage in terms of time saved using the surrogate model over the expensive model. In general, we can see that given the parameters selected, we roughly save 25% in terms of GPU hours. Overall, if we consider the total time it took to run 8 runs for all 4 datasets, the expensive models took ∼28 GPU days and the surrogate models ∼21 GPU days, saving approximately ∼7 GPU days in all.

Energy Consumption Analysis

We perform a comparison of the estimated energy consumption of the surrogate and the expensive models using the Green-algorithm by Lannelongue et al. [190]. Eq. 5.2 is used to get this estimate,

$$E = r_t * (P_c * U + P_m) * PUE * PSF \tag{5.2}$$

where E is the energy consumed in KW/h, r_t is the runtime, P_c is the power draw for the computing cores and depends on the CPU/GPU model, U is the usage factor and determines how much of the core is in use, P_m is the power draw for the memory. PUE is the Power Usage Efficiency and is a measure of how much additional energy is needed to operate the data centre. PSF is the Pragmatic Scaling Factor and can be used to estimate the performance when multiple runs are taken into consideration. A conservative PUE value of 1.67 was used, which represents the worldwide average power usage PUE across all data centres [190]. For simplicity, PSF was kept as 1 as we are interested in the average runtime rather than the cumulative runtime.

The real usage factor was minimal for both the surrogate and the expensive models, even during training for the surrogate model and additionally, the CPU energy consumption is negligible compared to that of the GPU, so we have reported just the energy saved by the GPU, overall 2.79 KW/h are saved using the surrogate model. Considering the total energy saved across all 4 magnifications for 8 runs each, we save approximately 89.28 KW/h: 25% less energy is consumed using the surrogate approach. This is comparatively

similar to the time saving of ∼25%, as such, there is a relative one-to-one saving in both energy and time when using the surrogate model.

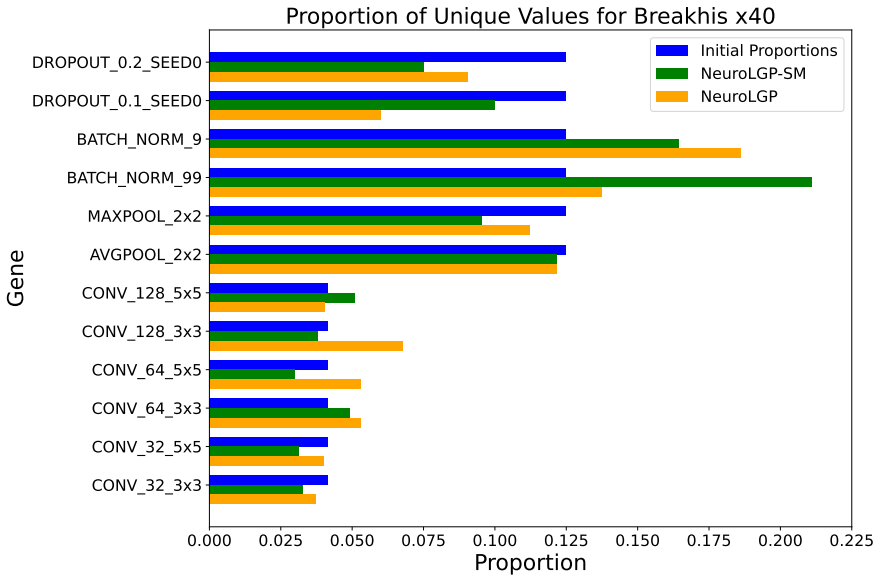
5.5.5 Analysis of Genotype

Figs. 5.9 (a - d) show the proportions of various genes for the four magnifications ×40, ×100, ×200 and ×400, respectively. Initial proportions (blue) represent the proportions of specific genes at initialisation (first generation). The surrogate (green, NeuroLGP-SM) and expensive (orange, NeuroLGP) models are also represented and denote the proportions in the final generation across all 8 runs. The comparison we make will aim to show how the proportions of the surrogate and expensive for specific layers change from initialisation (i.e., we compare green and orange proportions against blue).

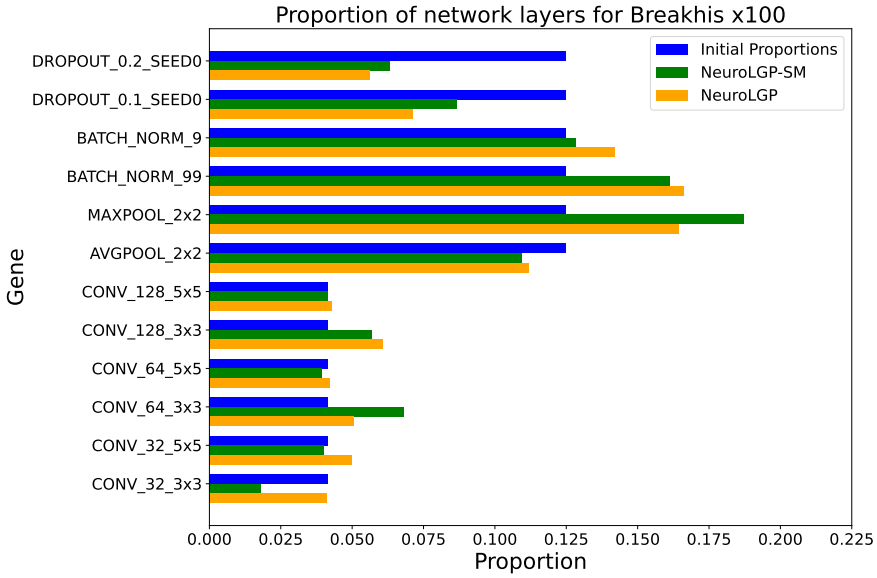
For the initial proportions, genes were proportioned based on their functional grouping. The four groups are dropout, batch normalisation, pooling, and convolutional layers. As such, each group makes up a quarter of the initial population and are subsequently divided again by specific genes. For instance, as there are 6 convolutional layer genes we divide $\frac{0.25}{6}$ to get the proportion for each convolutional layer.

Turning our attention to the surrogate and expensive models, we can see general trends in how the various gene groupings change by the final generation shown in green and orange for the NeuroLGP-SM and NeuroLGP, respectively. For instance, the dropout layer proportions sizes tend to decrease significantly by the final generation in the surrogate and expensive models. Similarly, the proportion of batch normalisation layers tends to increase. The pooling layers are more specific to the dataset. For instance, for ×100 and ×200 there is an increase specifically for the max pooling layer. On the other hand, for ×400 there is a general decrease, albeit the two models differ on the specific pooling layer they decrease. Again, for the convolutional layers there is a more specific trend for particular datasets. While there are some slight increases and decreases in proportion size for ×40, ×100 and ×200, for ×400 there are notable increases in proportion sizes for ×400 for convolutional layers using a 3×3 filter (Fig. 5.9d, CONV_32_3x3, CONV_64_3x3 and CONV_128_3x3).

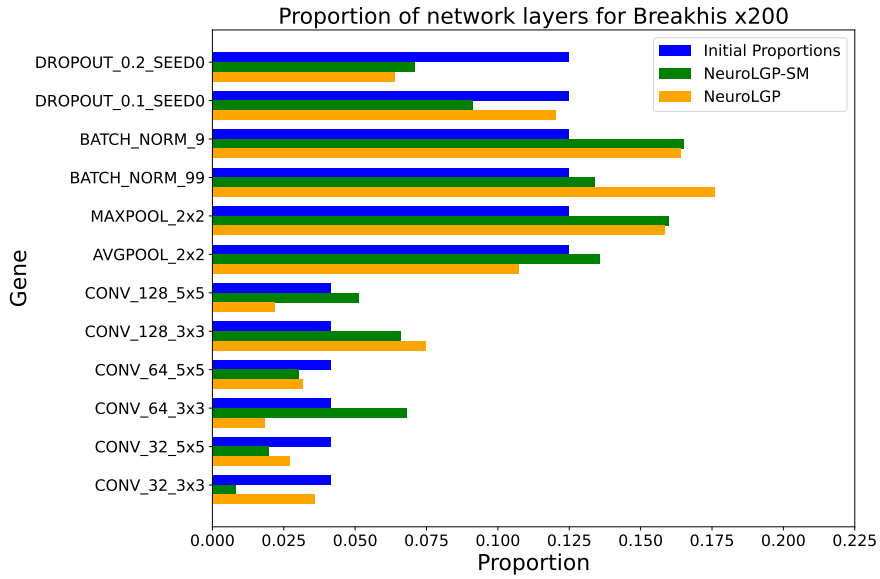
Some of these trends are unsurprising, we would expect that the proportion of dropout layers would be less, for instance, as CNN architectures are not as prone to overfitting [191]. It would seem that the decrease in pooling layers in the ×400 is being compensated by an increase in convolutional layers, in other words, some of the dimension reduction is being handled by the convolutional layers rather than the pooling. Further research could investigate how these proportions change relative to different resolution sizes. Interestingly, while in many cases the change in proportions are consistent for both the surrogate and expensive, in some instances the proportion sizes differ significantly,



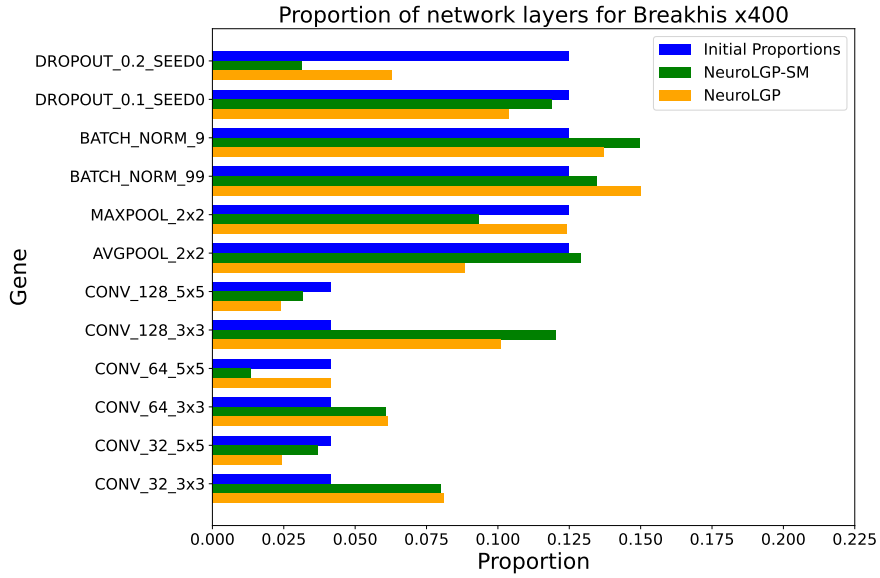
(a) Breakhis ×40 dataset



(b) Breakhis ×100 dataset



(c) Breakhis ×200 dataset



(d) Breakhis ×400 dataset

Figure 5.9: Proportion of evolved network layers for initialisation (blue) and final generations for NeuroLGP-SM (green) and NeuroLGP (orange) for ×200, and ×400 datasets

for instance, CONV_64_5x5 as seen in ×400. This is likely a result of solutions converging at different local optima, however, further analysis of the fitness landscape would be beneficial in gaining more insight.

5.5.6 Limitations of our Analysis

While a single run is the norm in terms of neuroevolution, we endeavoured to perform as many runs as possible, to add better confidence to our analysis. While we ran individual runs in parallel, both the surrogate and expensive approaches, for 8 runs each and for the 4 datasets, the total runtime amassed ∼80 GPU days. To conduct a full statistical analysis we would require another +80 GPU days of experimentation.

Furthermore, there are a number of aspects of LGP which are not covered in this Thesis but are beneficial for neuroevolution or are worth investigating further:

- LGP has both effective and non-effective portions of code within the genome. The non-effective portions of code (or Introns) would be interesting to study from the perspective of neutrality, an area which has yet to be explored in Neuroevolution [37, 127].
- LGP allows for control branches. This may allow for more complex design rules to be incorporated. Such rules may be important if we wish to design networks which use complex conditional-based workflows, for example, Mixture-of-Experts based architectures that use conditional computation [192]

5.6 Summary

In this Chapter, we demonstrate a novel surrogate-assisted neuroevolutionary approach, named Neuro-Linear Genetic Programming LGP surrogate model (NeuroLGP-SM). This approach makes use of Kriging Partial Least Squares (KPLS) to estimate the fitness of partially trained Deep Neural Networks (DNNs) using phenotypic distance vectors in a high-dimensional context. While phenotypic distance vectors have previously been used for traditional Artificial Neural Networks (ANNs) of only a few layers, their use in surrogate modelling for DNNs marks a significant leap forward in tackling high-dimensionality in neuroevolution. Our surrogate model management strategy makes use of a novel neuroevolutionary approach inspired by LGP, entitled NeuroLGP which allows us to evolve compact, robust and variable-length architectures.

We demonstrate that our approach is competitive or superior to other state-of-the-art handcrafted networks. Furthermore, our neuroevolution approach was shown to outperform a competitive baseline for both the expensive (NeuroLGP) and surrogate-based variant (NeuroLGP-SM) using four magnification subsets of the BreakHis dataset when

considering the best individuals across all runs. Using 3 metrics, in addition to visual analysis of quality of fit plots, we demonstrate the robustness of our surrogate model management strategy showing that our surrogate model not only accurately estimates the fitness of individuals but remains consistent over time and for all 4 datasets. We demonstrate that the proposed surrogate approach is 25% more efficient in terms of energy consumption and in terms of time saved, compared to a expensive variant that does not use surrogacy. Additionally, due to the unique encoding properties of our NeuroLGP approach, we can easily analyse the internal structures of the architectures, giving greater insight into the favourable components of the discovered architectures.

6

NeuroLGP-MB: Scaling Topological Complexity with a Pre-Selection Surrogate Model

In the previous chapter, we detailed a surrogate-assisted approach for neuroevolution inspired by LGP. In this chapter, we make some significant changes to our neuroevolution approach such that we can now encode branching and skip connections into our architecture allowing for much more complex topologies to be found. This is one of the key ingredients of modern DNNs. A strong motivating factor for why we wish to increase the complexity and search space is to demonstrate the applicability of the phenotypic distance approach to more challenging architecture structures. Furthermore, we introduce a new surrogate variant which uses a pre-selection method and was found to outperform a competitive baseline, the expensive model consisting of full evaluations and the previous surrogate variant for the BreakHis $\times 40$ and $\times 200$ datasets while performing similarly for a Chest X-Ray dataset.

In addition to performing a comparison of the four models in terms of their accuracy, we analysed the robustness of the new surrogate model variant and reported on the time savings. Remarkably, even though there is a significant overhead in evaluating individuals for the pre-selection method there was still a significant time saving ranging from 11.8 - 16.6% compared to the expensive model across the three datasets. We then perform an analysis of the depth and complexity on the elite members of the population demonstrating that deeper and relatively narrow networks are preferred. Finally, we analyse the effect of changing the initial epoch number for informing our surrogate model. It was found that a time saving of $\sim$41.5% was possible over the expensive model with just a single epoch, while maintaining relatively good accuracy, however, this came at the expense of the robustness of the surrogate model.

6.1 Introduction

In the preceding decades, EAs [20] have attracted considerable attention as viable search mechanisms for the automatic configuration of DNNs [57], either by evolving their network topologies, hyperparameters and/or weights. The aim of this evolutionary process, referred to as neuroevolution [37], is to find high-quality network architectures based on their performance which are typically attained by some measure of fitness relative to the problem domain, such as classification accuracy in classification problems.

One of the primary challenges to performing neuroevolution effectively is the significantly high computational overhead required in finding high-quality solutions. It has been noted, that for large DNN models, the cost of training a single epoch can be very expensive, where this issue is further compounded when many networks are required to train, as is the case in neuroevolution [21]. Another key challenge is finding a suitable evolutionary representation for finding complex architectures. Multi-branch and skip-connection architectures were popularised with models like googLeNet [29] and DenseNet [193], respectively and can help diversify how feature information is handled within DNN architectures, help improve their generalisation ability and help stabilise gradients [194], to name just a few benefits.

To this end, we propose an update to our original surrogate-assisted approach, NeuroLGP (see Chapter 5), such that we move from a chain-based topology to a multi-branch topology, with this new update referred to as NeuroLGP Multi-Branch (NeuroLGP-MB). As discussed in Chapter 5, one of the major challenges in distance-based surrogate-assisted neuroevolution is comparing networks with different topologies. While distance metrics can be created to compare such topologies using genotypic information, i.e, information pertaining to the structure of the architecture itself, these methods require clever encoding strategies for variable-length architectures. Moving to graph-based topologies further complicates the issue as the genotypic representations would require graph-edit metrics to compare distances [26], which can be computationally inefficient to calculate for larger graphs [24, 195].

The core aspects of this study and major developments over the original NeuroLGP, discussed in Chapter 5, have been summarised below:

- The original NeuroLGP representation has been updated from a *chain-structured representation* to a *multi-branch representation*, now referred to as NeuroLGP-MB, increasing the topological complexity and increasing the overall search space of potential architectures.

- Furthermore, the new NeuroLGP-MB approach has improved operations for handling invalid networks as a result of memory or architecture size constraints, have

been introduced. This approach attempts to silence problematic genes without destroying or significantly changing the chromosome.

- A new variation of the surrogate model has been introduced that uses an enhanced Design of Experiments (DoE) that pre-selects individuals to make up the first generation of our EA.
- An analysis was done on the elite members of the population, looking at how depth and complexity changed over time.
- An analysis was performed by varying epochs for the surrogate model, gaining insight into the potential time saving as well as offering a discussion on other aspects of varying epoches.

6.2 Methodology

6.2.1 NeuroLGP-Multi-Branch

In Chapter 5.3, we detailed the first version of the NeuroLGP approach. The new version Neuro-Linear Genetic Programming Multi-Branch (NeuroLGP-MB) extends our previous NeuroLGP approach to work with multiple branching connections by introducing the use of concatenation layers. Fig. 6.1 shows a comparison between the type of networks that can be produced by NeuroLGP and NeuroLGP-MB. On the left, shows a sequential network that is constructed in a linear fashion, as can be considered an example of a chain-structured topology. On the right, shows a multi-branch-based topology, demonstrating the types of networks capable of being produced by NeuroLGP-MB. This approach makes full use of the LGP paradigm, as even though the genotype can still be considered linear, the phenotypic representation can be used to create a Directed Acyclic Graph (DAG) and hence is capable of creating branching connections. The main implementation difference is that we extend the 2-register representation to a 3-register representation.

In this representation, the concatenation layer serves as a function of arity 2 (in other words, 3-register operand). In Fig. 6.2, we demonstrate how branching connections are constructed from the genotype (table on the left) and the resulting architecture (figure on the right). The rules for connecting edges of the graph are interpreted by traversing from the final layer to the first, represented by the descending index on the left column. The reasoning here is that multiple branches may connect to the same register at a later point and require special handling (discussed in more detail later). Register `r[0]` is a special register that is used to denote the final output of the architecture and its first occurrence in the 'out' column, using the bottom-up approach when looking at the table, allows us to see that it first occurs at index 1 (dark blue). We can now determine the next

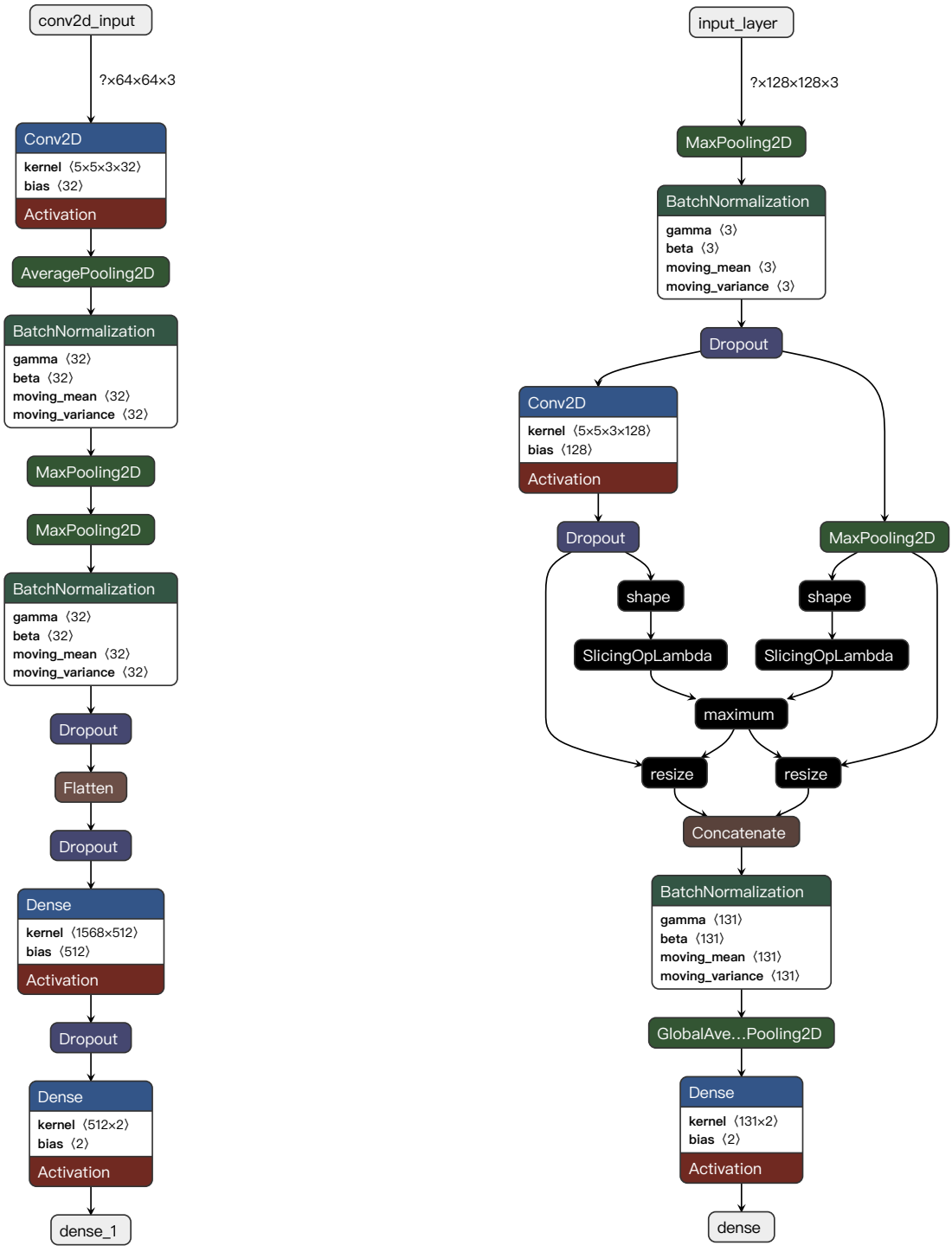


Table 6.1: Comparison of the original NeuroLGP architecture (left, detailed in Chapter 5) and the new NeuroLGP-MB architecture (right, detailed in this chapter). Graphs were generated using the open-source package Netron [13].

layer by looking at the input at index 1 (`r[2]` in 'In 1' column, dark blue) and seeing where it next occurs in the 'out' column, by moving up the table. This register next occurs in the 'out' column at index 6, which corresponds to the CONCAT layer which is our concatenation layer. From here, we can see that we have two input layers (`r[1]` in 'In 1' column, cyan and `r[3]` in 'In 2' column, magenta). These represent branching connections that correspond to Line 8 and 9, as `r[1]` is next found in 'out' of Line 9 and `r[3]` is next found in 'out' of Line 8. Since `r[2]` in 'In 1' column on Line 8 does not correspond to any proceeding outputs, it is determined to have terminated on this line and, as such, will connect straight to the input (terminating register denoted by a star symbol). This terminates the magenta branch. Line 9 connects to Line 16 via register `r[5]`and then terminates on `r[6]`, thus ending the cyan branch.

If we look at the diagram on the right, we can see how this process can represent the branching connections or a real architecture. Following the graph on the right-hand side, directly after the input layer, we see that we have a convolutional layer followed by a dropout layer. This corresponds to the cyan branch (Lines 16 and 9). Similarly, on the right we see the max pool layer after the input layer. This corresponds to the magenta branch (Line 8). Both these branches merge on the concatenation layer (Line 6, the black layers are used to denote operations to ensure dimensionality is correct and is not encoded in the genotype). Finally, the concatenation layer is followed by the batch normalisation layer before connecting to the global average pooling layer which is not part of the encoding. Earlier it was mentioned that multiple branches may connect on the same register. For instance, if `r[5]` was changed to `r[2]` on Line 16, then the input from Line 8 would no longer be terminating and the cyan and magenta branches would merge on the convolutional layer on Line 16 instead. In fact, many branches may reconnect on the same layer or at various layers throughout the chromosome.

In this work we use concatenation layers exclusively and have been popularised in such networks such as GoogleNet [29], InceptionV3 [196] and DenseNet [193]. Other popular architectures such as ResNet [30] use summation layers. While summation layers have not been encoded in this work it is still possible to use the NeuroLGP framework to encode these types of networks. Furthermore, the aforementioned concatenation-based architectures often use some form of modularity, such as repeating pre-defined blocks, using the same layers in each block, containing convolutions, activations, normalization, and pooling. These pre-defined blocks have not been encoded in the NeuroLGP, however it is possible to encode these. Nevertheless, the types of architectures found in NeuroLGP exhibit many features of these networks, such as two or more convolution layers stacked together, followed by a pooling layer, followed by a dropout layer before repeating into another stack of convolutional layers. The concatenation layers can occur at any point along the architecture pipeline, allowing for a diverse range of potential network topologies.

	Operand	Register		
Idx	**Layer**	**Out**	**In 1**	**In 2**
17	CONV_64_3x3_NOL2	r1	r0	
16	**CONV_128_5x5_KR00...**	**r5**	**r6***	
15	DROPOUT_0.3_SEED0	r4	r1	
14	AVGPOOL_3x3_2	r1	r1	
13	CONV_64_5x5_KR00001	r4	r2	
12	CONV_128_3x3_KR01:.	r4	r1	
11	CONV_64_3x3_KR00001	r1	r5	
10	BATCH_NORM_75	r1	r5	
9	**DROPOUT_0.5_SEED0**	**r1**	**r5**	
8	**MAXPOOL_3x3_2**	**r3**	**r2***	
7	CONV_64_3x3_KR01	r5	r5	
6	**CONCAT**	**r2**	**r1**	**r3**
5	CONV_64_5x5_KR00001...	r0	r5	
4	AVGPOOL_5x5_2	r4	r2	
3	CONV_128_5x5_KR01	r3	r0	
2	CONV_32_3x3_KR00001...	r1	r1	
1	**BATCH_NORM_1**	**r0**	**r2**	

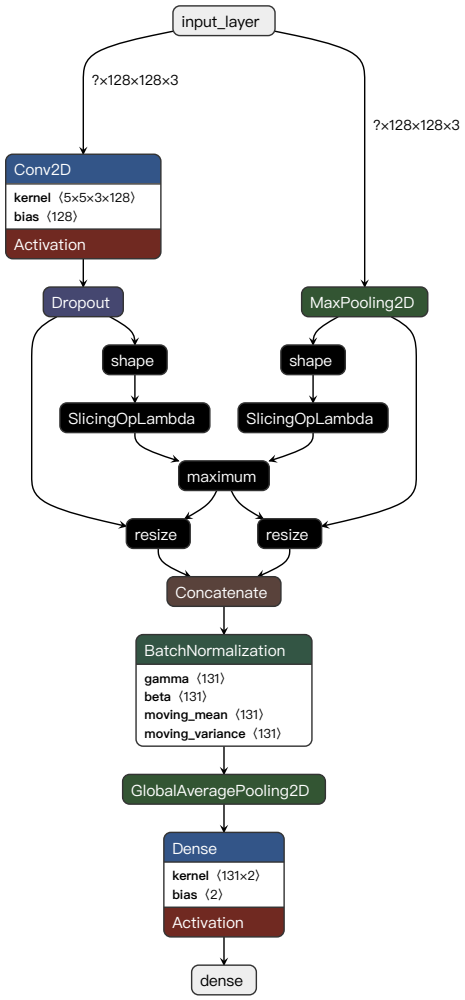


Table 6.2: *Left:* table showing the genotypic representation. Bold denotes effective code. Registers have been colour coded with full explanation in the text. *Right:* corresponding architecture for table on left. Following the effective code in the table allows us to see how this graph is constructed.

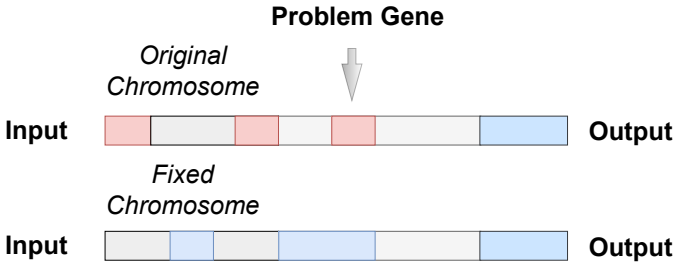


Figure 6.1: Demonstrating how gene silencing mechanism can be used to correct dimension error issues expressed by the chromosome. Effective code in red denotes potential problem layers. The arrow points to the layer where compilation failed. The left side of each chromosome represents the network input i.e., where image data is fed into and the right of each chromosome represents network output i.e., global average pooling layer. On the bottom, shows a chromosome where all effective code is blue, denoting a compilable network.

6.2.2 Genetic Operations

A number of additional changes have been implemented in NeuroLGP-MB. The original approach used a repair mechanism to ensure the first layer was always a convolutional layer, however, it is possible to compile networks with other layer types and still create valid networks, as such, this part of the repair mechanism has been removed to allow for less biased architectures to be found. The flattened structure layer in the original NeuroLGP approach has also been removed and replaced with a Global Average Pooling layer. The motivation here is that the size of the flattened layer in terms of the number of neurons was dependent on the total number of parameters and, as such, the dense layer could change dramatically depending on the number of parameters. With a Global Average Pooling Layer, this restriction is alleviated. Furthermore, by using a Global Average Pooling Layer the evolved networks will become more architecture-dependent.

The repair mechanism has been simplified in NeuroLGP-MB over the original version, where we use a more targeted approach inspired by epigenetics. While there are more extensive epigenetic-inspired operators in the literature [197, 198], our goal here is to replace the more destructive repair operator of NeuroLGP-MB with a simplified targeted mutation of the output register of the problem gene. During network compilation an error may be thrown for a specific layer, or problem gene, which will cause a negative dimension error. While this layer may be the cause of the dimension error, it may very well be that any previous layer or a combination of previous layers are the root cause. While iteratively removing these layers will solve the issue it can have the effect of drastically changing the chromosome up until that point. Another approach is to change the gene expression at that layer, thus silencing one or more of the genes up until that point

within the chromosome. This is akin to gene silencing in reverse genetics [199]. Fig. 6.1 demonstrates this concept of gene silencing as a repair mechanism.

The mutation operator has been updated such that is will mutate an individual multiple times. The reasoning here is that a large portion of the code is ineffective and, as such, mutating just a single time reduces the likelihood that an individual will change significantly. Additionally, other approaches have noted that when considering neuroevolution techniques where there are a low number of evaluations, having a high mutation rate can be beneficial [110]. Another update is the inclusion of a by-group mutation operation specifically for the operand. This will attempt to mutate an individual using the same group type. For instance, a mutation on a convolutional layer will always result in another convolutional layer, a dropout layer will always result in another dropout layer etc. No major change has been made to the crossover operator, as crossover still operates on effective portion of the code, however, crossover has the potential to alter the topology of the network if a concatenation layer lies within the crossover segment. Details of the crossover operator are discussed in Chapter 5.3.3.

6.2.3 Surrogate Model with Pre-Selection

Along with the baseline, expensive and surrogate models, we have included a variant which uses a pre-selection mechanism referred to as surrogate-PS. Surrogate models will often use Design-of-Experiments (DoE) as part of their modelling strategies. Popular examples of DoE methods include Latin Hypercube sampling and random sampling techniques [200]. Our particular modelling strategy is by design, greedy in nature, that is, we use individuals that we expect to give us the greatest improvement as training samples for our surrogate model. As such, we have focused on a sampling technique that selects the best individuals from an external population that is then used to inform both the initial surrogate model and populate the first generation of our EA.

The pre-selection method works as follows: (i) an external population is created that consists of P individuals we will sample from, (ii) we partially train these individuals to the same level as the initial epoch level of the surrogate, in other words, these individuals are only partially trained, (iii) we then select the top N individuals from P such that $N < P$ and where N is the same size as the initial population for our EA (iv) finally, we extract the phenotypic distance vectors from these individuals and subsequently fully train them. We then initialise our surrogate model with these individuals, where subsequently they make up the initial population of our EA. From here, the surrogate modelling strategy is the same as outlined in Chapter 5.3.2. In our experiments we use an external population size that is 5 times the size of the initial population, the initial epoch number is 10 and the full epoch number is 30.

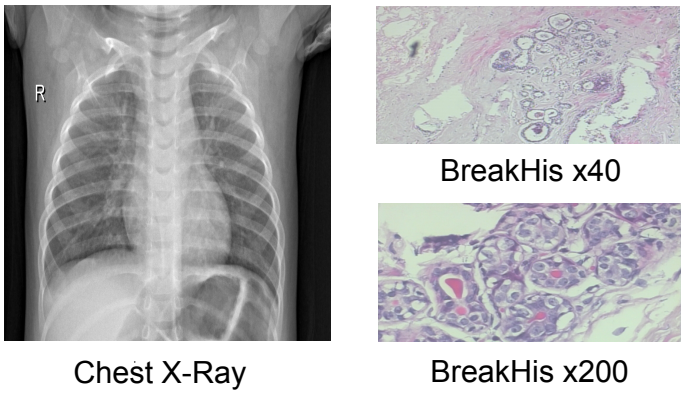


Figure 6.2: Sample images of the BreakHis and Chest X-ray datasets based on original resolutions. The BreakHis datasets (images on the right) incorporate different magnification sizes ($\times$40 and $\times$200)

6.3 Implementation Details

6.3.1 Datasets

The Breast Cancer Histopathological Image Classification (BreakHis) [173] is a binary classification dataset consisting of microscopic images as previously discussed in Chapter 5.4, however for this chapter we look specifically at two magnifications ($\times$40 and $\times$200) using a larger image resolution of 128x128 pixels (again using a down-scaled resolution from the original 700x460 pixels), with a 8-bit 3-channel RGB depth. Included as well, is the Chest X-Ray binary classification dataset for patients with pneumonia [201], originally consisting of 5,232 images, with 3,883 depicting images with pneumonia and 1,349 as normal. The images have been down-scaled to 128x128 pixels with a single grey-scale channel (from an original resolution of 1240$\times$840). Fig. 6.2 shows examples of the image data used in their original resolutions. The same data-augmentation techniques and up-sampling techniques are used as previously outlined in Chapter 5.4. Table 6.4 details the mutation and crossover rates, segment lengths for crossover operation, the initial chromosome lengths and minimum and maximum chromosome lengths (defined as program length in the table). Distances/lengths are based on effective length except for the distance of crossover points, the minimum and maximum chromosome lengths and initial chromosome length which consider the full chromosome length. Only the most salient information has been included in this table, full details of the configuration files, including additional details on the setup and a full list of evolvable layers have been included in Appendix B in Tables B.2. Experiments were run on Luxembourg's national MeluXina supercomputer using NVIDIA A100-40 GPU, which has 40GB of RAM available.

Dataset	# Train	# Test_2	Pheno. Size	Dimension
BreakHis ×40	1472	298	596	(128x128x3)
BreakHis ×200	1518	298	596	(128x128x3)
Chest X-Ray	3620	587	1174	(128x128x1)

Table 6.3: Details on datasets in terms of number of training and test instances, phenotype vector size and image dimensions. Validation, test_1 and test_2 use approximately the same number of instances.

Category	Attribute	Value
SURROGATE		
	METHOD	KPLS
	INITIAL EPOCH NUM	10
	INITIAL SURROGATE POPSIZE	30 (150 for Surrogate-PS)
	EXPENSIVE PROPORTION	0.4
EXPERIMENTAL SETUP		
	GEN SIZE	15
	POP SIZE	30
	ELITE PERCENTAGE	0.2
	SELECTION	TYPE: tournament, SIZE: 5, NUMBER OF TOURNAMENTS: 1
	CROSSOVER	RATE: 0.8 TYPE: linear, DISTANCE OF CROSSOVER POINTS: 80, MINIMUM PROGRAM LENGTH: 2, MAXIMUM PROGRAM LENGTH: 100, MAXIMUM SEGMENT LENGTH: 7, MAXIMUM DIFFERENCE IN SEGMENT LENGTH: 7
	MUTATE TIMES	5
	MUTATION RATE	0.5
	MUTATION OPERAND ONLY RATE	0.3
	MAX REGISTER	6
	INIT. CHROMOSOME LENGTH	30
	EPOCH NUM	30

Table 6.4: Details of experimental setup for NeuroLGP method.

6.3.2 Scaling Complexity

The above approach has been scaled in complexity in three ways. First, in terms of the phenotypic dimension size, second in terms of the search space by including additional layer types and third by increasing the topological complexity of possible neural networks. In Fig. 6.2, we demonstrate how we have scaled the phenotypic vector to a higher dimension by including the Chest X-ray dataset which has a phenotypic distance vector size of 1174, which is approximately twice the size of the phenotypic distance vector size of BreakHis ×40 and ×200. In traditional Kriging there would be a substantial barrier to increasing dimensions of this size as discussed in Chapter 5.2.

For the second approach, the number of available layers have been increased significantly. Some of the changes include the inclusion of TanH activation for convolutional

layers, a greater selection of hyperparameter settings for dropout, batch normalisation (in terms of momentum), regularisation, a greater variety of stride and kernel size for pooling layers and the option for no regularisation. The full details of these changes can be seen in Appendix B by comparing Tables B.1 and B.2, where Table B.1 details the configuration data used in Chapter 5 for the NeuroLGP approach and Table B.2 details the configuration data for NeuroLGP-MB.

Thirdly, we scale the complexity of the topology space. Prior to running any experiments, we ensured that our approach was robustly building each graph. To do this we generated 35 chromosomes, using a mixture of hand-crafted design and random generation to ensure that graphs of varying degrees of complexity were being generated. Fig. B.1, B.2 and B.3 sown in Appendix B represent a selection of these test architecture, showing the complexity of some of the topologies capable of being created. Some of the test cases include, but are not limited to layers that will branch to multiple edges with testing of up to degree 5 for some nodes, multiple concatenation layers within the same chromosome, concatenation layers occurring directly after the initial input, multiple branches occurring directly after a concatenation layer, to name a few. In some of these cases, special handling was required to ensure models were valid, for instance, if a concatenation layer occurs directly after the input we must ensure that the input layer splits and rejoins with both branches or determine if only a single branch will connect with the input.

6.4 Results

6.4.1 Discussion on Architectures Found

Fig. 6.5 shows a typical example of a portion of NeuroLGP-MB architecture (left) as seen in the final generation and a portion of a VGG16 architecture (right). VGG16 has been chosen as a reference as it has previously been used on the BreakHis dataset before [186]. VGG16 does not contain branching connections, but we can see some of the typical characteristics that we often see in CNN architectures such as 2-3 convolutional layers followed by a pooling layer. This pattern was observed in many of the networks produced by NeuroLGP but often there was a batch normalisation layer in between as seen on the left of Fig. 6.5.

Some additional characteristics of the architectures found by the NeuroLGP approach were batch normalisation layers occurring directly after the input layer. It is important to note, other than rescaling and data augmentation no preprocessing was done on the input data. Potentially, batch normalisation helps with variability in the contrast and brightness of the histopathological images but further analysis would be required to determine this. There was also a tendency in some networks to have a greater number of pooling layers

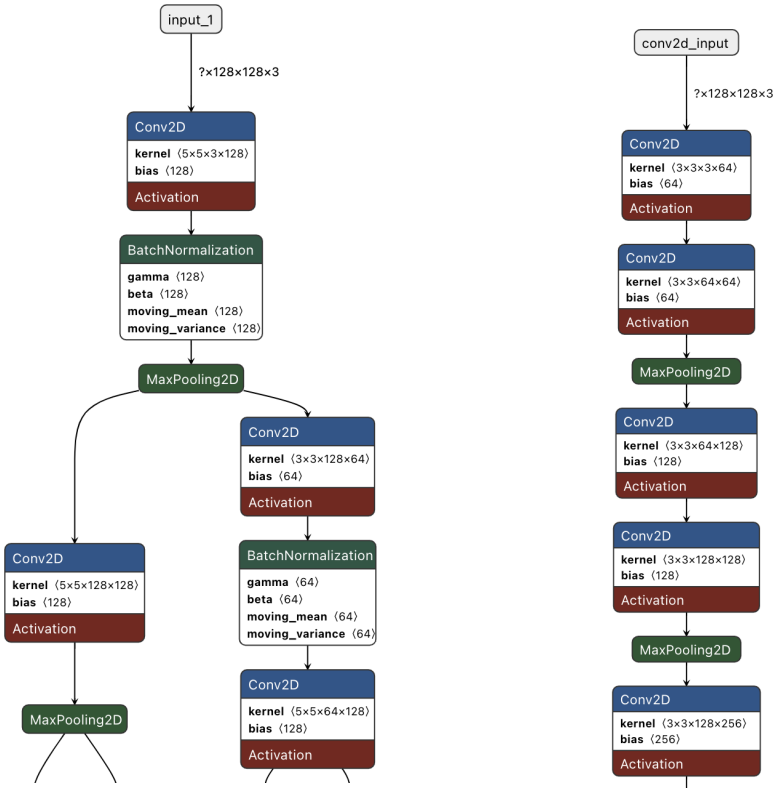


Table 6.5: Comparison of the NeuroLGP-MB architecture (left) and the VGG16 (right, detailed in this chapter). Graphs were generated using the open-source package Netron [13].

and dropout just before the global average pooling layer at the end of the architecture pipeline.

6.4.2 Performance Analysis

Table 6.6 summarises the results for the BreakHis ×40, BreakHis ×200 and Chest X-Ray for each of the three experiments, including training, validation, test_1 and test_2, where the best individual is selected from each run and averaged across 4 runs. Test_2 is of particular interest as this represents the with-held dataset from the entire evolutionary process (test_1 is used to determine the fitness of each network and is used by our EA but is only withheld within the context of each individual network). The table is read as follows: the first column represents the dataset that is being considered, the second column denotes the particular experiment type and the subsequent columns represent the training, validation, test_1 and test_2 datasets, respectively. Test_2 is of particular interest as this represents the with-held dataset from the entire evolutionary process.

A motivation for including training accuracy is to get an idea of how well the training, in general, correlates with test_2 accuracy. For instance, if we have a low training accuracy

Dataset	Experiment	Train	Val	Test_1	Test_2
BreakHis ×40	Baseline	0.822 ± 0.018	0.862 ± 0.026	0.837 ± 0.022	0.887 ± 0.012
	Expensive	0.897 ± 0.032	0.894 ± 0.037	0.873 ± 0.038	0.907 ± 0.029
	Surrogate	0.865 ± 0.054	0.898 ± 0.032	0.873 ± 0.044	0.904 ± 0.029
	Surrogate-PS	0.907 ± 0.061	0.910 ± 0.032	0.889 ± 0.040	0.919 ± 0.032
BreakHis ×200	Baseline	0.944 ± 0.017	0.883 ± 0.033	0.890 ± 0.008	0.897 ± 0.014
	Expensive	0.941 ± 0.021	0.912 ± 0.024	0.911 ± 0.013	0.930 ± 0.030
	Surrogate	0.935 ± 0.044	0.890 ± 0.035	0.905 ± 0.011	0.919 ± 0.031
	Surrogate-PS	0.964 ± 0.010	0.923 ± 0.025	0.909 ± 0.020	0.939 ± 0.010
Chest X-Ray	Baseline	0.796 ± 0.119	0.894 ± 0.016	0.914 ± 0.020	0.912 ± 0.015
	Expensive	0.856 ± 0.107	0.895 ± 0.023	0.920 ± 0.030	0.917 ± 0.016
	Surrogate	0.863 ± 0.051	0.900 ± 0.018	0.914 ± 0.018	0.909 ± 0.021
	Surrogate-PS	0.865 ± 0.067	0.906 ± 0.011	0.921 ± 0.006	0.914 ± 0.014

Table 6.6: Results for average train, validation, test_1 and test_2, where the best individual is selected from each run, across 4 runs, for each experiment type for the BreakHis ×40, BreakHis ×200 and Chest X-Ray datasets.

and low test_2 accuracy, this may indicate a degree of underfitting. Conversely, if we have a high training accuracy and lower test_2 accuracy, this may indicate overfitting. In general, the differences are within the range of ± 0.03, with a notable exception of the Chest X-Ray dataset, where the training set accuracy is substantially lower than the test_2 set. In this case, the Baseline approach has the lowest training accuracy of 0.796, however, this may be a result of the relatively high variance 0.119. Regardless, this seems to suggest that every experiment type is to some extent underfitting the Chest X-Ray dataset.

Looking specifically at test_2 for the four experiment types for BreakHis ×40 we can see that on average the expensive, surrogate and surrogate-PS accuracies are higher than the Baseline. Of these methods, surrogate-PS has the highest accuracy on average with 0.919 ± 0.032 *vs.* 0.887 ± 0.012, 0.907 ± 0.029 and 0.904 ± 0.029, for baseline, expensive and surrogate models, respectively. Again if we look at BreakHis ×200 we can see that surrogate-PS outperforms the other methods with an accuracy of 0.939 ± 0.010 *vs.* 0.897 ± 0.014, 0.930 ± 0.030 and 0.919 ± 0.031, for baseline, expensive and surrogate models, respectively. In the case of Chest X-Ray we can see that the accuracies are all relatively similar, 0.912 ± 0.015, 0.917 ± 0.016, 0.909 ± 0.021 and 0.914 ± 0.014, for baseline, expensive, surrogate and surrogate-PS, respectively. For context, Kermany et al. [202], note a 92.8% accuracy on the test set for a transfer-based learning architecture, however, it should be noted in our work we use a different training, validation and test split than that of the original dataset [201]. Regardless, we can see that the baseline is performing competitively with the state-of-the-art. In general, we can say, with the exception of Chest X-Ray the surrogate-PS approach has the highest accuracy.

Table 6.7: Average MSE, Kendall's Tau and R^2 for different datasets. The naming convention has been shortened for ease of reading, such that, the original surrogate model is denoted as 'Sur' where the pre-selection method is denoted as 'Sur-PS'.

Metric	Datasets					
	×40		×200		Chest X-Ray	
	Sur	Sur-PS	Sur	Sur-PS	Sur	Sur-PS
MSE	0.0067	0.0035	0.0017	0.0031	0.0372	0.0173
Kendall's Tau	0.6536	0.6480	0.6480	0.7435	0.6298	0.6790
R^2	0.6239	0.6185	0.9373	0.7458	0.4536	0.6082

6.4.3 Surrogate Analysis

Three metrics are used to determine how well the surrogate model performs in predicting the fitness of our partially trained models: (i) the Mean Squared Error (MSE) gives a measure of how accurate our model is in terms of predicting fitness where values close to 0 are preferable, (ii) Kendall's Tau is used to measure the correlation between the predicted fitness and the actual fitness [104], where values close to 0 indicates no correlation, -1 a perfect negative correlation, +1 a perfect positive correlation, and (iii) the R^2 score again measures the correlation between the predicted fitness and the actual fitness, but is insensitive to monotonic relationships unlike Kendall's Tau and ranges from $-\infty$ to 1 and is a measure of how much variance in the data is explained by the prediction model, where values closer to 1 are preferable.

Table 6.7 summarises the effectiveness of both the surrogate model and the pre-selection surrogate model for each of the datasets for the 3 metrics. If we compare the metrics for the base surrogate and pre-selection variants against each other (sur *vs.* sur-PS), we can see that, on the surface, the metrics tend to vary as to which is preferable, for instance, if we look at ×40 we can see that surrogate-PS has a lower MSE of 0.0035 *vs.* 0.0067 but the Kendell's Tau and R^2 are preferable for the surrogate model with 0.6536 *vs.* 0.6480 and 0.6239 *vs.* 0.6185. Only in the case of Chest X-Ray do the metrics seem to be consistently preferable. These results are unsurprising, to some degree, if we consider the mechanism for how the surrogate model is informed. Even though the pre-selection model is informing our surrogate with better solutions, this does not mean that the correlation between the phenotypic distance vector and actual fitness will be improved. Later, in Chapter 6.4.6, we look into how changing correlation by varying epoch size affects surrogate model performance. The main takeaway for now is that the performance of the surrogate-PS model is in general comparable to the surrogate model.

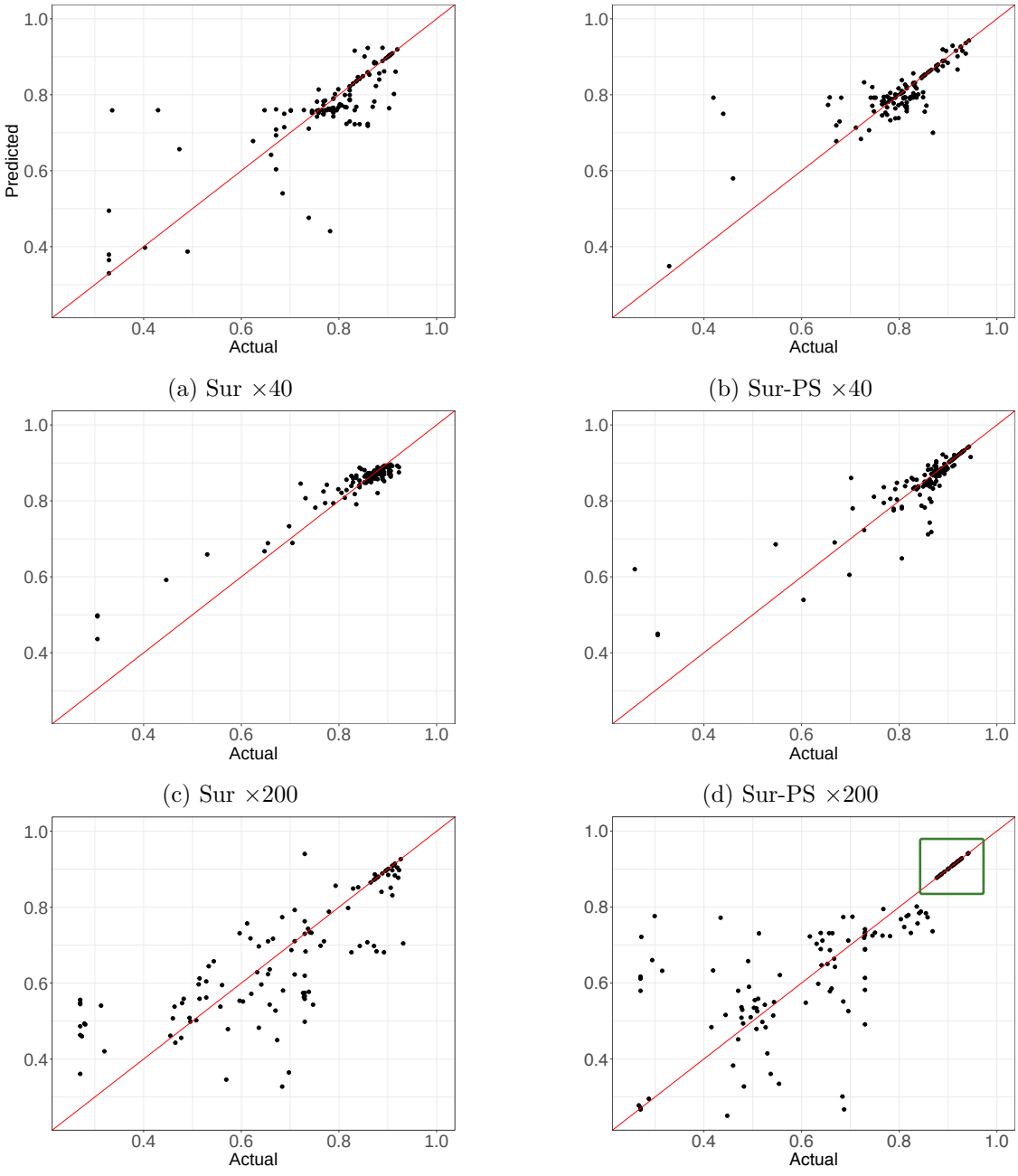
Now, if we focus specifically on the datasets we can see that then Kendall's Tau and R^2 values show a relative moderate to high correlation, with values typically above 0.6 (with the exception of Chest X-Ray surrogate model that has a R^2 value of 0.4536). The

MSE values are low (preferable) for BreakHis ×40 and BreakHis ×200, however for Chest X-Ray the MSE values are an order of magnitude higher. Next, we will look specifically at the predicted *vs.* actual accuracy plots, which detail the quality of fit.

Fig. 6.3 details the quality of fit in terms of how well the predicted accuracies relate to the actual accuracies, where in each subplot, values closer to the red diagonal line, represent individuals that have a better quality of fit. The first column relates to the surrogate model (Fig. 6.3a, 6.3c, 6.3e) and the right column relates to the surrogate-PS model (Fig. 6.3b, 6.3d, 6.3f). We can see that our previous analysis of the metrics conforms with the plots, in that, there is a relatively moderate to high correlation between the predicted *vs.* actual accuracy, however, we can see there is a much higher degree of variance with Chest X-Ray. Of particular interest is there is a notable separation between the best-performing individuals (green rectangle) and individuals with an accuracy of less than 0.83 in Fig 6.3f. We can also see that these individuals lie very close to the red line indicating a high surrogate performance for the individuals. Our analysis from Chapter 6.4.2, demonstrated that the baseline, performed competitively with the other three methods, managed to find networks comparable with the state-of-the-art. What is likely occurring in Fig 6.3f is that because such high-performing individuals are sampled in the pre-selection stage, then subsequently the surrogate-PS model struggles to find better-performing individuals to re-inform the surrogate model. In general, we can see that in the surrogate model for BreakHis ×40 and BreakHis ×200 is adequately predicting. For Chest X-Ray a higher initial epoch number may be required in order to gain a better estimate of the performance.

6.4.4 Time Analysis

Table 6.8 shows a comparison between the expensive, surrogate and surrogate-PS model in average runtime per GPU hour, in the second, third and fourth columns respectively. We can see that there is a relatively high variance in the number of GPU hrs of the surrogate and surrogate-PS models, but both approaches are significantly faster than the expensive model. If we look at the percentage reduction compared to the expensive model (last two columns), we can see that the percentage saves for the original surrogate model ranges from 23.2 - 28.7% across the three datasets. However, if we look at the surrogate-PS model, considering the significant overhead in evaluating individuals for this method (150 individuals trained to 10 epochs), there was still a significant time saving ranging from 11.8 - 16.6%. Based on the results of Chapter 6.4.2, where the surrogate-PS model was found to perform the best for BreakHis ×40 and BreakHis ×200, there is a clear advantage in considering the surrogate-PS model.



(a) Sur ×40

(b) Sur-PS ×40

(c) Sur ×200

(d) Sur-PS ×200

(e) Sur Chest X-Ray

(f) Sur-PS Chest X-Ray

Figure 6.3: Predicted *vs.* actual accuracies (black points), for BreakHis ×40, ×200, and Chest X-Ray datasets, shown in (a) – (f), respectively, across 4 independent runs for the original surrogate-assisted approach and surrogate-PS). The red line denotes where the accuracy for both predicted and actual are the same, where points closer to this line are preferential.

	GPU hours per run			% Reduction	
Dataset	**Exp (hrs)**	**Sur (hrs)**	**Sur-PS (hrs)**	**Sur (%)**	**Sur-PS (%)**
BreakHis ×40	25.46 ± 0.05	19.55 ± 0.23	21.45 ± 0.24	∼23.2	∼15.8
BreakHis ×200	27.32 ± 0.02	20.42 ± 0.74	22.77 ± 0.30	∼25.3	∼16.6
Chest X-Ray	21.13 ± 0.02	14.86 ± 0.16	18.63 ± 0.13	∼29.7	∼11.8

Table 6.8: Average number of GPU hours per run for expensive, surrogate and surrogate-PS models. The last two columns denote the reduction in time for surrogate and surrogate-PS when compared against the expensive model.

6.4.5 Network Depth and Complexity

Fig. 6.4 shows the change in average number of layers of the elite population across 15 generations, for BreakHis ×40 and BreakHis ×200 datasets. The elite population represents individuals that have been retained, as such, representing the best individuals and allows us to analyse how the depth of networks vary across generations. The elite population is 20% of the total population size for all datasets. Looking at Fig. 6.4, we can see that in both datasets, surrogate-PS (red line) tends to generate individuals with the largest number of layers, or greatest depth, followed by the expensive (blue line) and then the surrogate (green line). These results show that as the accuracy in each generation increases the depth of the networks found also increases.

Fig. 6.5 again shows the change in the average number of layers, focusing now on the Chest X-Ray dataset. We can see that on average the number of layers is not changing over time for the expensive, surrogate and surrogate-PS models. There may be a number of contributing factors as to why the depth does not increase. There is a strong indication from our previous analysis that random sampling, akin to the baseline model, is capable of finding competitive networks, in other words, the fitness landscape of the Chest X-Ray may be relatively uncomplicated where shallower networks perform well. Potentially, initialising the networks with a larger depth may improve the search capability for Chest X-Ray. Another possibility is to increase the maximum segment lengths during crossover to see if larger segments will result in more growth.

Next, we perform an analysis of the topological complexity of the network. While the number of parameters are often used to denote network complexity, in this analysis we focus primarily on topological complexity. To this end, we define complexity based on the layers that result in nodes (i.e., layers) of more than degree 2 (i.e., have more than 2 edges). In traditional NeuroLGP, all layers will have a single input and output and as such will at most have degree 2. On the other hand, in NeuroLGP-MB if we have a concatenation layer, then this will mean that we will have at least two nodes that have degree three (two nodes as a result of the layer in which the branch occurs and the layer in which the branch reconnects). While this does not give us an exact measure of the

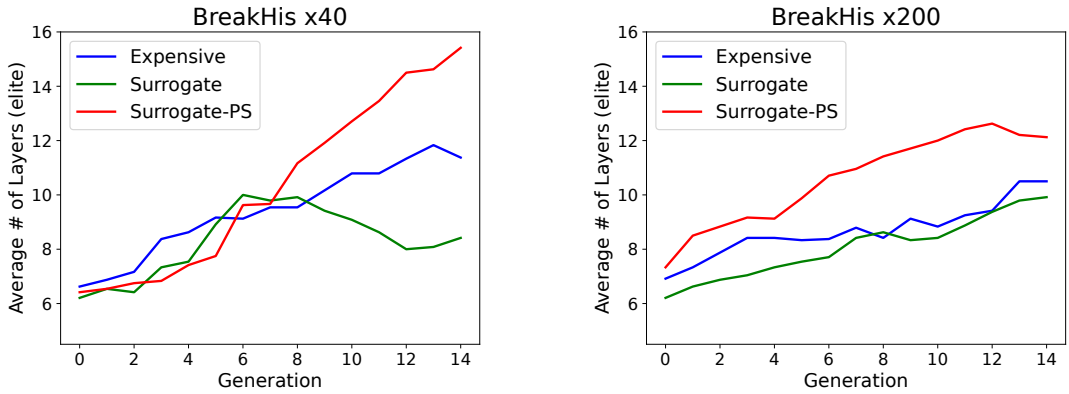


Figure 6.4: Average number of layers for expensive, surrogate and surrogate-PS across 15 generation for BreakHis ×40 and BreakHis ×200 datasets

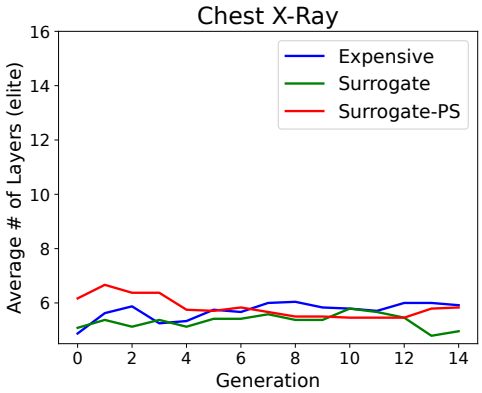


Figure 6.5: Average number of layers for expensive, surrogate and surrogate-PS across 15 generations for Chest X-Ray dataset

width of the neural network, it does give us some indication of the overall topological complexity.

As the concatenation layer is the contributing factor in determining network topological complexity, we perform an analysis of their proportions relative to the elite population members in the final generation specifically. Table 6.9 shows the proportions of concatenation layers. In each case, the initial proportions of concatenation layers were set at 0.1. We find, that on average, there does not appear to be any specific relationship across experiment types or between final depth with complexity. For instance, if we look at BreakHis ×200 for the surrogate model we can see that the proportion is 0.3, or 1 in 33 layers. Given that the final generation on average has ∼10 layers (with 6 elite members per run) then approximately 1 in every 3 elite members will have a concatenation layer per run on average. Conversely, if we look at BreakHis ×200 for the surrogate-PS model we can see that the proportion is 0.12, or approximately 1 in 8 layers. Given that the final

	Concatenation Proportions		
Dataset	**Expensive**	**Surrogate**	**Surrogate-PS**
×40	0.06	0.07	0.04
×200	0.06	0.03	0.12
Chest X-ray	0.12	0.11	0.06

Table 6.9: Proportions of concatenation layer for elite population members in final generation for expensive, surrogate and surrogate-PS, for BreakHis ×40, BreakHis ×200 and Chest X-Ray datasets. Proportion values are between 0 and 1.

generation on average has ∼12 layers (with 6 elite members per run) then approximately each elite member will have ∼1.5 concatenation layers per run on average.

However, a deeper dive reveals that specific runs had a higher degree of concatenation layers than others. This is understandable to some degree, as a model in a specific run with no concatenation layers but which is the best-performing individual, is less likely to produce offspring with a high number of concatenation layers and vice versa. In general, this analysis would indicate that during evolution there is a particular focus on depth, with a tendency to find deeper networks which are relatively less topologically complex. Despite this, we find that some experiments were preferencing concatenation layers for certain runs. Future work could look at promoting concatenation layers, as well as other branching operations through mutation and crossover.

6.4.6 Varying Epoch Length

Table 6.10 shows the test_2 accuracy for varying initial epoch numbers of surrogate training for the BreakHis ×40 and BreakHis ×200. For instance, if we look at column 3 named 'Epoch 1' this refers to the performance of the surrogate model if the phenotypic vector is obtained only after the first epoch. In the case of 'epoch 3' and 'epoch 10' the phenotypic vector is retained after the third and tenth epoch respectively. All other parameters are kept consistent such as the proportion of the population to inform the surrogate and in each case, the full number of epochs is 30.

Focusing specifically on MSE, Kendalls's Tau and R^2 (first, second and third rows for each dataset) we can see that in general epoch 10 has the better performance, i.e., MSE closer to 0 and Kendalls's Tau and R^2 closer to 1. However, for epochs 1 and 3, MSE and Kendall's Tau vary as to which metrics are preferable. The R^2 tends to have the most dramatic drop when epoch size is decreased as seen on the third row for each dataset.

Interestingly, the decrease in epoch size does not necessarily correspond to a drop in accuracy which can be seen on the fourth row for each dataset in Table 6.10. It is likely that the higher level of uncertainty is helping to retain networks that have the potential to be high-performing once evolved, however, more analysis would be required to determine

if this is the case. What is important to note though, is that lower epoch numbers, are not generally better, for instance in the case of epoch 3 where BreakHis ×40 had the lowest average accuracy of 0.888 ± 0.018, compared to epoch 1 and 10, 0.914 ± 0.020 and 0.904 ± 0.029, respectively. Knowing the exact number of epochs to train a surrogate model *a priori* is one of the limitations of surrogate modelling and although the models have a relatively high accuracy for lower epochs, in general, we can say that it is more robust to ensure there is a high correlation between the predicted *vs.* actual accuracies to ensure consistency.

Finally, we look at the percentage of time saved over the expensive model as seen on the fifth row for each dataset in Table 6.10. We can see that training for 3 epochs saves roughly ∼36% over the expensive model for both models, while training for just a single epoch saves ∼41.5% for both models. This demonstrates that varying epoch numbers can have a dramatic effect on time saved over expensive models. However, as mentioned this comes at the expense of the robustness of the surrogate model.

	Initial Epoch for Surrogate Training			
Dataset	**Result**	**Epoch 1**	**Epoch 3**	**Epoch 10**
BreakHis ×40	MSE	0.0109	0.0049	0.0067
	K. Tau	0.5724	0.5240	0.6536
	R^2	0.4467	0.5189	0.6239
	Acc.	0.914 ± 0.020	0.888 ± 0.018	0.904 ± 0.029
	Time	∼ 41.5%	∼ 36.9%	∼ 23.2%
BreakHis ×200	MSE	0.0128	0.0147	0.0017
	K. Tau	0.6377	0.5709	0.6272
	R^2	0.1703	0.3065	0.9373
	Acc.	0.907 ± 0.034	0.940 ± 0.012	0.919 ± 0.032
	Time	∼ 41.5%	∼ 35.2%	∼ 25.3%

Table 6.10: Average test_2 accuracies for different initial epoch numbers during surrogate training, for the BreakHis ×40, BreakHis and ×200.

6.4.7 Discussion and Limitations

While a single run is the norm in terms of neuroevolution, again as was the case with the experimentation in Chapter 5, we sought to perform as many runs as possible to add validity to our analysis. In total, we amassed ∼20 GPU days across 4 runs for the various experiments. It should be noted that while our experimentation relating to Chapter 5 and this chapter are computationally expensive for a modest budget, the number of full evaluations for our DNN architectures is relatively low at 1800 evaluations for the expensive model (450 per run) and 456 evaluations for the surrogate models (114 per run), where other approaches report 1000s of GPU days or 10,000s of evaluations [80, 85].

The predictive performance of the surrogate model is linked to how correlated the phenotypic distance vector is with the classification accuracy after the initial number of epochs. A limitation here is knowing the ideal epoch to initially train *a priori* is difficult, however future work could investigate using correlation metrics between the predicted versus the actual dataset during runtime. Furthermore, the predictive curve for each network will change for different architectures, for instance, the Freeze-thaw Bayesian Optimisation technique tackles this by allowing a start-stop mechanism for choosing which networks to allow to run [112].

Recent developments have been made in analysing the topological complexity of neural networks [203]. While neural network complexity is often defined in terms of the number of parameters, analysing the graph structures of DNNs could grant researchers a more holistic way of describing neural network architectures. In our work, we looked at the proportion sizes of concatenation layers but in the future, we would like to study the graph structures in more detail.

6.5 Summary

Multi-branch topologies for the neuroevolution of DNNs can be a challenging prospect for distance-based surrogate models when considering the structure of the architecture alone. In this work, we adapt our previous neuroevolution technique, referred to as the original NeuroLGP approach, which strictly uses a chain-structured topology, so that it now encodes a multi-branch topology. This new approach, named NeuroLGP multi-branch (NeuroLGP-MB) has further been adapted with an update to the surrogate-assisted approach such that it used a pre-selection method to better initialise the surrogate model.

We tested our new approach on BreakHis $\times 40$, BreakHis $\times 200$ and Chest X-Ray dataset, using four models: a baseline, expensive, surrogate and the new surrogate-PS model. We found that the new pre-selection variant produced the best networks on average for BreakHis $\times 40$ and BreakHis $\times 200$. Interestingly, the Chest X-Ray dataset produced results that were similar across all models. We found that in general, the new surrogate-PS was also robust like the original surrogate approach, while also maintaining a time advantage over the expensive model with an 11.8 - 16.6% time reduction compared to the expensive model. An analysis of the depth and topological complexity revealed the preference for deeper networks that are not overly topologically complex. Furthermore, our analysis of varying initial epoch numbers to inform the surrogate model revealed that if we relax the robustness of our model management strategy we can improve the time saving from $\sim$25% to $\sim$41.5% for the BreakHis datasets, while still finding high-performing individuals.

7 Conclusions

In this thesis, we have developed a semantic-based surrogate-assisted neuroevolution approach for evolving DNN architectures. Neuroevolution is an incredibly computationally expensive process, where thousands of network evaluations are often required, taking many GPU days and weeks to complete. Surrogate-Assisted Evolutionary Algorithms (SAEAs) are a viable and effective way of considerably reducing this cost. In particular, we have focused on a semantic-based approach which has many properties that make it suitable for the task of surrogate modelling. For example, since semantics is based on the behaviour of the neural network architecture rather than its structure and since it also produces fixed-size vectors, it allows us to naturally compare architectures of different topologies rather than using encodings of the network architecture. Throughout this thesis, we have provided an in-depth analysis of the robustness of our surrogate modelling technique while also detailing its ability to effectively estimate the fitness of DNN architectures. In this chapter, we will focus on summarising the original contributions of this thesis, offer conclusions on the use of SAEAs in neuroevolution for NAS in DNNs, and discuss the limitations and future work.

7.1 Original Contributions of this Thesis

In this thesis, we developed a method to incorporate semantics into a SAEA in neuroevolution for neural architecture search (NAS). To achieve this we have:

- Demonstrated the robustness of considering semantic-based distance metrics for promoting diversity in GP, using multiple MO frameworks as a test case. This work ultimately served as a foundation for considering semantics in surrogate modelling.
- Pointed out how traditional surrogate-based optimisation, such as Kriging, is unsuitable in neuroevolution for NAS in DNNs due to the high-dimensional data required for this task.

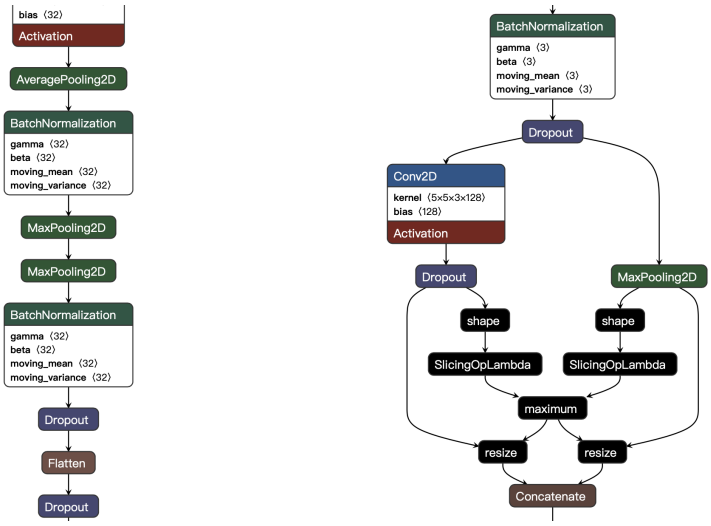


Figure 7.1: Comparison of the original NeuroLGP architecture (left, detailed in Chapter 5) and NeuroLGP-MB architecture (right, detailed in Chapter 6).

- Incorporated more ad-hoc surrogate-based optimisation techniques that graciously manage high-dimensional data. Specifically, we incorporated Kriging Partial Least Squares (KPLS) [31] in surrogate-assisted neuroevolution for NAS in DNNs.

- Using a suitable approach, named Neuro-Linear Genetic Programming (NeuroLGP) that allows us to achieve two important aspects: (i) a representation that allows us to carry out some analysis such as the evolution of building blocks in DNNs, and (ii) the use of semantics (or behaviour of programs as referred in genetic programming). To the best of our knowledge, this is the first time LGP has been used to evolve neural network architectures.

- Introduced NeuroLGP Multi-Branch (NeuroLGP-MB) that scaled the topological complexity of the original NeuroLGP approach, to be in line with more modern architectures, by encoding multi-branch and skip connections into the genotype. Fig. 7.1 shows fragments of the types of architectures capable of being produced by the original NeuroLGP left) and NeuroLGP-MB (right).

- We used three metrics to analyse the robustness of our surrogate model along with a visual analysis of performance, detailing the high correlation between surrogate estimated individuals and their corresponding true fitness.

- We demonstrated a significant time and energy reduction of the surrogate model over an expensive model that required all networks to be fully evaluated. Furthermore, we demonstrated that when using a pre-selection method for the surrogate model,

we not only maintained a competitive time reduction but increased the average accuracy over the expensive model.

- We demonstrated that the method is competitive with state-of-the-art hand-crafted architectures when tested on a number of different diagnostic image datasets.

7.2 Conclusions on the use of SAEAs in Neuroevolution for NAS in DNNs

In this work, we focus specifically on Kriging, which is one of the most popular and widely used SAEAs and relies on spatial interpolation to estimate the unsampled regions of the search space. However, there are considerable caveats to using traditional Kriging for neuroevolution: (i) Kriging is known to struggle with high-dimensional data due to the intense computational cost arising from the numerous matrix inversions that are required to adequately estimate parameters. (ii) Kriging requires fixed-size vectors for comparison. If using genotypic information then this can be problematic as some encoding of the network architecture is required. For instance, metrics such as graph-edit distance can be used but are inefficient for large graphs [24, 195]. (iii) Surrogate models require robust model management strategies to ensure that, not only is the training data for the surrogate model adequately correlated with the real data, but also that there are enough training samples to inform the surrogate model.

In this thesis, we have made an attempt to overcome some of these issues. Specifically, we have done the following:

- To begin with, we started our research by using NeuroLGP. This approach, fully detailed in Chapter 5, allows us to codify the building blocks of a deep neural network. It also allowed us to incorporate semantics, in the same manner as adopted by the genetic programming community when dealing with indirect semantics. This is particularly important because the semantics of individuals representing DNNs are used for the KPLS, which is suitable for high-dimensional data necessary to deal with truly large DNNs. We have discovered that semantic outputs are suitable for KPLS, allowing for an accurate estimate of the performance of our DNNs architectures. We concluded that incorporating semantics into neuroevolution for NAS in SAEAs is feasible by leveraging relatively large vector-stored semantic information, similar to how indirect semantics are used in genetic programming.

- Owing to the use of semantics, we used a fixed-size vector representation based on phenotypic information. This, in turn, allows us to naturally compare architectures which are variable-length in nature and which have graph-based topologies, without

the need for specialised encoding, as discussed in Chapters 5 and 6, respectively. This representation enables its application to state-of-the-art surrogate models while searching for truly deep and modern neural network architectures.

- As already mentioned, we performed an in-depth analysis of the surrogate model robustness using three metrics, but further to that, we analysed the effect of changing the initial epoch number, which controls when our semantic information is used to inform the surrogate model. Interestingly, we found that reducing epoch numbers led to a substantial reduction in time, while still maintaining high-performing individuals, though this came at the expense of the robustness of the model. We can conclude that while balancing robustness and time saved in our model is a key consideration, our approach is nevertheless able to find high-performing individuals in many settings.

- Additionally, our surrogate approach requires a relatively low number of full evaluations. Often thousands of evaluations are required to find high-performing architectures with neuroevolution. This is an important consideration as it can further help reduce the cost of performing neuroevolution for NAS.

7.3 Limitations and Future Work

- It has been noted that other NAS approaches often will perform analysis using a single run [37], which in turn can lead to issues in terms of reproducability [85]. This limitation is a result of the inherent computational cost of NAS as a whole, for example, across Chapters 5 and 6 we amassed over ∼100 GPU days for our experimentation alone, not counting the significant development and testing time that was required to ensure the validity of our approach. Despite this, we strove to perform as many runs as possible under a modest budget and offered a limited statistical analysis in Chapter 5.5.3.

- Future work will look at scaling dataset size to benchmark problems like CIFAR-10 [83] and ImageNet [84], where there are 10,000s of data instances and have a larger number of classes. A challenge remains with considering phenotypic distance vector sizes as the size increases linearly relative to the number of classes, so this would need to be taken into consideration.

  While it would be beneficial to do a comparison with other NAS approaches, given the expensive nature of the experimentation, this was outside the scope of this thesis. Regardless, the surrogate model management strategy should be applicable

to other NAS approaches and future research could investigate its potential benefits in this context.

- The approach outlined in this thesis focused mainly on CNN architectures, however, the surrogate model management strategy is applicable to other types of DNN architectures as long the output vector size remains fixed, for example, AutoEncoder [49] or Transformer [70] architectures could be investigated in the future. Furthermore, other aspects of the architecture could be explored in terms of their behavioural output, for instance, architectures that use embedding spaces.

- As discussed in Chapter 5, there are aspects of LGP that were not fully explored within this thesis but could offer future insight. For instance, the LGP genome contains both effective and non-effective portions of code within the genome. This would be interesting to study from the perspective of neutrality, an area which has yet to be explored in neuroevolution [37, 127]. Additionally, LGP allows for control branches, which could allow for more complex design rules to be incorporated, which may be beneficial for exploring Mixture-of-Experts based architectures that use conditional computation [192].

- To date, surrogate-assisted neuroevolution has mainly focused on single-objective optimisation. Future work could consider MO optimisation. It would be interesting to explore how our proposed approaches behave under both Pareto dominance-based approaches such as NSGA-II and the like as well as decomposition-based methods such as MOEA/D. It is also possible to consider MO within the surrogate framework, such as using MO for in-fill sampling using uncertainty and fitness as objectives [119].

- By demonstrating the effectiveness of semantic-based surrogate-assisted neuroevolution of DNNs, we could now consider using semantics with the NeuroLGP approach in other ways, for example, in promoting diversity, which has yielded better results in GP compared to when it is absent during evolutionary search.

A

Appendix A

A.1 Additional Tables Relating to Chapter 4

Tables A.2 and A.3 report, for each problem defined in Table 4.2 both, the average hypervolume over 50 runs and also the hypervolume of the accumulated PO front with respect to all 50 runs. To obtain a statistically sound conclusion, a series of Wilcoxon rank-sum tests were run on the average hypervolume results. To account for the problem of multiple comparisons that arose from testing the canonical method 16 times for each data set, a Bonferroni correction $\frac{\alpha}{m} = 3.125 \times 10^{-3}$ was used where $\alpha = 0.05$. These statistically significant differences are highlighted in boldface in Table A.2 (this shown in the appendix). Moreover, in this table, the symbols "+" and "–", indicate that the results of a given semantic-based approach are significantly better or worse, respectively, than those found by the canonical NSGA-II (Table A.1), all the above based on the Wilcoxon rank-sum test. For ease of comparison, Table A.1 has been reproduced in the appendix but was originally included in Chapter 4.3.1.

A.2 Additional Comparison of Pay-off Tables for Chapter 4

In Chapter 4.3.1 we provide a detailed comparison against the canonical approaches of NSGA-II and SPEA-II compared against the three semantics approaches, SDO, PSDO and SSC (shown in Tables 4.7 and 4.8). We have included here an additional comparison between the semantic approaches.

If we consider just SDO *vs.* PSDO, results tend to be mixed. NSGA-II PSDO produced more wins for certain data sets like Yeast_2 and Abal_2 when compared with NSGA-II SDO but under-performed for Spect and Abal_1. Typically when NSGA-II PSDO outperformed against NSGA-II SDO, it was for a specific LBSS or UBSS setting. For instance, three of the wins associated with Spect were a result of keeping UBSS constant at 1.0 with LBSS values of (0.001, 0.01, 0.1). NSGA-II PSDO performed significantly worse most often when LBSS was undefined except for Abal_1 where the results were reversed,

Table A.1: Average hypervolume (± std. deviation) and last run Pareto Front for NSGA-II and SPEA2 for 50 independent runs.

Dataset	NSGA-II Hypervolume		SPEA2 Hypervolume	
	Average	PO Front	Average	PO Front
Ion	0.766 ± 0.114	0.938	0.786 ± 0.094	0.948
Spect	0.534 ± 0.024	0.647	0.544 ± 0.032	0.659
$Yeast_1$	0.838 ± 0.011	0.876	0.838 ± 0.008	0.877
$Yeast_2$	0.950 ± 0.009	0.976	0.946 ± 0.015	0.978
$Abal_1$	0.847 ± 0.058	0.961	0.832 ± 0.078	0.960
$Abal_2$	0.576 ± 0.122	0.842	0.544 ± 0.147	0.834

i.e when LBSS was undefined PSDO performed as good or significantly better but was significantly worse for 10 of the other LBSS settings.

We now turn our attention to SPEA2 and its variants. When considering Table 4.8, there was little or no significant difference when comparing SPEA2 SDO and SPEA2 PSDO strategies with only SPEA2 SDO *vs.* SPEA2 PSDO in $Yeast_1$ producing 1 'Win'. From this we can conclude that both methods, SDO and PSDO, that treat semantic distance as an additional criterion to be optimised perform similar and yield consistently better results to SSC. This latter method is based on the notion of single-objective GP adapted to MOGP, showing that the benefits observed in SOGP are depleted in MOGP.

A.3 Additional Images Relating to Chapter 4

This appendix contains additional images for relating to Chapter 4, which covers Semantic Neighbourhood Ordering in MOEA/D. For ease of readability canonical images have been arranged on the left-hand side and semantic ordering images have been on the right-hand side.

Table A.2: Average hypervolume (± std. deviation) and last run Pareto Front for NSGA-II SDO, NSGA-II PSDO and NSGA-II SSC methods.

		Hypervolume							
		Average UBSS				PO Front UBSS			
	LBSS	0.25	0.5	0.75	1.0	0.25	0.5	0.75	1.0
NSGA-II SDO									
Ion	–	**0.860 ± 0.033**+	**0.869 ± 0.037**+	**0.869 ± 0.033**+	**0.845 ± 0.057**+	0.948	0.958	0.962	0.950
	0.001	**0.817 ± 0.087**+	**0.819 ± 0.104**+	**0.857 ± 0.057**+	**0.861 ± 0.047**+	0.942	0.957	0.954	0.958
	0.01	**0.825 ± 0.084**+	**0.843 ± 0.073**+	**0.861 ± 0.045**+	**0.861 ± 0.038**+	0.946	0.956	0.957	0.944
	0.1	**0.846 ± 0.070**+	**0.848 ± 0.068**+	**0.844 ± 0.075**+	**0.864 ± 0.044**+	0.950	0.956	0.953	0.960
Spect	–	**0.591 ± 0.027**+	**0.593 ± 0.025**+	**0.594 ± 0.023**+	**0.600 ± 0.019**+	0.684	0.679	0.689	0.694
	0.001	**0.562 ± 0.021**+	**0.558 ± 0.025**+	**0.561 ± 0.019**+	**0.560 ± 0.016**+	0.668	0.653	0.660	0.644
	0.01	**0.564 ± 0.025**+	**0.560 ± 0.023**+	**0.566 ± 0.024**+	**0.559 ± 0.016**+	0.672	0.650	0.669	0.643
	0.1	**0.563 ± 0.022**+	**0.563 ± 0.024**+	**0.567 ± 0.018**+	**0.561 ± 0.024**+	0.664	0.658	0.655	0.658
$Yeast_1$	–	**0.850 ± 0.006**+	**0.849 ± 0.008**+	**0.849 ± 0.006**+	**0.849 ± 0.006**+	0.881	0.881	0.882	0.881
	0.001	**0.845 ± 0.007**+	**0.847 ± 0.006**+	**0.848 ± 0.004**+	**0.848 ± 0.005**+	0.879	0.882	0.879	0.880
	0.01	**0.848 ± 0.006**+	**0.849 ± 0.005**+	**0.848 ± 0.005**+	**0.850 ± 0.005**+	0.881	0.881	0.879	0.881
	0.1	**0.847 ± 0.005**+	**0.848 ± 0.005**+	**0.848 ± 0.005**+	**0.850 ± 0.005**+	0.878	0.879	0.879	0.883
$Yeast_2$	–	**0.961 ± 0.007**+	**0.961 ± 0.007**+	**0.960 ± 0.008**+	**0.962 ± 0.007**+	0.978	0.979	0.979	0.979
	0.001	**0.959 ± 0.008**+	**0.958 ± 0.007**+	**0.961 ± 0.006**+	**0.961 ± 0.006**+	0.981	0.978	0.979	0.978
	0.01	**0.955 ± 0.009**+	**0.959 ± 0.007**+	**0.960 ± 0.009**+	**0.961 ± 0.007**+	0.979	0.980	0.979	0.978
	0.1	**0.958 ± 0.009**+	**0.960 ± 0.007**+	**0.961 ± 0.007**+	**0.962 ± 0.006**+	0.978	0.978	0.981	0.979
$Abal_1$	–	0.849 ± 0.081	0.862 ± 0.087	0.847 ± 0.089	0.849 ± 0.085	0.964	0.970	0.966	0.967
	0.001	**0.892 ± 0.051**+	**0.905 ± 0.036**+	**0.907 ± 0.036**+	**0.906 ± 0.034**+	0.970	0.968	0.969	0.971
	0.01	**0.908 ± 0.038**+	**0.900 ± 0.056**+	**0.919 ± 0.022**+	**0.919 ± 0.026**+	0.969	0.973	0.970	0.972
	0.1	**0.910 ± 0.037**+	**0.911 ± 0.046**+	**0.912 ± 0.049**+	**0.916 ± 0.031**+	0.970	0.972	0.969	0.970
$Abal_2$	–	0.591 ± 0.102	0.623 ± 0.138	0.634 ± 0.115	0.617 ± 0.137	0.862	0.878	0.881	0.873
	0.001	**0.729 ± 0.070**+	**0.722 ± 0.063**+	**0.709 ± 0.080**+	**0.735 ± 0.074**+	0.877	0.870	0.879	0.885
	0.01	**0.721 ± 0.067**+	**0.725 ± 0.075**+	**0.721 ± 0.074**+	**0.723 ± 0.066**+	0.881	0.879	0.884	0.880
	0.1	**0.724 ± 0.076**+	**0.739 ± 0.065**+	**0.736 ± 0.063**+	**0.756 ± 0.065**+	0.888	0.883	0.886	0.890
Better (+) / Worse (-)		22 / 0	22 / 0	22 / 0	22 / 0				
Same (=) / NSS		0 / 2	0 / 2	0 / 2	0 / 2				
NSGA-II PSDO									
Ion	–	0.794 ± 0.100	**0.811 ± 0.084** +	0.823 ± 0.091	0.795 ± 0.105	0.904	0.932	0.945	0.939
	0.001	**0.867 ± 0.035**+	**0.874 ± 0.029**+	**0.880 ± 0.036**+	**0.873 ± 0.045**+	0.959	0.952	0.965	0.945
	0.01	**0.852 ± 0.050**+	**0.867 ± 0.051**+	**0.880 ± 0.031**+	**0.867 ± 0.050**+	0.947	0.950	0.944	0.949
	0.1	**0.853 ± 0.062**+	**0.869 ± 0.048**+	**0.875 ± 0.051**+	**0.872 ± 0.049**+	0.941	0.951	0.956	0.938
Spect	–	**0.552 ± 0.020**+	**0.546 ± 0.022**+	**0.555 ± 0.022**+	**0.554 ± 0.017**+	0.648	0.665	0.638	0.640
	0.001	**0.550 ± 0.026**+	**0.562 ± 0.025**+	**0.561 ± 0.025**+	**0.592 ± 0.026**+	0.661	0.670	0.658	0.706
	0.01	**0.550 ± 0.025**+	**0.563 ± 0.026**+	**0.558 ± 0.025**+	**0.583 ± 0.020**+	0.649	0.675	0.667	0.669
	0.1	**0.551 ± 0.023**+	**0.560 ± 0.024**+	**0.557 ± 0.025**+	**0.593 ± 0.020**+	0.666	0.664	0.678	0.682
$Yeast_1$	–	**0.846 ± 0.006**+	**0.846 ± 0.005**+	**0.847 ± 0.005**+	**0.848 ± 0.006**+	0.864	0.868	0.871	0.869
	0.001	**0.849 ± 0.006**+	**0.848 ± 0.005**+	**0.850 ± 0.007**+	**0.850 ± 0.005**+	0.873	0.868	0.871	0.869
	0.01	**0.850 ± 0.005**+	**0.849 ± 0.007**+	**0.850 ± 0.006**+	**0.851 ± 0.006**+	0.870	0.874	0.872	0.872
	0.1	**0.850 ± 0.006**+	**0.850 ± 0.005**+	**0.850 ± 0.005**+	**0.851 ± 0.006**+	0.876	0.873	0.872	0.870
$Yeast_2$	–	**0.957 ± 0.007**+	**0.959 ± 0.007**+	**0.957 ± 0.009**+	**0.959 ± 0.007**+	0.973	0.978	0.976	0.978
	0.001	**0.960 ± 0.010**+	**0.962 ± 0.005**+	**0.964 ± 0.005**+	**0.962 ± 0.008**+	0.976	0.976	0.978	0.977
	0.01	**0.962 ± 0.006**+	**0.962 ± 0.006**+	**0.962 ± 0.005**+	**0.962 ± 0.006**+	0.977	0.975	0.974	0.975
	0.1	**0.964 ± 0.006**+	**0.960 ± 0.010**+	**0.963 ± 0.005**+	**0.961 ± 0.007**+	0.976	0.976	0.977	0.975
$Abal_1$	–	**0.890 ± 0.051**+	**0.881 ± 0.070**+	**0.885 ± 0.046**+	**0.884 ± 0.058**+	0.959	0.966	0.961	0.952
	0.001	0.861 ± 0.079	0.848 ± 0.073	**0.877 ± 0.078**+	0.864 ± 0.075	0.962	0.957	0.962	0.959
	0.01	0.864 ± 0.067	0.858 ± 0.076	0.865 ± 0.070	0.873 ± 0.066	0.967	0.959	0.962	0.962
	0.1	0.858 ± 0.082	**0.887 ± 0.061**+	0.864 ± 0.074	0.860 ± 0.075	0.963	0.968	0.962	0.955
$Abal_2$	–	**0.704 ± 0.083**+	**0.699 ± 0.072**+	**0.706 ± 0.069**+	**0.711 ± 0.076**+	0.826	0.859	0.874	0.858
	0.001	**0.725 ± 0.070**+	**0.743 ± 0.079**+	**0.745 ± 0.060**+	**0.733 ± 0.075**+	0.859	0.871	0.854	0.877
	0.01	**0.741 ± 0.086**+	**0.735 ± 0.074**+	**0.724 ± 0.070**+	**0.728 ± 0.069**+	0.873	0.867	0.870	0.873
	0.1	**0.743 ± 0.061**+	**0.723 ± 0.073**+	**0.719 ± 0.088**+	**0.722 ± 0.063**+	0.877	0.847	0.846	0.851
Better (+) / Worse (-)		20 / 0	22/ 0	21 / 0	20 / 0				
Same (=) / NSS		0 / 4	0 / 2	0 / 3	0 / 4				
NSGA-II SSC									
Ion	–	0.761 ± 0.108	0.749 ± 0.161	0.763 ± 0.152	0.744 ± 0.137	0.941	0.937	0.951	0.949
	0.001	0.765 ± 0.134	0.753 ± 0.124	0.699 ± 0.188	0.803 ± 0.103	0.954	0.935	0.928	0.946
	0.01	0.760 ± 0.125	0.751 ± 0.123	0.710 ± 0.161	0.802 ± 0.104	0.947	0.929	0.928	0.947
	0.1	0.775 ± 0.095	0.738 ± 0.184	0.746 ± 0.141	0.778 ± 0.099	0.957	0.951	0.945	0.936
Spect	–	0.525 ± 0.025	0.532 ± 0.029	0.537 ± 0.020	0.535 ± 0.029	0.633	0.634	0.634	0.634
	0.001	0.530 ± 0.029	0.539 ± 0.030	0.542 ± 0.023	0.540 ± 0.025	0.651	0.635	0.638	0.654
	0.01	0.535 ± 0.029	0.537 ± 0.027	0.541 ± 0.027	0.540 ± 0.028	0.655	0.633	0.658	0.651
	0.1	0.532 ± 0.029	0.531 ± 0.026	0.534 ± 0.027	0.533 ± 0.022	0.632	0.641	0.635	0.635
$Yeast_1$	–	0.819 ± 0.041	0.829 ± 0.023	0.835 ± 0.014	0.834 ± 0.017	0.874	0.875	0.878	0.878
	0.001	0.825 ± 0.031	0.834 ± 0.029	0.834 ± 0.019	0.826 ± 0.039	0.877	0.877	0.877	0.877
	0.01	0.827 ± 0.027	0.835 ± 0.016	0.836 ± 0.019	0.830 ± 0.030	0.874	0.877	0.877	0.879
	0.1	0.831 ± 0.027	0.828 ± 0.034	0.831 ± 0.028	0.835 ± 0.014	0.879	0.876	0.876	0.875
$Yeast_2$	–	0.950 ± 0.013	0.948 ± 0.010	0.945 ± 0.032	0.947 ± 0.009	0.978	0.977	0.978	0.977
	0.001	0.946 ± 0.013	0.944 ± 0.028	0.947 ± 0.013	0.950 ± 0.011	0.976	0.976	0.977	0.979
	0.01	0.947 ± 0.014	0.944 ± 0.024	0.946 ± 0.015	0.949 ± 0.012	0.978	0.978	0.978	0.978
	0.1	0.948 ± 0.014	0.948 ± 0.012	0.946 ± 0.009	0.947 ± 0.016	0.978	0.978	0.977	0.977
$Abal_1$	–	0.844 ± 0.084	0.839 ± 0.083	0.834 ± 0.070	0.824 ± 0.099	0.963	0.967	0.962	0.962
	0.001	0.851 ± 0.062	0.812 ± 0.086	0.845 ± 0.077	0.844 ± 0.079	0.964	0.961	0.959	0.967
	0.01	0.850 ± 0.076	0.833 ± 0.091	0.829 ± 0.096	0.836 ± 0.090	0.972	0.957	0.959	0.963
	0.1	0.869 ± 0.064	0.838 ± 0.083	0.844 ± 0.075	0.834 ± 0.084	0.963	0.965	0.965	0.962
$Abal_2$	–	0.521 ± 0.121	0.532 ± 0.103	0.529 ± 0.128	**0.511 ± 0.118**-	0.810	0.802	0.841	0.801
	0.001	0.561 ± 0.082	0.534 ± 0.102	0.542 ± 0.104	0.502 ± 0.161	0.823	0.865	0.829	0.820
	0.01	0.494 ± 0.147	0.536 ± 0.114	0.533 ± 0.134	0.547 ± 0.123	0.844	0.826	0.841	0.850
	0.1	**0.513 ± 0.132**-	0.549 ± 0.120	0.514 ± 0.112	0.532 ± 0.131	0.806	0.820	0.785	0.831
Better (+) / Worse (-)		0 / 1	0 / 0	0 / 0	0 / 1				
Same (=) / NSS		0 / 23	0/ 24	1 / 23	0 / 23				

Table A.3: Average hypervolume (± std. deviation) and last run Pareto Front for SPEA2 SDO, SPEA2 PSDO and SPEA2 SSC methods.

		Hypervolume							
		Average UBSS				PO Front UBSS			
	LBSS	0.25	0.5	0.75	1.0	0.25	0.5	0.75	1.0
SPEA2 SDO									
Ion	–	**0.859 ± 0.031**+	**0.869 ± 0.029**+	**0.862 ± 0.034**+	**0.865 ± 0.047**+	0.951	0.952	0.950	0.961
	0.001	**0.858 ± 0.041**+	**0.852 ± 0.075**+	**0.870 ± 0.055**+	**0.874 ± 0.055**+	0.946-	0.955	0.952	0.956
	0.01	**0.837 ± 0.097**+	**0.851 ± 0.077**+	**0.875 ± 0.032**+	**0.863 ± 0.049**+	0.956	0.951	0.953	0.959
	0.1	**0.852 ± 0.071**+	**0.856 ± 0.053**+	**0.873 ± 0.035**+	**0.862 ± 0.038**+	0.947	0.949	0.952	0.950
Spect	–	**0.591 ± 0.020**+	**0.599 ± 0.021**+	**0.597 ± 0.018**+	**0.595 ± 0.022**+	0.678	0.688	0.686	0.695
	0.001	**0.569 ± 0.021**+	**0.565 ± 0.024**+	**0.566 ± 0.023**+	**0.563 ± 0.023**+	0.668	0.666	0.672	0.658
	0.01	**0.568 ± 0.023**+	**0.567 ± 0.024**+	**0.564 ± 0.025**+	**0.563 ± 0.023**+	0.666	0.674	0.664	0.658
	0.1	**0.566 ± 0.023**+	0.560 ± 0.020	**0.567 ± 0.027**+	0.561 ± 0.022	0.666	0.654	0.673	0.658
$Yeast_1$	–	**0.850 ± 0.007**+	**0.850 ± 0.006**+	**0.849 ± 0.008**+	**0.849 ± 0.004**+	0.882	0.881	0.881	0.881
	0.001	**0.848 ± 0.006**+	**0.847 ± 0.007**+	**0.848 ± 0.004**+	**0.850 ± 0.006**+	0.880	0.883	0.880	0.883
	0.01	**0.848 ± 0.006**+	**0.847 ± 0.006**+	**0.850 ± 0.005**+	**0.850 ± 0.005**+	0.881	0.880	0.882	0.879
	0.1	**0.847 ± 0.005**+	**0.849 ± 0.006**+	**0.848 ± 0.005**+	**0.849 ± 0.006**+	0.879	0.882	0.880	0.882
$Yeast_2$	–	**0.962 ± 0.007**+	**0.962 ± 0.006**+	**0.962 ± 0.006**+	**0.963 ± 0.008**+	0.979	0.979	0.979	0.977
	0.001	**0.958 ± 0.008**+	**0.960 ± 0.007**+	**0.960 ± 0.005**+	**0.960 ± 0.005**+	0.980	0.979	0.979	0.977
	0.01	**0.959 ± 0.008**+	**0.961 ± 0.007**+	**0.961 ± 0.005**+	**0.962 ± 0.007**+	0.979	0.980	0.978	0.978
	0.1	**0.961 ± 0.007**+	**0.961 ± 0.007**+	**0.960 ± 0.007**+	**0.964 ± 0.007**+	0.980	0.979	0.979	0.980
$Abal_1$	–	**0.875 ± 0.059**+	**0.868 ± 0.081**+	**0.875 ± 0.059**+	**0.873 ± 0.069**+	0.965	0.974	0.968	0.972
	0.001	**0.895 ± 0.061**+	**0.911 ± 0.031**+	**0.905 ± 0.044**+	**0.903 ± 0.036**+	0.974	0.973	0.972	0.972
	0.01	**0.903 ± 0.038**+	**0.906 ± 0.042**+	**0.901 ± 0.048**+	**0.910 ± 0.039**+	0.966	0.969	0.972	0.974
	0.1	**0.888 ± 0.067**+	**0.918 ± 0.032**+	**0.910 ± 0.046**+	**0.916 ± 0.027**+	0.974	0.970	0.968	0.967
$Abal_2$	–	0.620 ± 0.148	**0.633 ± 0.124**+	**0.651 ± 0.146**+	0.630 ± 0.138	0.874	0.861	0.879	0.876
	0.001	**0.717 ± 0.069**+	**0.709 ± 0.079**+	**0.722 ± 0.083**+	**0.733 ± 0.075**+	0.868	0.883	0.886	0.891
	0.01	**0.706 ± 0.084**+	**0.720 ± 0.067**+	**0.726 ± 0.067**+	**0.747 ± 0.070**+	0.884	0.880	0.877	0.887
	0.1	**0.732 ± 0.064**+	**0.733 ± 0.066**+	**0.749 ± 0.063**+	**0.737 ± 0.081**+	0.880	0.876	0.883	0.877
Better (+) / Worse (-)		23 / 0	23 / 0	24 / 0	22 / 0				
Eq. (≡) / NSS		0 / 1	0/ 1	0 / 0	0 / 2				
SPEA2 PSDO									
Ion	–	**0.864 ± 0.038**+	**0.861 ± 0.032**+	**0.865 ± 0.031**+	**0.868 ± 0.034**+	0.926	0.932	0.921	0.922
	0.001	**0.852 ± 0.076**+	**0.857 ± 0.055**+	**0.872 ± 0.033**+	**0.869 ± 0.036**+	0.872	0.895	0.846	0.893
	0.01	**0.880 ± 0.036**+	**0.864 ± 0.041**+	**0.857 ± 0.068**+	**0.871 ± 0.036**+	0.955	0.922	0.944	0.905
	0.1	**0.856 ± 0.056**+	**0.874 ± 0.034**+	**0.866 ± 0.03**+	**0.874 ± 0.034**+	0.931	0.921	0.830	0.900
Spect	–	**0.595 ± 0.028**+	**0.599 ± 0.020**+	**0.595 ± 0.020**+	**0.602 ± 0.025**+	0.669	0.670	0.664	0.657
	0.001	**0.563 ± 0.028**+	**0.568 ± 0.020**+	**0.570 ± 0.022**+	**0.565 ± 0.021**+	0.670	0.621	0.623	0.607
	0.01	**0.569 ± 0.027**+	**0.565 ± 0.021**+	**0.566 ± 0.023**+	**0.562 ± 0.022**+	0.628	0.620	0.599	0.610
	0.1	**0.567 ± 0.025**+	**0.563 ± 0.022**+	**0.564 ± 0.022**+	**0.562 ± 0.022**+	0.632	0.633	0.606	0.620
$Yeast_1$	–	**0.849 ± 0.006**+	**0.849 ± 0.005**+	**0.850 ± 0.006**+	**0.849 ± 0.006**+	0.874	0.875	0.867	0.872
	0.001	**0.847 ± 0.005**+	**0.847 ± 0.007**+	**0.848 ± 0.006**+	**0.847 ± 0.005**+	0.867	0.868	0.865	0.865
	0.01	**0.847 ± 0.005**+	**0.848 ± 0.006**+	**0.849 ± 0.006**+	**0.849 ± 0.006**+	0.865	0.864	0.864	0.865
	0.1	**0.848 ± 0.006**+	**0.849 ± 0.006**+	**0.849 ± 0.007**+	**0.849 ± 0.006**+	0.863	0.866	0.867	0.871
$Yeast_2$	–	**0.961 ± 0.008**+	**0.951 ± 0.007**+	**0.962 ± 0.008**+	**0.963 ± 0.007**+	0.979	0.977	0.978	0.975
	0.001	**0.956 ± 0.010**+	**0.959 ± 0.005**+	**0.958 ± 0.005**+	**0.960 ± 0.008**+	0.977	0.975	0.977	0.975
	0.01	**0.959 ± 0.006**+	**0.960 ± 0.006**+	**0.960 ± 0.005**+	**0.963 ± 0.006**+	0.973	0.976	0.973	0.976
	0.1	**0.960 ± 0.006**+	**0.958 ± 0.010**+	**0.963 ± 0.005**+	**0.964 ± 0.007**+	0.975	0.972	0.975	0.974
$Abal_1$	–	0.858 ± 0.067	0.840 ± 0.097	0.855 ± 0.082	**0.869 ± 0.073**+	0.962	0.954	0.962	0.958
	0.001	**0.888 ± 0.048**+	**0.890 ± 0.059**+	**0.902 ± 0.045**+	**0.905 ± 0.037**+	0.964	0.958	0.958	0.965
	0.01	**0.893 ± 0.049**+	**0.908 ± 0.043**+	**0.913 ± 0.046**+	**0.917 ± 0.026**+	0.957	0.959	0.967	0.949
	0.1	**0.912 ± 0.031**+	**0.904 ± 0.053**+	**0.916 ± 0.033**+	**0.919 ± 0.027**+	0.953	0.963	0.962	0.958
$Abal_2$	–	**0.655 ± 0.104**+	**0.621 ± 0.121**+	**0.662 ± 0.107**+	**0.647 ± 0.117**+	0.863	0.864	0.866	0.862
	0.001	**0.720 ± 0.061**+	**0.717 ± 0.066**	**0.715 ± 0.067**+	**0.726 ± 0.059**+	0.861	0.818	0.840	0.850
	0.01	**0.698 ± 0.080**+	**0.716 ± 0.070**+	**0.723 ± 0.077**+	**0.728 ± 0.091**+	0.862	0.869	0.837	0.839
	0.1	**0.727 ± 0.064**+	**0.723 ± 0.063**+	**0.752 ± 0.061**+	**0.741 ± 0.086**+	0.834	0.791	0.834	0.837
Better (+) / Worse (-)		23 / 0	23 / 0	23 / 0	24 / 0				
Same (=) / NSS		0 / 1	0 / 1	0 / 1	0 / 0				
SPEA2 SSC									
Ion	–	0.724 ± 0.157	0.767 ± 0.081	0.743 ± 0.120	0.764 ± 0.100	0.935	0.924	0.939	0.939
	0.001	0.747 ± 0.173	0.767 ± 0.121	0.755 ± 0.155	0.790 ± 0.101	0.951	0.936	0.934	0.947
	0.01	0.741 ± 0.172	0.765 ± 0.117	0.757 ± 0.147	0.790 ± 0.101	0.955	0.942	0.940	0.947
	0.1	0.787 ± 0.106	0.782 ± 0.108	0.778 ± 0.124	0.787 ± 0.119	0.939	0.942	0.956	0.961
Spect	–	0.521 ± 0.045	0.536 ± 0.021	0.543 ± 0.028	0.533 ± 0.027	0.639	0.648	0.657	0.650
	0.001	0.533 ± 0.028	0.536 ± 0.022	0.530 ± 0.035	0.536 ± 0.021	0.644	0.659	0.634	0.634
	0.01	0.530 ± 0.027	0.534 ± 0.029	0.535 ± 0.023	0.538 ± 0.028	0.648	0.644	0.636	0.660
	0.1	0.537 ± 0.021	0.542 ± 0.025	0.536 ± 0.034	0.533 ± 0.028	0.640	0.650	0.641	0.645
$Yeast_1$	–	0.824 ± 0.030	0.824 ± 0.042	0.831 ± 0.020	0.828 ± 0.030	0.877	0.876	0.877	0.874
	0.001	0.826 ± 0.029	0.824 ± 0.062	0.828 ± 0.025	0.833 ± 0.017	0.877	0.875	0.876	0.877
	0.01	0.830 ± 0.020	0.829 ± 0.033	0.832 ± 0.021	0.832 ± 0.020	0.874	0.876	0.876	0.876
	0.1	0.828 ± 0.032	0.836 ± 0.015	0.830 ± 0.028	0.836 ± 0.014	0.875	0.877	0.877	0.876
$Yeast_2$	–	0.950 ± 0.010	0.947 ± 0.011	0.950 ± 0.010	0.951 ± 0.010	0.977	0.976	0.978	0.979
	0.001	0.947 ± 0.015	0.947 ± 0.010	0.948 ± 0.011	0.948 ± 0.010	0.978	0.977	0.978	0.976
	0.01	0.948 ± 0.012	0.948 ± 0.013	0.943 ± 0.022	0.950 ± 0.010	0.978	0.979	0.977	0.978
	0.1	0.944 ± 0.024	0.943 ± 0.017	0.947 ± 0.010	0.945 ± 0.015	0.976	0.977	0.975	0.975
$Abal_1$	–	0.831 ± 0.071	0.856 ± 0.088	0.822 ± 0.080	0.851 ± 0.061	0.960	0.960	0.966	0.961
	0.001	0.812 ± 0.094	0.854 ± 0.082	0.836 ± 0.076	0.847 ± 0.065	0.963	0.965	0.966	0.963
	0.01	0.819 ± 0.098	0.824 ± 0.106	0.841 ± 0.070	0.851 ± 0.063	0.969	0.965	0.964	0.962
	0.1	0.844 ± 0.083	0.833 ± 0.088	0.853 ± 0.095	0.837 ± 0.090	0.965	0.967	0.963	0.965
$Abal_2$	–	0.548 ± 0.120	0.500 ± 0.139	0.515 ± 0.137	0.532 ± 0.107	0.819	0.790	0.815	0.802
	0.001	0.515 ± 0.135	0.518 ± 0.125	0.541 ± 0.111	0.521 ± 0.127	0.812	0.829	0.836	0.807
	0.01	0.537 ± 0.105	0.500 ± 0.163	0.527 ± 0.152	0.521 ± 0.095	0.820	0.817	0.816	0.816
	0.1	0.561 ± 0.111	0.556 ± 0.094	0.516 ± 0.142	0.558 ± 0.098	0.840	0.838	0.838	0.838
Better (+) / Worse (-)		0 / 0	0 / 0	0 / 0	0 / 0				
Eq. (≡) / NSS		0 / 0	0/ 0	0 / 0	0 / 0				

Ion

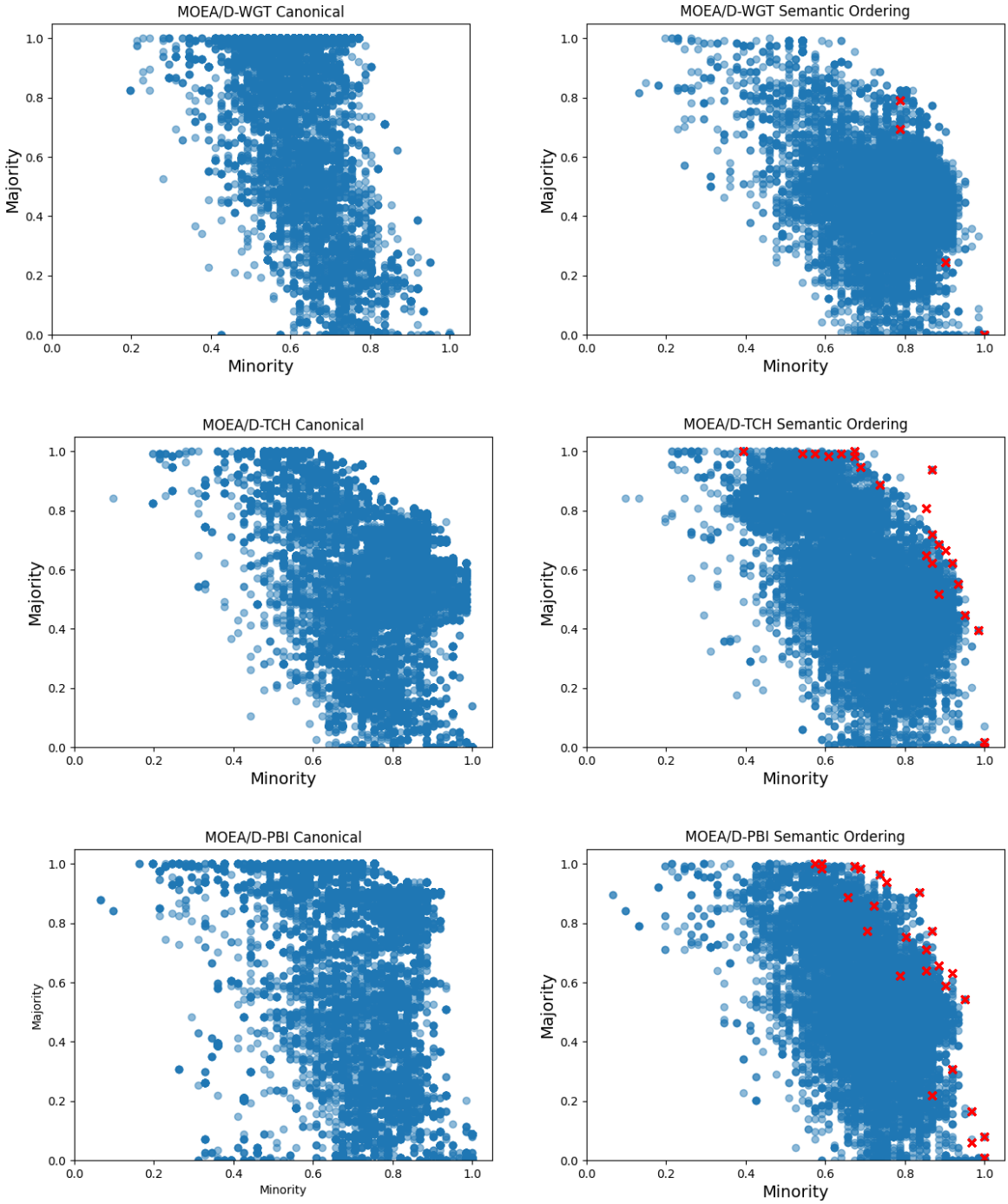


Figure A.1: Solutions for every generation for the Ion dataset for WGT, TCH and PBI approaches. Red cross symbols ('x') denote pivot selections from the external population for all generations for a single run.

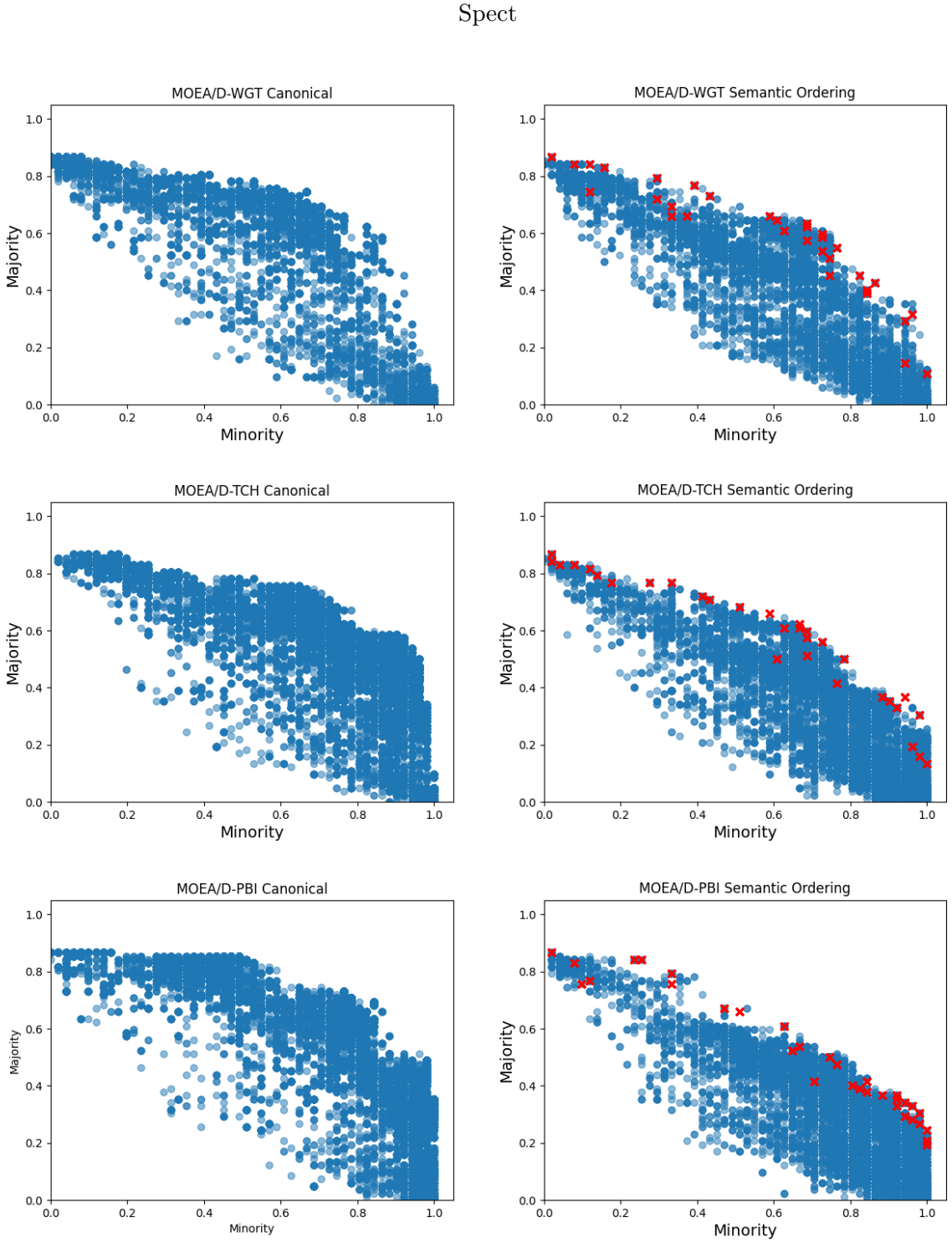


Figure A.2: Solutions for every generation for the Spect dataset for WGT, TCH and PBI approaches. Red cross symbols ('x') denote pivot selections from the external population for all generations for a single run.

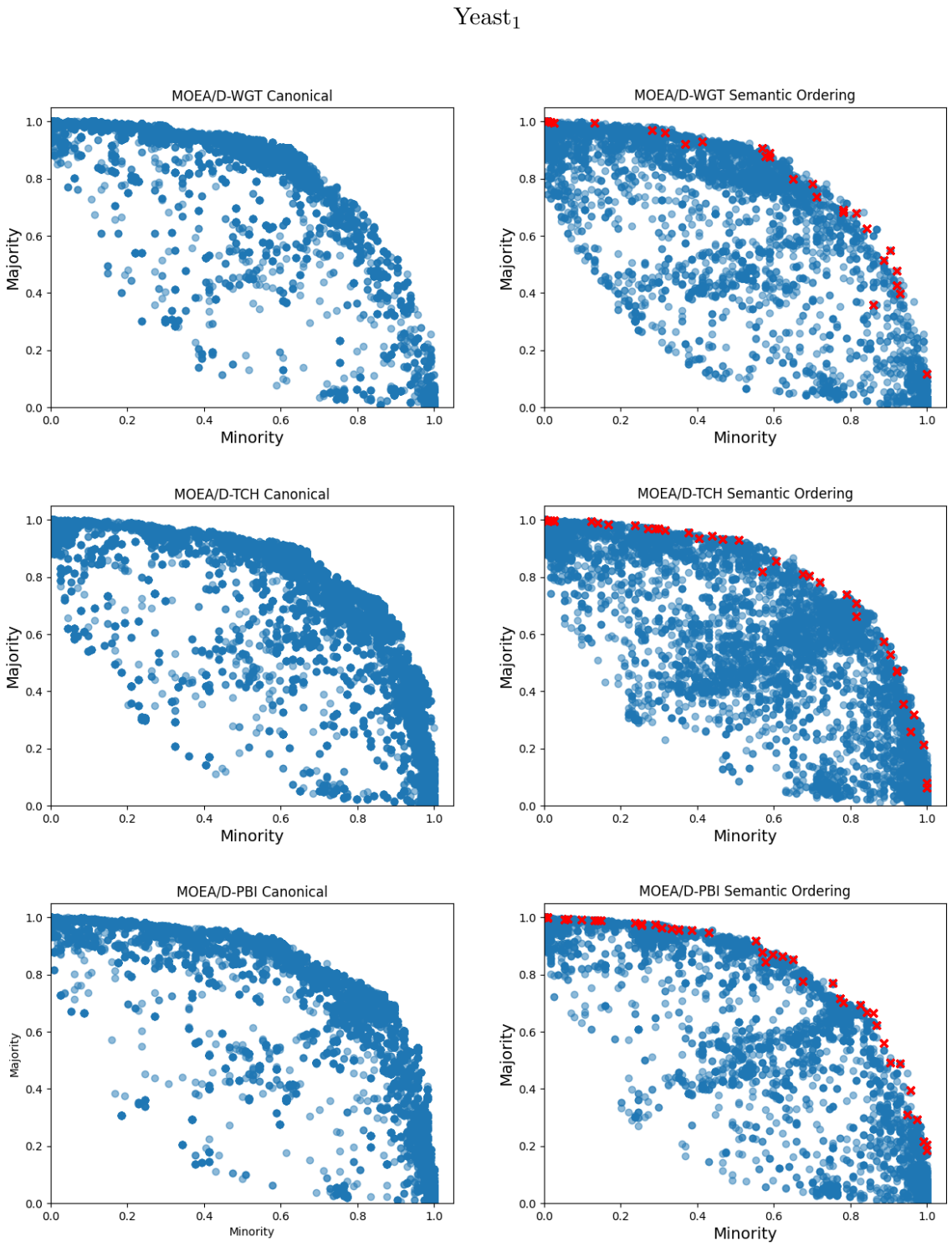


Figure A.3: Solutions for every generation for the Yeast$_1$ dataset for WGT, TCH and PBI approaches. Red cross symbols ('x') denote pivot selections from the external population for all generations for a single run.

Yeast$_2$

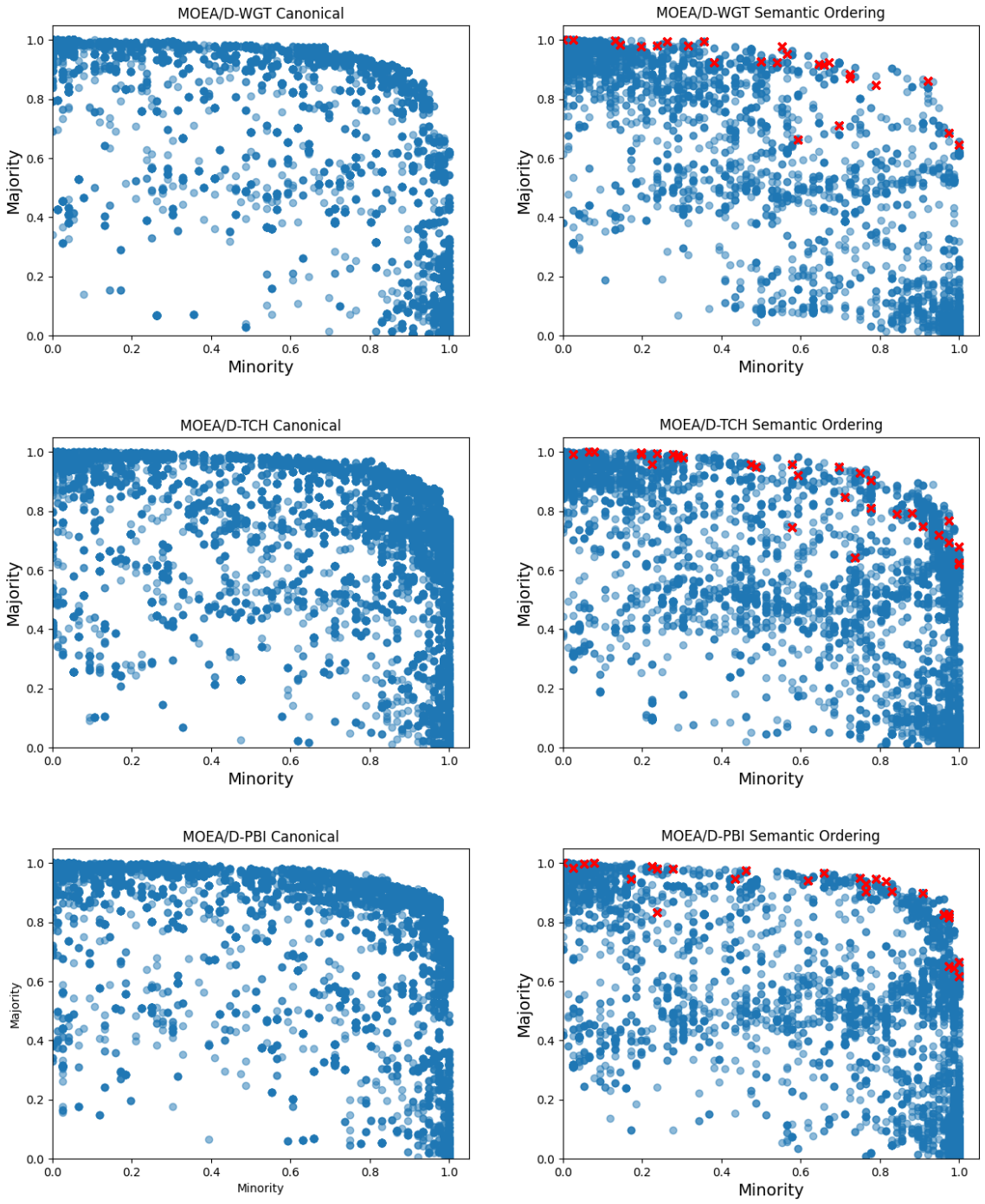


Figure A.4: Solutions for every generation for the Yeast$_2$ dataset for WGT, TCH and PBI approaches. Red cross symbols ('x') denote pivot selections from the external population for all generations for a single run.

Climate

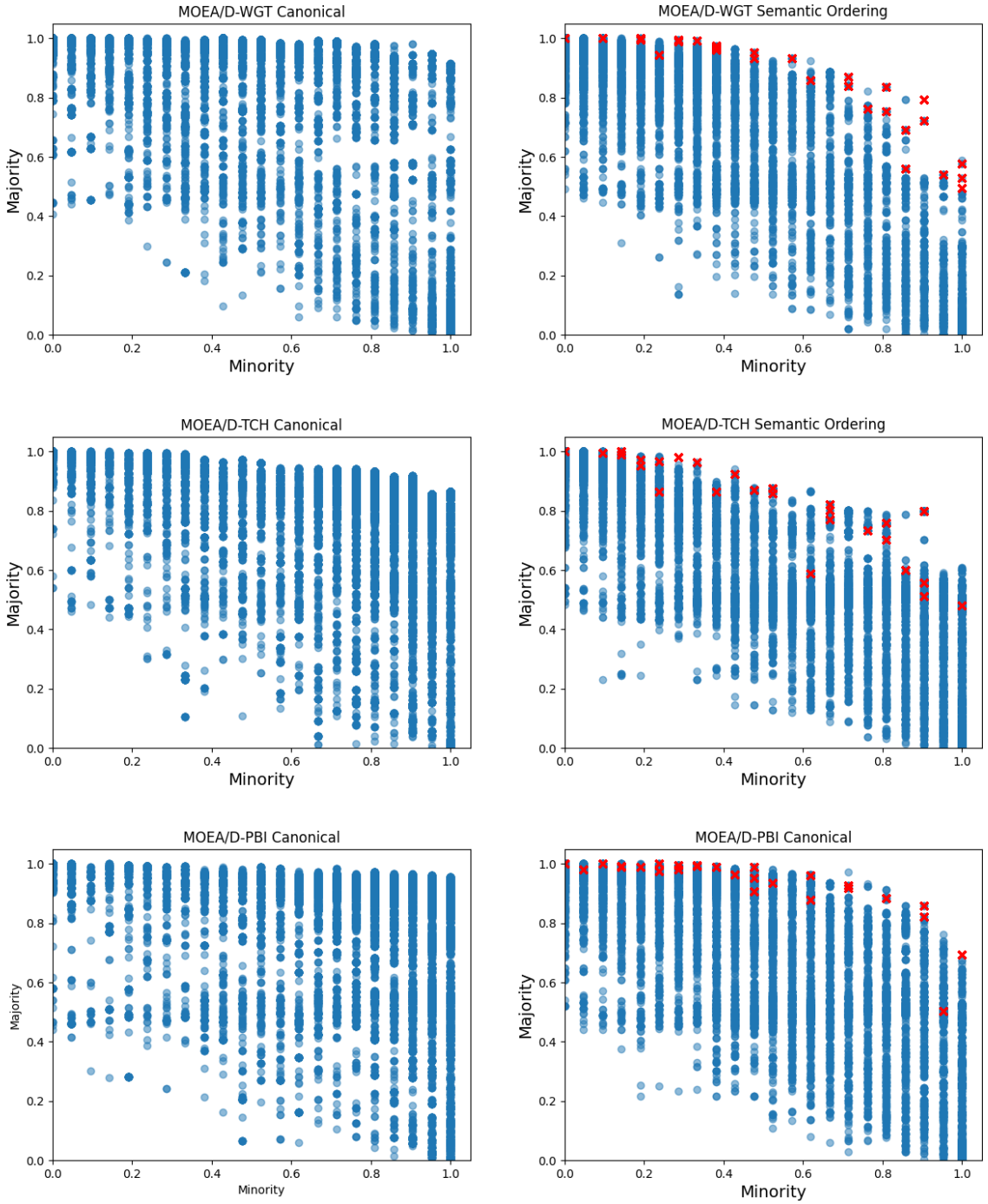


Figure A.5: Solutions for every generation for the Climate dataset for WGT, TCH and PBI approaches. Red cross symbols ('x') denote pivot selections from the external population for all generations for a single run.

Glass$_1$

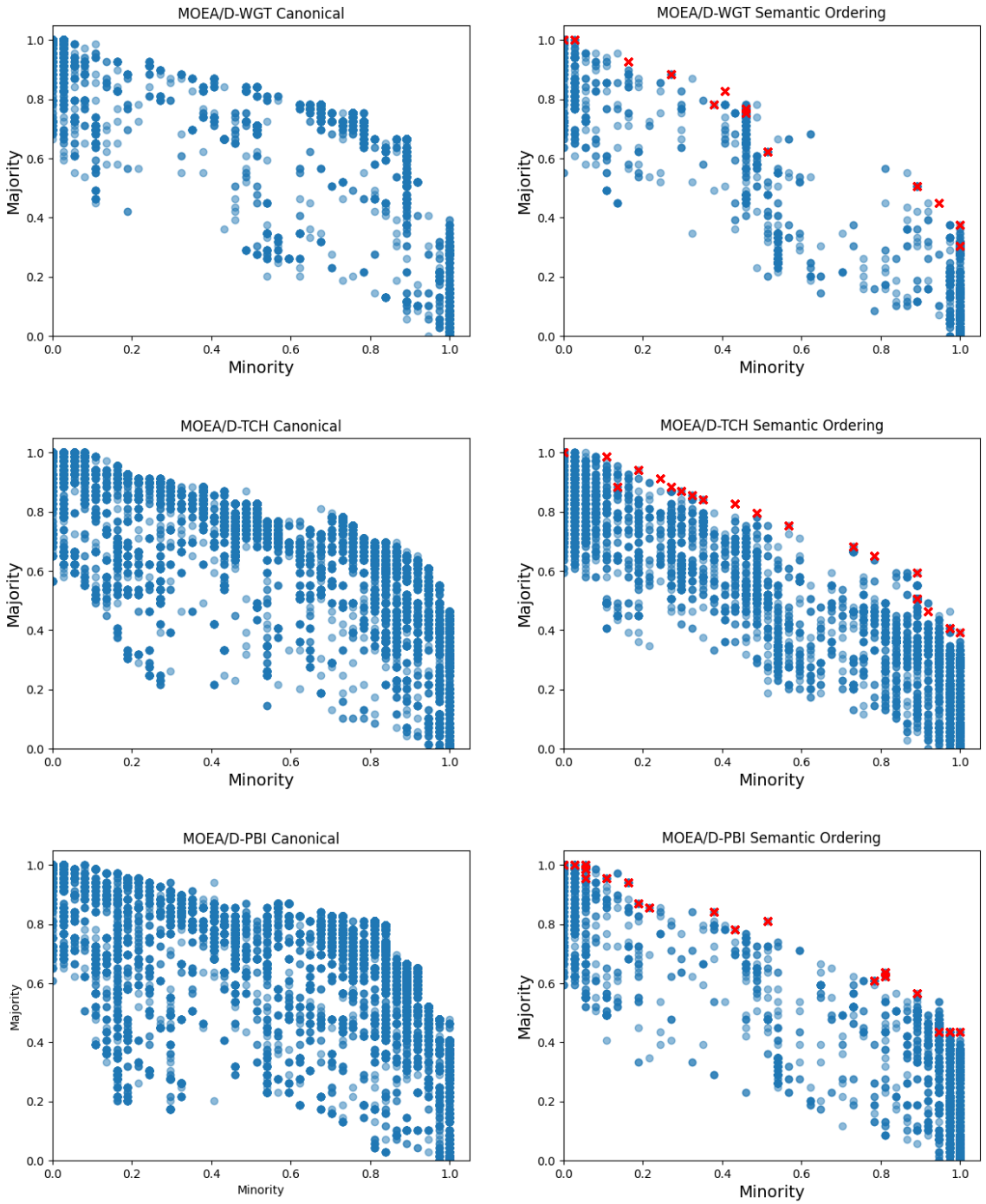


Figure A.6: Solutions for every generation for the Glass dataset for WGT, TCH and PBI approaches. Red cross symbols ('x') denote pivot selections from the external population for all generations for a single run.

Parkinson's

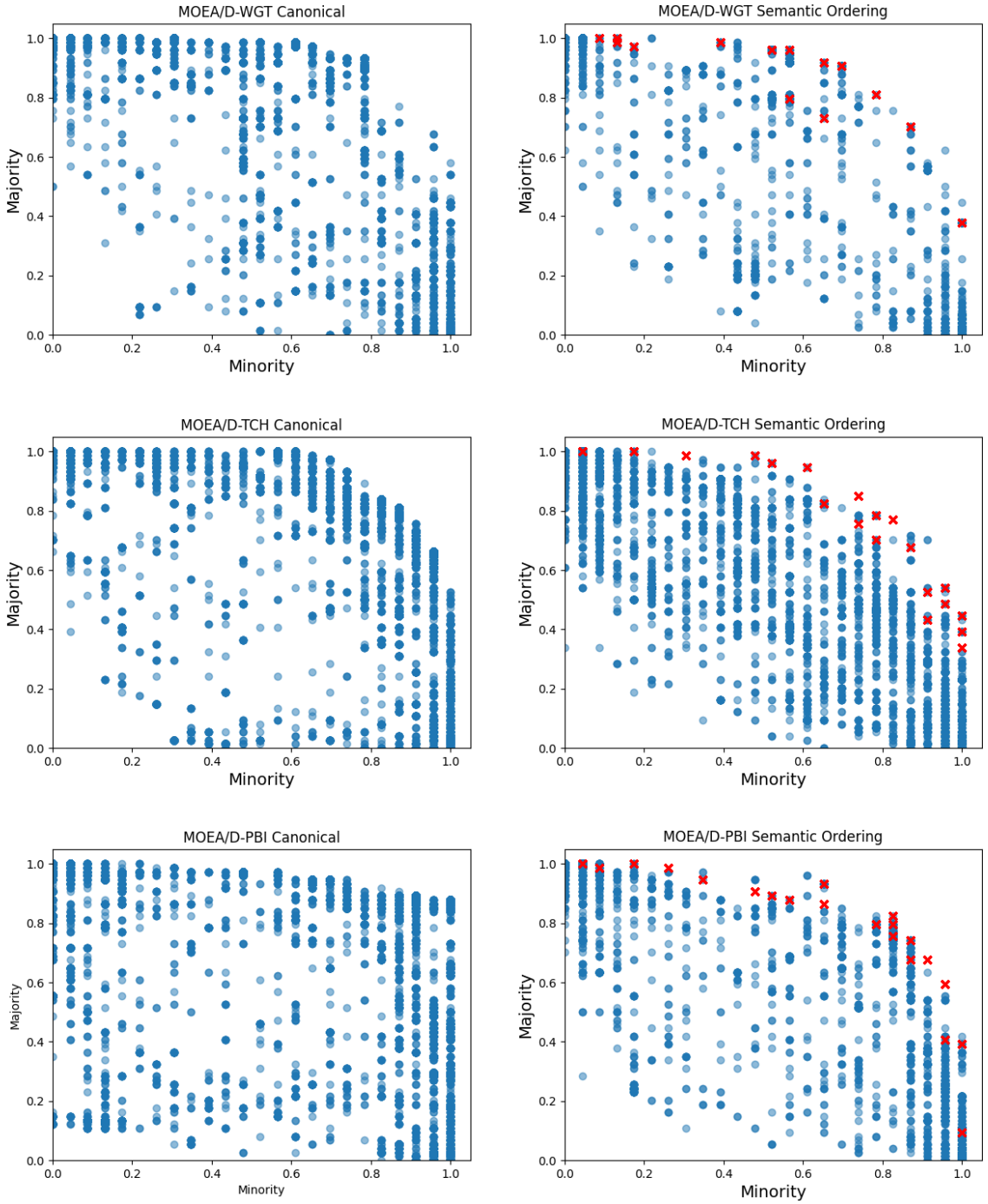


Figure A.7: Solutions for every generation for the Parkinson's dataset for WGT, TCH and PBI approaches. Red cross symbols ('x') denote pivot selections from the external population for all generations for a single run.

Wine

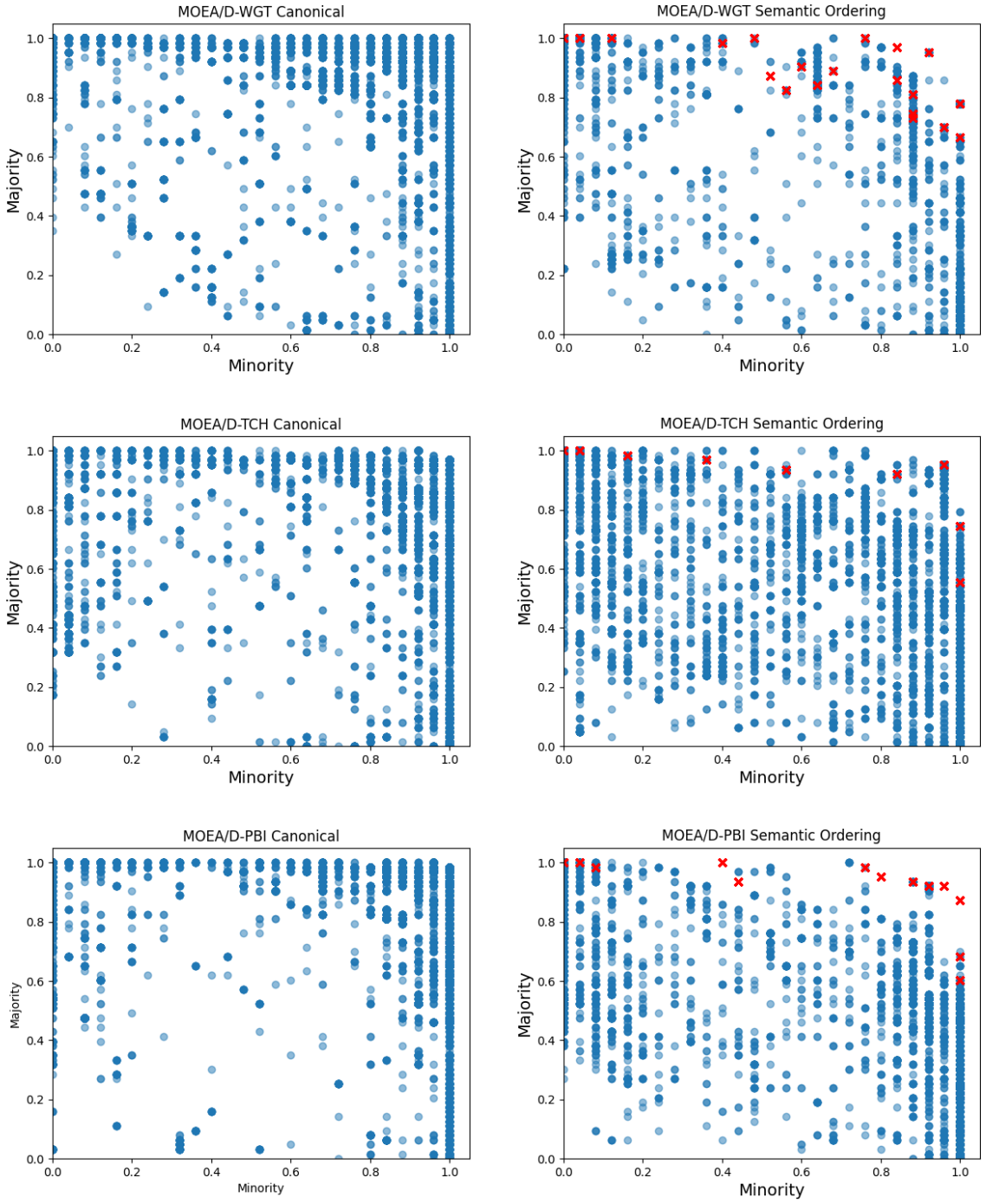


Figure A.8: Solutions for every generation for the Wine dataset for WGT, TCH and PBI approaches. Red cross symbols ('x') denote pivot selections from the external population for all generations for a single run.

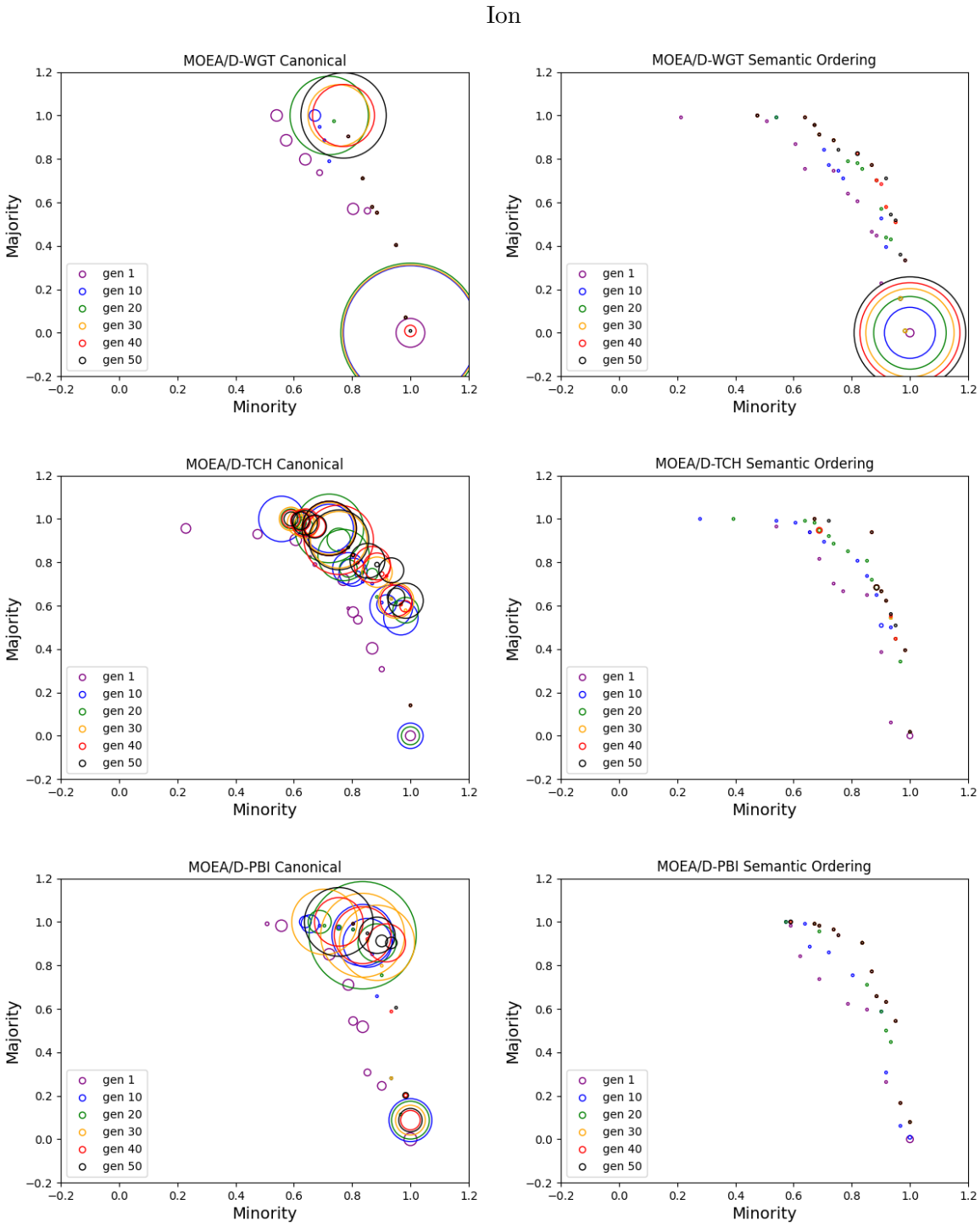


Figure A.9: Duplicate frequency of individuals at first Pareto Front for Ion dataset for WGT, TCH and PBI approaches for generations 1, 10, 20, 30, 40 and 50, for a single run.

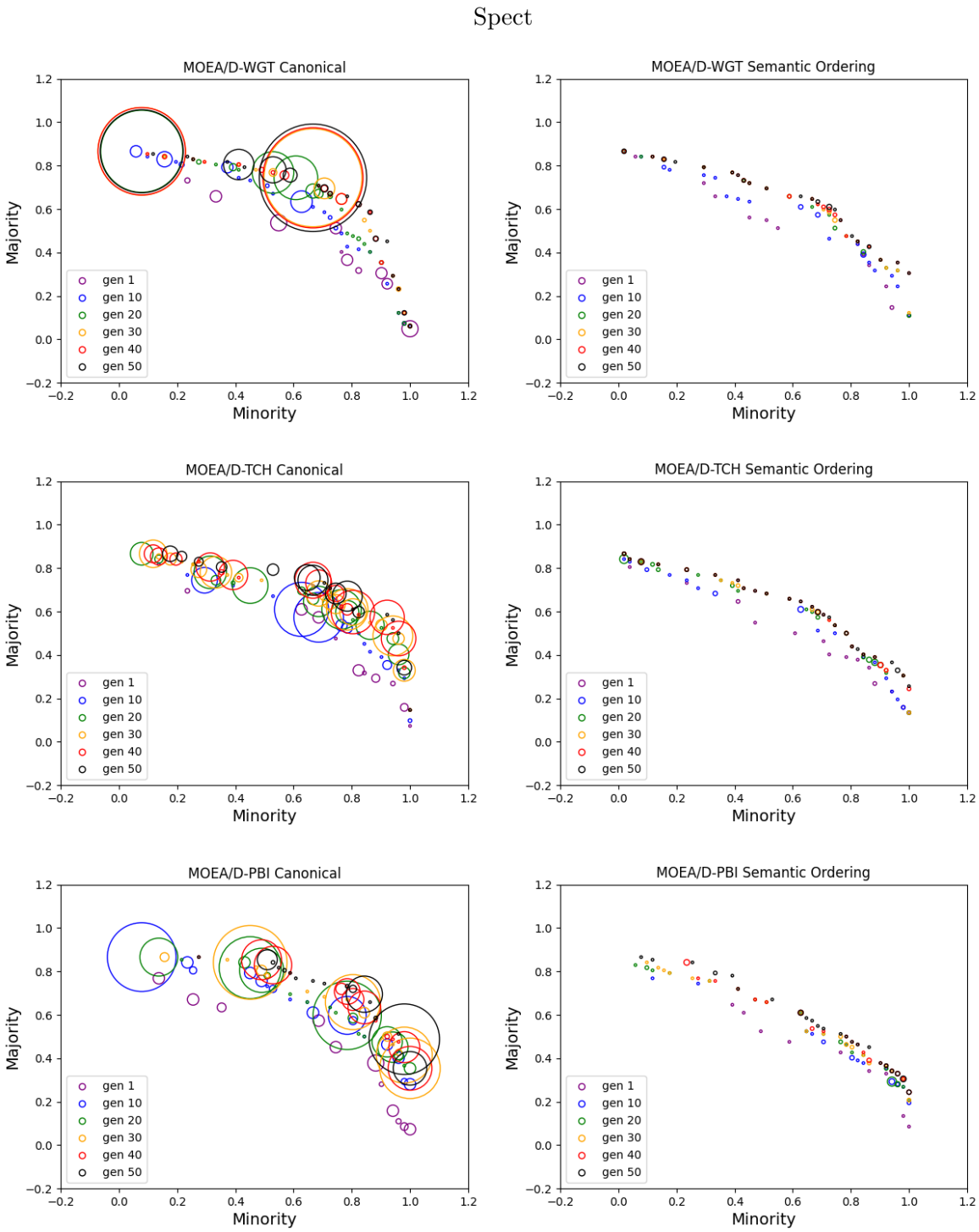


Figure A.10: Duplicate frequency of individuals at first Pareto Front for Spect dataset for WGT, TCH and PBI approaches for generations 1, 10, 20, 30, 40 and 50, for a single run.

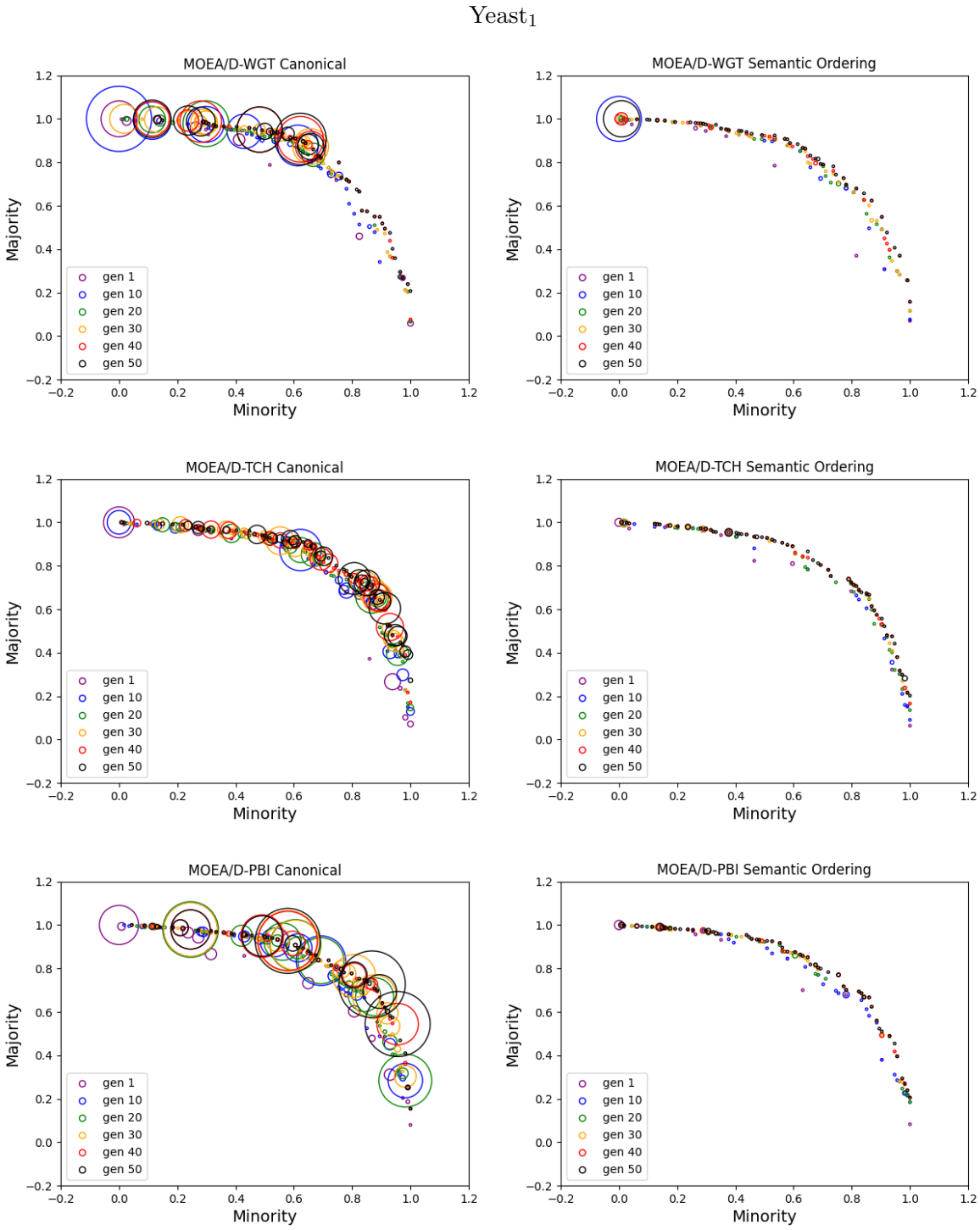


Figure A.11: Duplicate frequency of individuals at first Pareto Front for Yeast$_1$ dataset for WGT, TCH and PBI approaches for generations 1, 10, 20, 30, 40 and 50, for a single run.

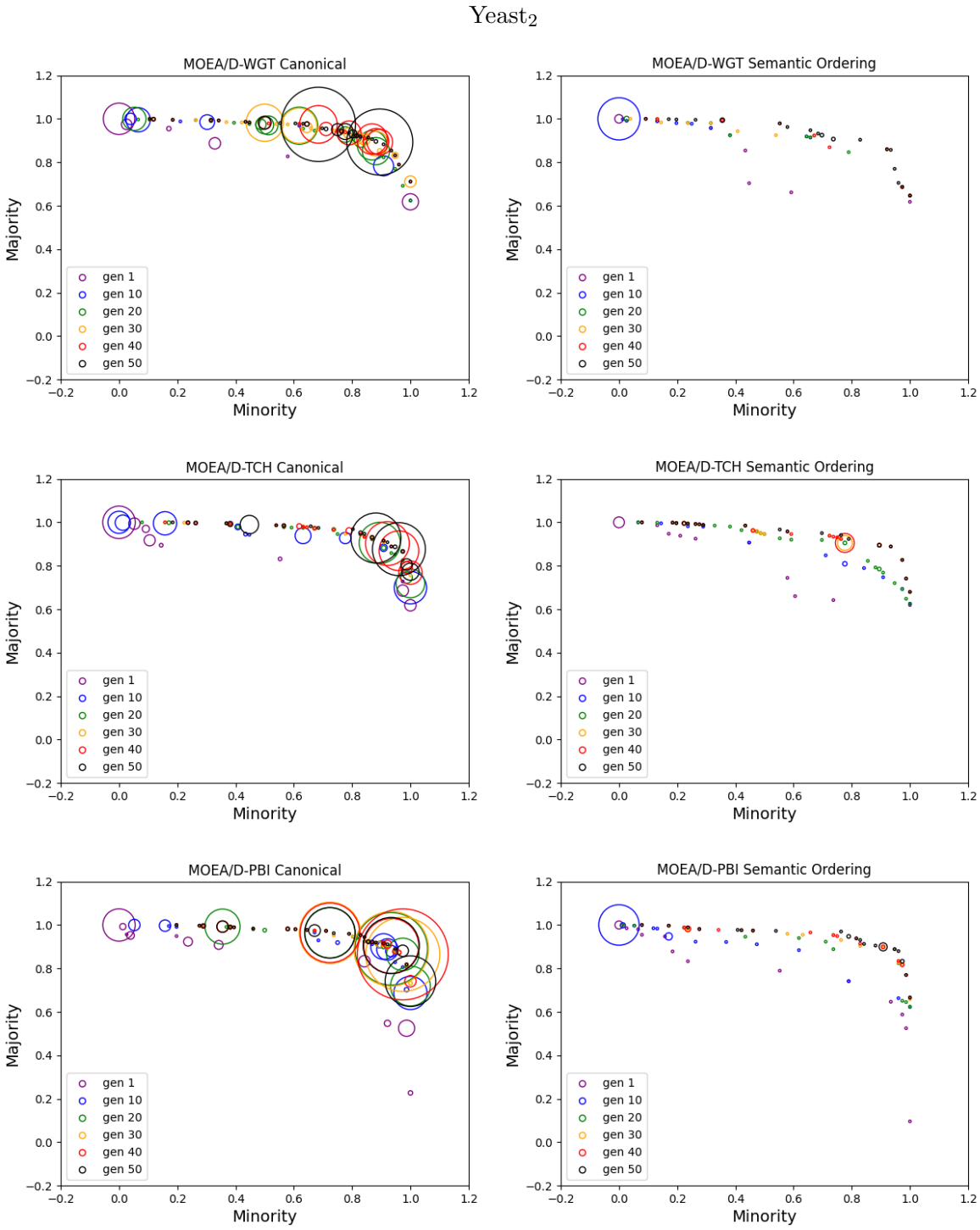


Figure A.12: Duplicate frequency of individuals at first Pareto Front for Yeast$_2$ dataset for WGT, TCH and PBI approaches for generations 1, 10, 20, 30, 40 and 50, for a single run.

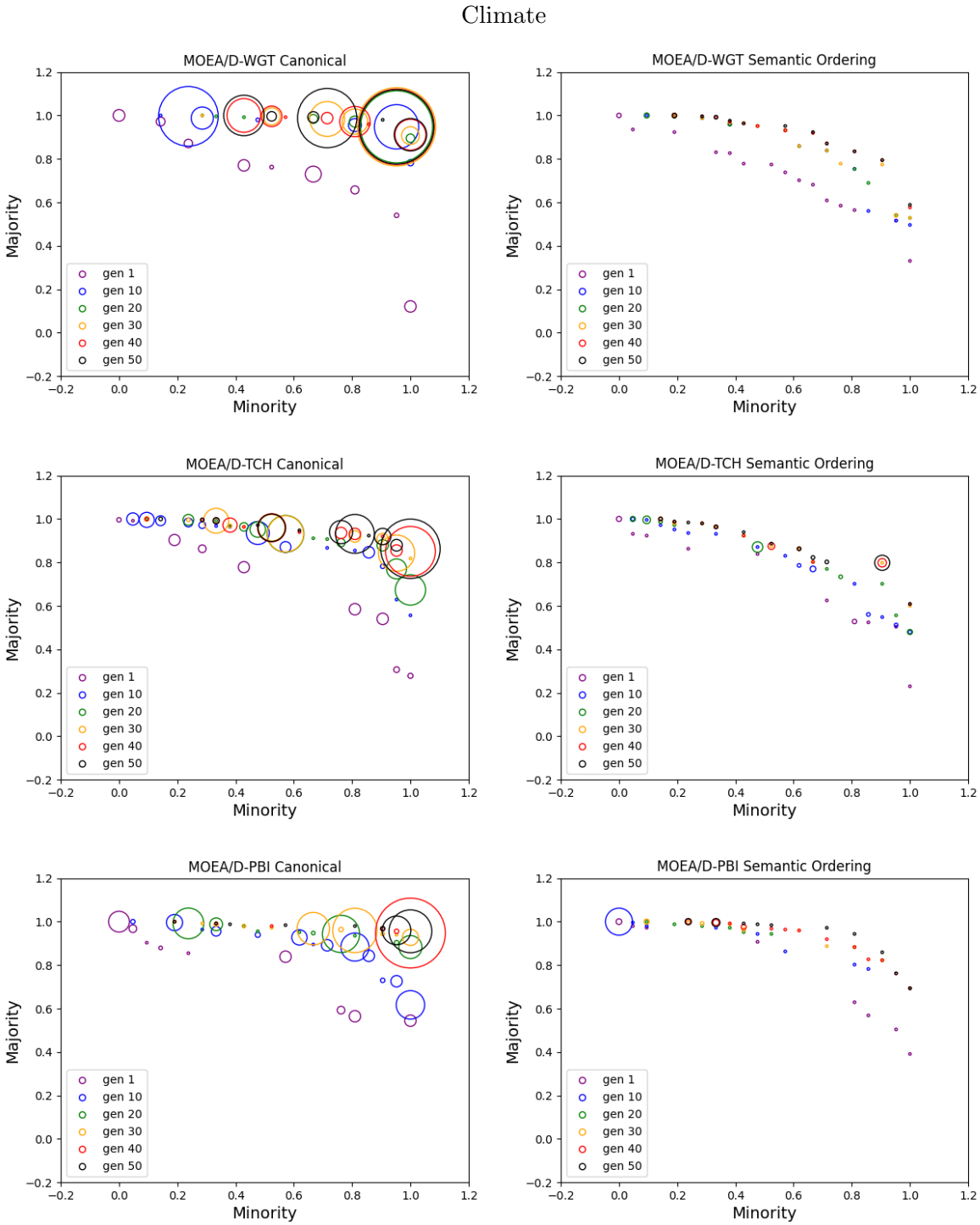


Figure A.13: Duplicate frequency of individuals at first Pareto Front for Climate dataset for WGT, TCH and PBI approaches for generations 1, 10, 20, 30, 40 and 50, for a single run.

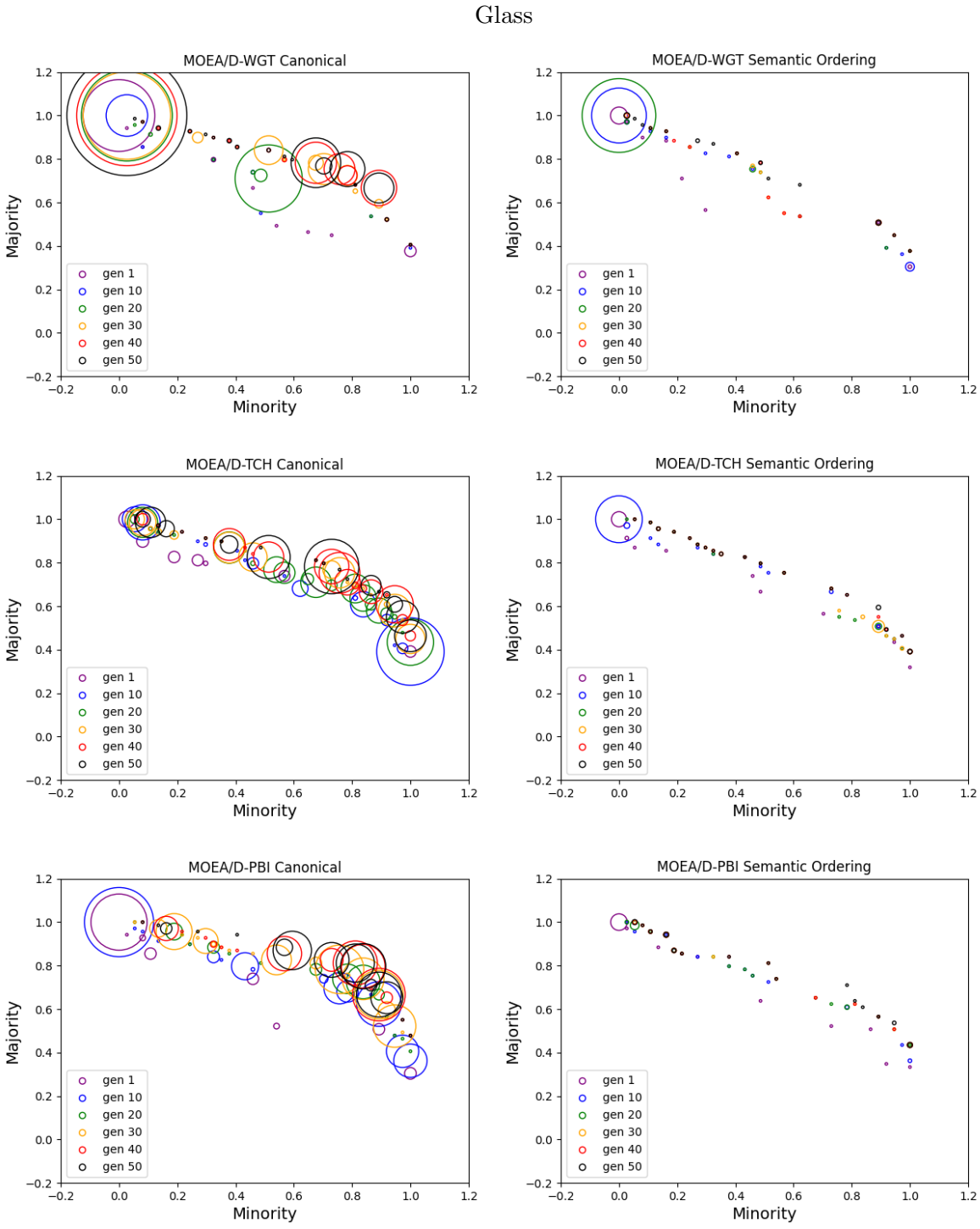


Figure A.14: Duplicate frequency of individuals at first Pareto Front for Glass dataset for WGT, TCH and PBI approaches for generations 1, 10, 20, 30, 40 and 50, for a single run.

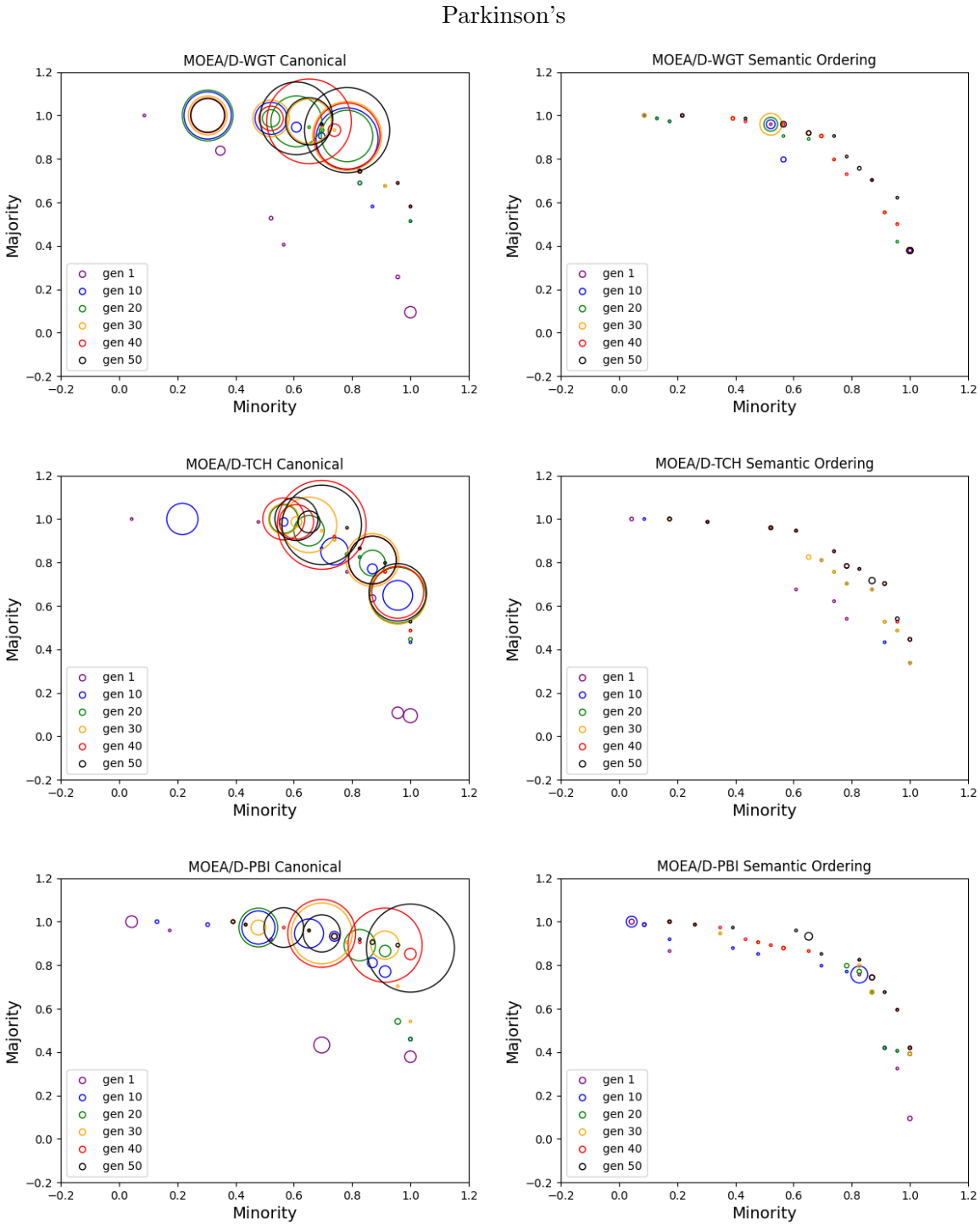


Figure A.15: Duplicate frequency of individuals at first Pareto Front for Parkinson's dataset for WGT, TCH and PBI approaches for generations 1, 10, 20, 30, 40 and 50, for a single run.

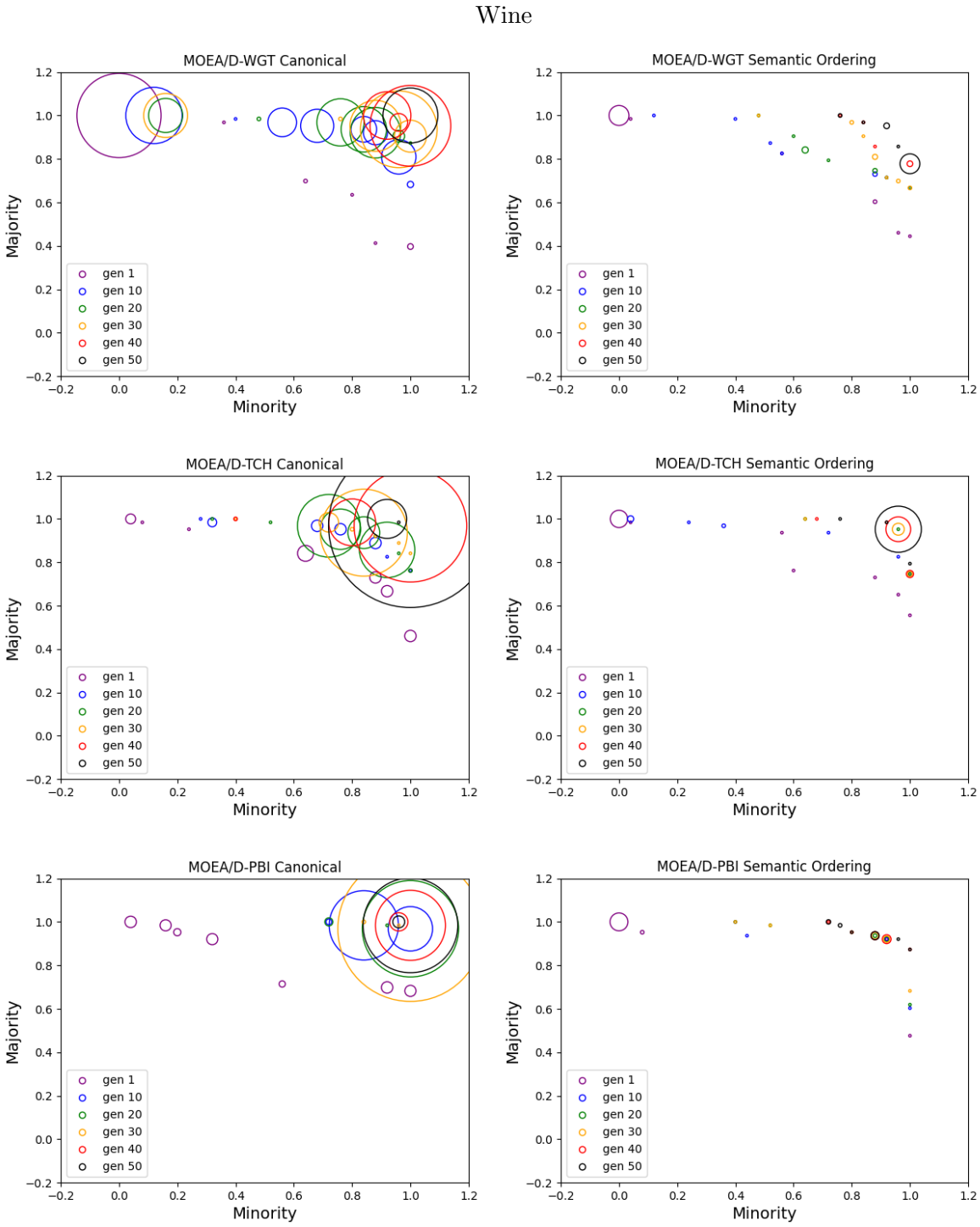


Figure A.16: Duplicate frequency of individuals at first Pareto Front for Wine dataset for WGT, TCH and PBI approaches for generations 1, 10, 20, 30, 40 and 50, for a single run.

B

Appendix B

B.1 Additional Figures Relating to Chapter 6

Fig. B.1 B.2 and B.3 are three examples of graphs created to test the NeuroLGP-MB approach from a list of 35. A key aspect in designing these tests was i) to ensure that the expected graph matched the genotype and ii) to ensure that graphs of suitable complexity were being tested. For instance, in each of these examples multiple concatenation layers are tested, with varying degrees at each concatenation node as well as varying degrees at the node in which the branch occurs. For instance, in Fig. B.1 the input layer has degree three as its edges branch into three different layers.

B.2 Example Configuration Files used in Chapters 5 and 6

The configuration files for the original NeuroLGP-V1 and NeuroLGP-MB are shown in Tables B.1 and B.2 respectively. Of note is the vast increase in potential layers and hyperparameters for NeuroLGP-MB over NeuroLGP. Furthermore, Table B.3 shows the genotype-to-phenotype mapping of functions to their corresponding layers and hyperparameters from Table B.2. In the case of the concatenation layer their is special handling and as such has not been included in this table. Additionally, there is special handling for the initial layer to ensure weight initialisation is specified.

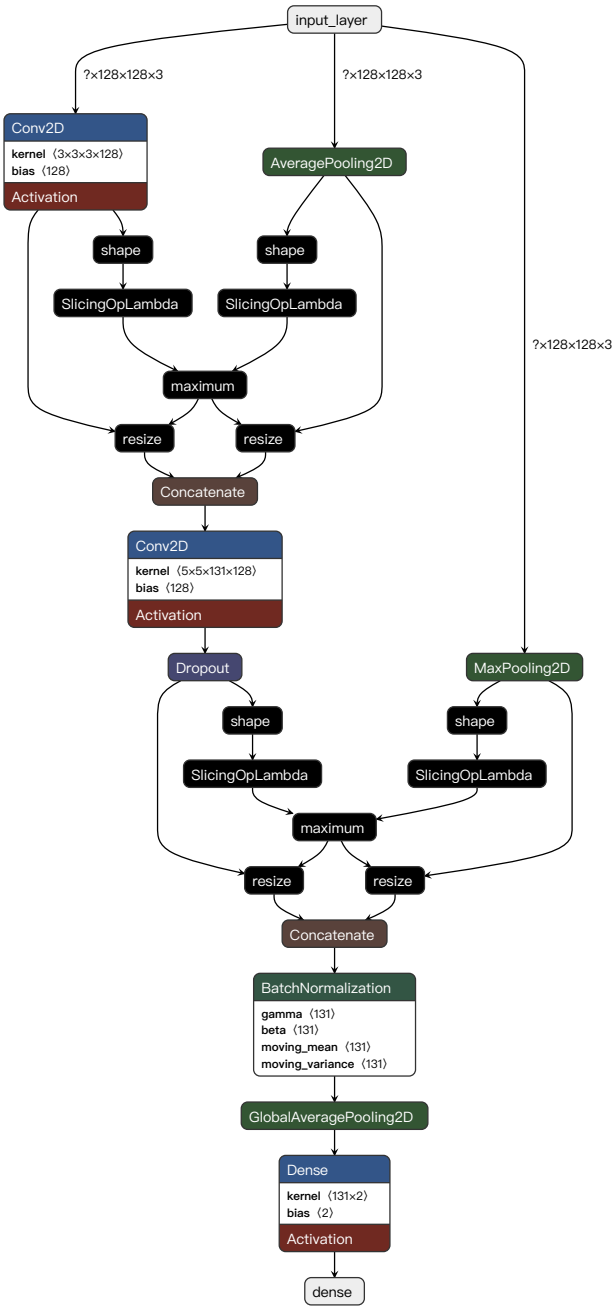


Figure B.1: Test #4, testing branching inputs

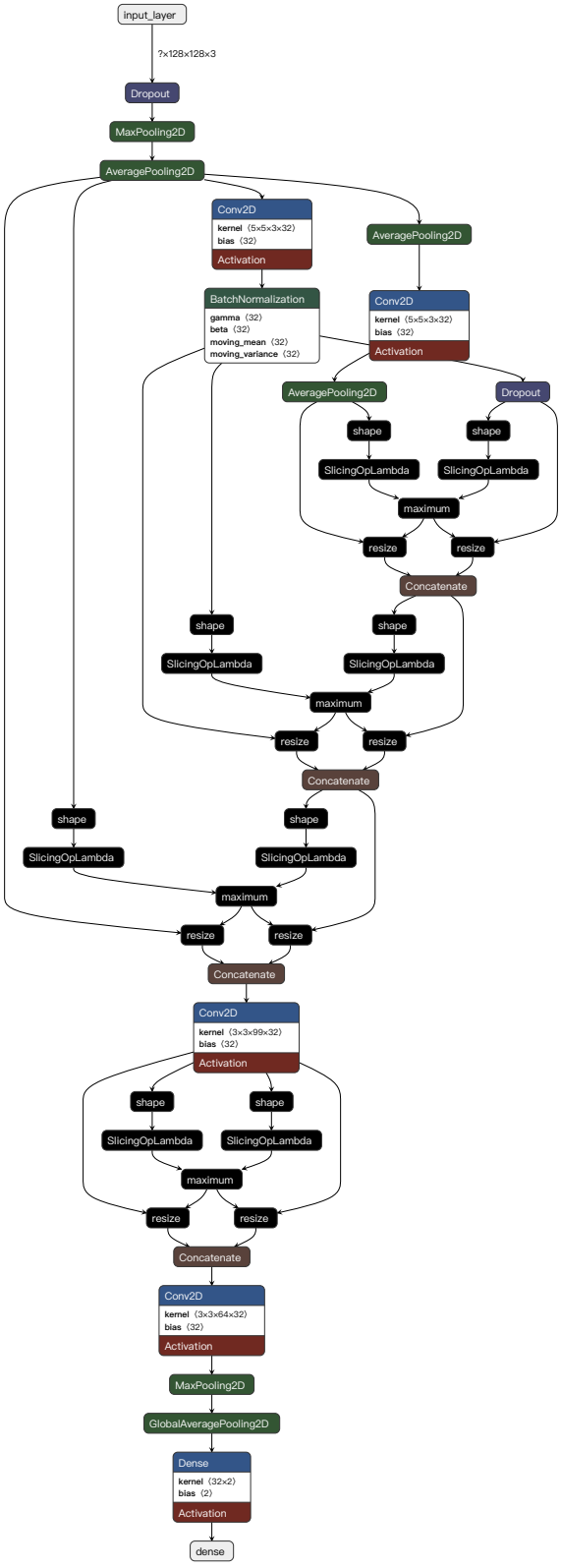


Figure B.2: Test #19, multiple sums, joining, 3 branches (complex topology)

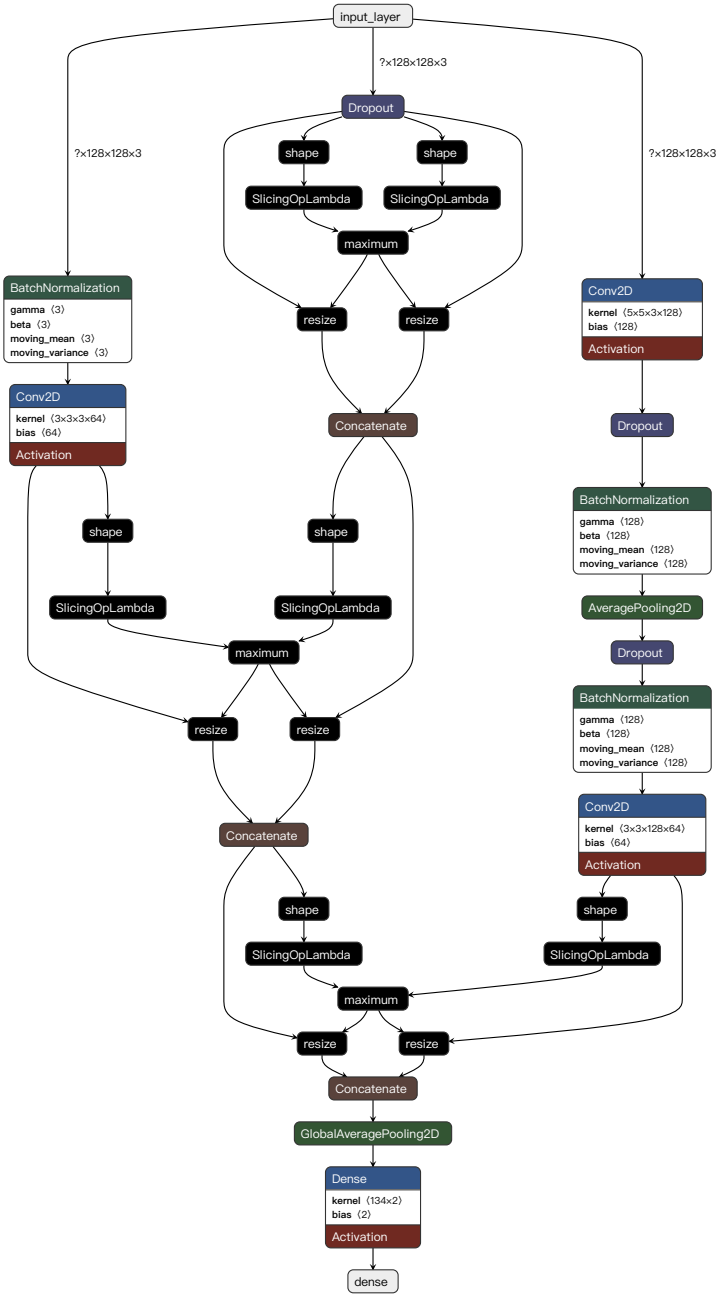


Figure B.3: Test #23, randomly generated

Category	Attribute	Value
PROBLEM DOMAIN		
	PROBLEM TYPE	Neuroevolution
	PROBLEM CODE	breakhis40
	DATA SOURCE	None
	INPUT DIMENSIONS	(64, 64, 3)
OPTIMIZATION		
	ALGORITHM	singleObjNeuroLGP
	NUMBER OF OBJECTIVES	1
	OBJECTIVE 1	mse
SURROGATE		
	METHOD	KPLS
	INITIAL EPOCH NUM	10
	INITIAL SURROGATE POPSIZE	30
	EXPENSIVE PROPORTION	0.4
EXPERIMENTAL SETUP		
	EXP ID	003
	RESULTS PATH	D:/Main/GitHub/GenProgMO/results/
	EXPERIMENT TYPE	surrogate
	DATA GENERATOR	true
	GEN SIZE	15
	POP SIZE	30
	ELITE PERCENTAGE	0.2
	SELECTION	TYPE: tournament, SIZE: 5, NUMBER OF TOURNAMENTS: 1
	CROSSOVER	RATE: 0.3 TYPE: linear, DISTANCE OF CROSSOVER POINTS: 5, MINIMUM PROGRAM LENGTH: 2, MAXIMUM PROGRAM LENGTH: 70, MAXIMUM SEGMENT LENGTH: 5, MAXIMUM DIFFERENCE IN SEGMENT LENGTH: 3
	MUTATION RATE	0.9
	MAX REGISTER	6
	CHROMOSOME LENGTH	70
	EPOCH NUM	30
PRIMITIVE SETUP		
	FUNCTION	CONV_32_3x3 CONV_32_5x5 CONV_64_3x3 CONV_64_5x5 CONV_128_3x3 CONV_128_5x5 AVGPOOL_2x2 MAXPOOL_2x2 BATCH_NORM_99 BATCH_NORM_9 DROPOUT_0.1_SEED0 DROPOUT_0.2_SEED0

Table B.1: Example configuration files for original NeuroLGP

Category	Attribute	Value
PROBLEM DOMAIN		
	PROBLEM TYPE	Neuroevolution
	PROBLEM CODE	BreakHis40_128
	DATA SOURCE	None
	INPUT DIMENSIONS	(128, 128, 3)
OPTIMIZATION		
	ALGORITHM	singleObjNeuroLGP
	NUMBER OF OBJECTIVES	1
	OBJECTIVE 1	mse
SURROGATE		
	METHOD	KPLS
	INITIAL EPOCH NUM	10
	INITIAL SURROGATE POPSIZE	30
	EXPENSIVE PROPORTION	0.4
EXPERIMENTAL SETUP		
	EXP ID	660
	RESULTS PATH	D:/Main/GitHub/GenProgMO/results/
	EXPERIMENT TYPE	surrogate
	DATA GENERATOR	true
	GEN SIZE	15
	POP SIZE	30
	ELITE PERCENTAGE	0.2
	SELECTION	TYPE: tournament, SIZE: 5, NUMBER OF TOURNAMENTS: 1
	CROSSOVER	RATE: 0.8 TYPE: linear, DISTANCE OF CROSSOVER POINTS: 80, MINIMUM PROGRAM LENGTH: 2, MAXIMUM PROGRAM LENGTH: 100, MAXIMUM SEGMENT LENGTH: 7, MAXIMUM DIFFERENCE IN SEGMENT LENGTH: 7
	MUTATE TIMES	5
	MUTATION RATE	0.5
	MUTATION OPERAND ONLY RATE	0.3
	MAX REGISTER	6
	CHROMOSOME LENGTH	30
	EPOCH NUM	30
PRIMITIVE SETUP		
	FUNCTION	CONV_32_3x3_KR00001 CONV_32_5x5_KR00001 CONV_64_3x3_KR00001 CONV_64_5x5_KR00001 CONV_128_3x3_KR00001 CONV_128_5x5_KR00001 CONV_32_3x3_NOL2 CONV_32_5x5_NOL2 CONV_64_3x3_NOL2 CONV_64_5x5_NOL2 CONV_128_3x3_NOL2 CONV_128_5x5_NOL2 CONV_32_3x3_KR00001_TANH CONV_32_5x5_KR00001_TANH CONV_64_3x3_KR00001_TANH CONV_64_5x5_KR00001_TANH CONV_128_3x3_KR00001_TANH CONV_128_5x5_KR00001_TANH
++++++	++++++++++++++++++++++++	++++++++++++++++++++++++++++++++++

++++++	++++++++++++++++++++++++	++++++++++++++++++++++++++++++++++

Category	Attribute	Value
PRIMITIVE SETUP (cont'd)		
	FUNCTION	CONV_32_3x3_KR01
		CONV_32_5x5_KR01
		CONV_64_3x3_KR01
		CONV_64_5x5_KR01
		CONV_128_3x3_KR01
		CONV_128_5x5_KR01
		CONV_32_3x3_KR01_TANH
		CONV_32_5x5_KR01_TANH
		CONV_64_3x3_KR01_TANH
		CONV_64_5x5_KR01_TANH
		CONV_128_3x3_KR01_TANH
		CONV_128_5x5_KR01_TANH
		AVGPOOL_2x2
		MAXPOOL_2x2
		AVGPOOL_3x3
		MAXPOOL_3x3
		AVGPOOL_5x5
		MAXPOOL_5x5
		BATCH_NORM_99
		BATCH_NORM_9
		BATCH_NORM_75
		BATCH_NORM_6
		BATCH_NORM_45
		BATCH_NORM_3
		BATCH_NORM_1
		BATCH_NORM_01
		DROPOUT_0.1_SEED0
		DROPOUT_0.2_SEED0
		DROPOUT_0.3_SEED0
		DROPOUT_0.4_SEED0
		DROPOUT_0.5_SEED0
		CONCAT

Table B.2: Example configuration files for NeuroLGP-MB

Genotype-to-Phenotype Mapping	
Function	**Layer**
CONV_32_3x3_KR00001	Conv2D(32, (3, 3), kernel_regularizer=l2(0.0001), activation='relu')
CONV_32_5x5_KR00001	Conv2D(32, (5, 5), kernel_regularizer=l2(0.0001), activation='relu')
CONV_64_3x3_KR00001	Conv2D(64, (3, 3), kernel_regularizer=l2(0.0001), activation='relu')
CONV_64_5x5_KR00001	Conv2D(64, (5, 5), kernel_regularizer=l2(0.0001), activation='relu')
CONV_128_3x3_KR00001	Conv2D(128, (3, 3), kernel_regularizer=l2(0.0001), activation='relu')
CONV_128_5x5_KR00001	Conv2D(128, (5, 5), kernel_regularizer=l2(0.0001), activation='relu')
CONV_32_3x3_NOL2	Conv2D(32, (3, 3), activation='relu')
CONV_32_5x5_NOL2	Conv2D(32, (5, 5), activation='relu')
CONV_64_3x3_NOL2	Conv2D(64, (3, 3), activation='relu')
CONV_64_5x5_NOL2	Conv2D(64, (5, 5), activation='relu')
CONV_128_3x3_NOL2	Conv2D(128, (3, 3), activation='relu')
CONV_128_5x5_NOL2	Conv2D(128, (5, 5), activation='relu')
CONV_32_3x3_KR00001_TANH	Conv2D(32, (3, 3), kernel_regularizer=l2(0.0001), activation='tanh')
CONV_32_5x5_KR00001_TANH	Conv2D(32, (5, 5), kernel_regularizer=l2(0.0001), activation='tanh')
CONV_64_3x3_KR00001_TANH	Conv2D(64, (3, 3), kernel_regularizer=l2(0.0001), activation='tanh')
CONV_64_5x5_KR00001_TANH	Conv2D(64, (5, 5), kernel_regularizer=l2(0.0001), activation='tanh')
CONV_128_3x3_KR00001_TANH	Conv2D(128, (3, 3), kernel_regularizer=l2(0.0001), activation='tanh')
CONV_128_5x5_KR00001_TANH	Conv2D(128, (5, 5), kernel_regularizer=l2(0.0001), activation='tanh')
CONV_32_3x3_NOL2_TANH	Conv2D(32, (3, 3), activation='tanh')
CONV_32_5x5_NOL2_TANH	Conv2D(32, (5, 5), activation='tanh')
CONV_64_3x3_NOL2_TANH	Conv2D(64, (3, 3), activation='tanh')
CONV_64_5x5_NOL2_TANH	Conv2D(64, (5, 5), activation='tanh')
CONV_128_3x3_NOL2_TANH	Conv2D(128, (3, 3), activation='tanh')
CONV_128_5x5_NOL2_TANH	Conv2D(128, (5, 5), activation='tanh')
CONV_32_3x3_KR01	Conv2D(32, (3, 3), kernel_regularizer=l2(0.1), activation='relu')
CONV_32_5x5_KR01	Conv2D(32, (5, 5), kernel_regularizer=l2(0.1), activation='relu')
CONV_64_3x3_KR01	Conv2D(64, (3, 3), kernel_regularizer=l2(0.1), activation='relu')
CONV_64_5x5_KR01	Conv2D(64, (5, 5), kernel_regularizer=l2(0.1), activation='relu')
CONV_128_3x3_KR01	Conv2D(128, (3, 3), kernel_regularizer=l2(0.1), activation='relu')
CONV_128_5x5_KR01	Conv2D(128, (5, 5), kernel_regularizer=l2(0.1), activation='relu')
CONV_32_3x3_KR01_TANH	Conv2D(32, (3, 3), kernel_regularizer=l2(0.1), activation='tanh')
CONV_32_5x5_KR01_TANH	Conv2D(32, (5, 5), kernel_regularizer=l2(0.1), activation='tanh')
CONV_64_3x3_KR01_TANH	Conv2D(64, (3, 3), kernel_regularizer=l2(0.1), activation='tanh')
CONV_64_5x5_KR01_TANH	Conv2D(64, (5, 5), kernel_regularizer=l2(0.1), activation='tanh')
CONV_128_3x3_KR01_TANH	Conv2D(128, (3, 3), kernel_regularizer=l2(0.1), activation='tanh')
CONV_128_5x5_KR01_TANH	Conv2D(128, (5, 5), kernel_regularizer=l2(0.1), activation='tanh')
BATCH_NORM_99	BatchNormalization(momentum = 0.99)
BATCH_NORM_9	BatchNormalization(momentum = 0.9)
BATCH_NORM_75	BatchNormalization(momentum = 0.75)
BATCH_NORM_6	BatchNormalization(momentum = 0.6)
BATCH_NORM_45	BatchNormalization(momentum = 0.45)
BATCH_NORM_3	BatchNormalization(momentum = 0.3)
BATCH_NORM_1	BatchNormalization(momentum = 0.1)
BATCH_NORM_01	BatchNormalization(momentum = 0.01)
AVGPOOL_2x2_1	AveragePooling2D(pool_size=(2, 2), strides=1)
MAXPOOL_2x2_1	MaxPooling2D(pool_size=(2, 2), strides=1)
AVGPOOL_2x2_2	AveragePooling2D(pool_size=(2, 2), strides=2)
MAXPOOL_2x2_2	MaxPooling2D(pool_size=(2, 2), strides=2)
AVGPOOL_3x3_1	AveragePooling2D(pool_size=(3, 3), strides=1)
MAXPOOL_3x3_1	MaxPooling2D(pool_size=(3, 3), strides=1)
AVGPOOL_3x3_2	AveragePooling2D(pool_size=(3, 3), strides=2)
MAXPOOL_3x3_2	MaxPooling2D(pool_size=(3, 3), strides=2)
AVGPOOL_3x3_3	AveragePooling2D(pool_size=(3, 3), strides=3)
MAXPOOL_3x3_3	MaxPooling2D(pool_size=(3, 3), strides=3)
AVGPOOL_5x5_1	AveragePooling2D(pool_size=(5, 5), strides=1)
MAXPOOL_5x5_1	MaxPooling2D(pool_size=(5, 5), strides=1)
AVGPOOL_5x5_2	AveragePooling2D(pool_size=(5, 5), strides=2)
MAXPOOL_5x5_2	MaxPooling2D(pool_size=(5, 5), strides=2)
AVGPOOL_5x5_3	AveragePooling2D(pool_size=(5, 5), strides=3)
MAXPOOL_5x5_3	MaxPooling2D(pool_size=(5, 5), strides=3)
DROPOUT_0.1_SEED0	Dropout(0.1, seed=0)
DROPOUT_0.2_SEED0	Dropout(0.2, seed=0)
DROPOUT_0.3_SEED0	Dropout(0.3, seed=0)
DROPOUT_0.4_SEED0	Dropout(0.4, seed=0)
DROPOUT_0.5_SEED0	Dropout(0.5, seed=0)

Table B.3: Genotype-to-phenotype mapping of NeuroLGP-MB

Bibliography

[1] Edgar Galván, Leonardo Trujillo, and Fergal Stapleton. Semantics in multi-objective genetic programming. *Applied Soft Computing*, 115:108143, 2022.

[2] Edgar Galván and Fergal Stapleton. Semantic-based distance approaches in multi-objective genetic programming. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 149–156. IEEE, 2020.

[3] Fergal Stapleton and Edgar Galván. Semantic neighborhood ordering in multi-objective genetic programming based on decomposition. In *2021 IEEE Congress on Evolutionary Computation (CEC)*, pages 580–587. IEEE, 2021.

[4] Edgar Galván, Leonardo Trujillo, and Fergal Stapleton. Highlights of semantics in multi-objective genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 19–20, 2022.

[5] Fergal Stapleton and Edgar Galván. Initial steps towards tackling high-dimensional surrogate modeling for neuroevolution using kriging partial least squares. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, GECCO '23 Companion, page 83–84, New York, NY, USA, 2023. Association for Computing Machinery.

[6] Fergal Stapleton, Brendan Cody-Kenny, and Edgar Galván. Neurolgp-sm: A surrogate-assisted neuroevolution approach using linear genetic programming. In *International Conference on Optimization and Learning (OLA)*, 2024.

[7] Fergal Stapleton and Edgar Galván. Neurolgp-sm: Scalable surrogate-assisted neuroevolution for deep neural networks. In *2024 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, 2024.

[8] Edgar Galván and Fergal Stapleton. Evolutionary multi-objective optimisation in neurotrajectory prediction. *Applied Soft Computing*, 146:110693, 2023.

[9] Fergal Stapleton, Edgar Galván, Ganesh Sistu, and Senthil Yogamani. Neuroevolutionary multi-objective approaches to trajectory prediction in autonomous vehicles. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '22, page 675–678, New York, NY, USA, 2022. Association for Computing Machinery.

[10] Markus Brameier and Wolfgang Banzhaf. *Linear genetic programming*, volume 1. Springer, 2007.

[11] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[12] François Chollet et al. Keras. `https://keras.io`, 2015.

[13] Lutz Roeder. Netron: Visualizer for neural network, deep learning and machine learning models., 2022.

[14] OpenAI. Gpt-4 technical report, 2024.

[15] Yifang Ma, Zhenyu Wang, Hong Yang, and Lin Yang. Artificial intelligence applications in the development of autonomous vehicles: A survey. *IEEE/CAA Journal of Automatica Sinica*, 7(2):315–329, 2020.

[16] Geert Litjens, Clara I Sánchez, Nadya Timofeeva, Meyke Hermsen, Iris Nagtegaal, Iringo Kovacs, Christina Hulsbergen-Van De Kaa, Peter Bult, Bram Van Ginneken, and Jeroen Van Der Laak. Deep learning as a tool for increased accuracy and efficiency of histopathological diagnosis. *Scientific reports*, 6(1):26286, 2016.

[17] Yann LeCun, Y. Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–44, 05 2015.

[18] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. *CoRR*, abs/1707.07012, 2018.

[19] Barret Zoph and Quoc V. le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[20] Agoston E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer Verlag, 2003.

[21] Gaurav Menghani. Efficient deep learning: A survey on making deep learning models smaller, faster, and better. *ACM Computing Surveys*, 55(12):1–37, 2023.

[22] Yaochu Jin. Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation*, 1(2):61–70, 2011.

[23] Alexander Hagg, Martin Zaefferer, Jörg Stork, and Adam Gaier. Prediction of neural network performance by phenotypic modeling. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1576–1582, 2019.

[24] Adam Gaier, Alexander Asteroth, and Jean-Baptiste Mouret. Data-efficient neuroevolution with kernel-based surrogate models. In *Proceedings of the genetic and evolutionary computation conference*, pages 85–92, 2018.

[25] Yanan Sun, Handing Wang, Bing Xue, Yaochu Jin, Gary G. Yen, and Mengjie Zhang. Surrogate-assisted evolutionary deep learning using an end-to-end random forest-based performance predictor. *IEEE Transactions on Evolutionary Computation*, 24(2):350–364, 2019.

[26] Alberto Sanfeliu and King-Sun Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE transactions on systems, man, and cybernetics*, (3):353–362, 1983.

[27] Jörg Stork, Martin Zaefferer, and Thomas Bartz-Beielstein. Improving neuroevolution efficiency by surrogate model-based optimization with phenotypic distance kernels. In *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*, pages 504–519. Springer, 2019.

[28] Nguyen Quang Uy, Nguyen Xuan Hoai, Michael O'Neill, R. I. McKay, and Edgar Galván-López. Semantically-based crossover in genetic programming: application to real-valued symbolic regression. *Genetic Programming and Evolvable Machines*, 12(2):91–119, 2011.

[29] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[31] Mohamed Amine Bouhlel, Nathalie Bartoli, Abdelkader Otsmane, and Joseph Morlier. Improving kriging surrogates of high-dimensional design models by partial least squares dimension reduction. *Structural and Multidisciplinary Optimization*, 53:935–952, 2016.

[32] Agoston E. Eiben and Jim Smith. From evolutionary computation to the evolution of things. *Nature*, 521:476–482, 05 2015.

[33] David E. Goldberg. *Genetic Algorithms in Search Optimization and Machine Learning.* Addison-Wesley, 1989.

[34] Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies –a comprehensive introduction. *Natural Computing: An International Journal*, 1(1):3–52, May 2002.

[35] Lawrence J. Fogel, Alvin J. Owens, and Michael J. Walsh. *Artificial intelligence through simulated evolution.* John Wiley & Sons, 1966.

[36] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, USA, 1992.

[37] Edgar Galván and Peter Mooney. Neuroevolution in deep neural networks: Current trends and future challenges. *IEEE Transactions on Artificial Intelligence*, 2:476–493, 2021.

[38] Edgar Galván and Marc Schoenauer. Promoting semantic diversity in multi-objective genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1021–1029, 2019.

[39] Tomasz P. Pawlak, Bartosz Wieloch, and Krzysztof Krawiec. Semantic backpropagation for designing search operators in genetic programming. *IEEE Transactions on Evolutionary Computation*, 19(3):326–340, 2014.

[40] Carlos A. Coello Coello. Evolutionary multi-objective optimization: a historical view of the field. *IEEE Computational Intelligence Magazine*, 1(1):28–36, Feb 2006.

[41] Carlos A. Coello Coello. A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information Systems*, 1(3):269–308, 1999.

[42] Kalyanmoy Deb. *Multi-Objective Optimization Using Evolutionary Algorithms.* John Wiley & Sons, Inc., New York, NY, USA, 2001.

[43] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[44] Hermann Minkowski. *Geometrie der zahlen.* BG Teubner, 1910.

[45] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. Spea2: Improving the strength pareto evolutionary algorithm. Technical report, Swiss Federal Institute of Technology Zurich (ETH), 2001.

[46] Qingfu Zhang and Hui Li. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation*, 11(6):712–731, 2007.

[47] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, July 2006.

[48] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[49] Mark A Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991.

[50] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[51] Henry A. Rowley, Shumeet Baluja, and Takeo Kanade. Neural network-based face detection. *IEEE Transactions on pattern analysis and machine intelligence*, 20(1):23–38, 1998.

[52] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017.

[53] Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on audio, speech, and language processing*, 22(10):1533–1545, 2014.

[54] Cícero dos Santos and Maíra Gatti. Deep convolutional neural networks for sentiment analysis of short texts. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 69–78, Dublin, Ireland, August 2014. Dublin City University and Association for Computational Linguistics.

[55] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, page 974–983, New York, NY, USA, 2018. Association for Computing Machinery.

[56] David H. Hubel and Torsten N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106, 1962.

[57] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[58] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[59] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[60] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[61] Colin White, Mahmoud Safari, Rhea Sukthanker, Binxin Ru, Thomas Elsken, Arber Zela, Debadeepta Dey, and Frank Hutter. Neural architecture search: Insights from 1000 papers. *arXiv preprint arXiv:2301.08727*, 2023.

[62] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[63] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[64] Sergey Ioffe. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[65] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.

[66] Frank Rosenblatt. Principles of neurodynamics: Perceptrons and the theory of brain mechanisms, 1961.

[67] John Denker, W. Gardner, Hans Graf, Donnie Henderson, R. Howard, W. Hubbard, Lawrence D. Jackel, Henry Baird, and Isabelle Guyon. Neural network recognizer for hand-written zip code digits. *Advances in neural information processing systems*, 1, 1988.

[68] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[69] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, nov 1997.

[70] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[71] Ingo Rechenberg. Evolutionsstrategien. In *Simulationsmethoden in der Medizin und Biologie: Workshop, Hannover, 29. Sept.–1. Okt. 1977*, pages 83–114. Springer, 1978.

[72] John H. Holland. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press, 1975.

[73] Nichael Lynn Cramer. A representation for the adaptive generation of simple sequential programs. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 183–187, 1985.

[74] JohnR. Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing*, 4, 06 1994.

[75] Erik Goodman. The 2023 humies awards. *ACM SIGEVOlution*, 16(3):1–4, 2023.

[76] Rainer Storn and Kenneth Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.

[77] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4, 1995.

[78] Michael O'Neill and Conor Ryan. Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, 5(4):349–358, 2001.

[79] Julian F. Miller et al. An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In *Proceedings of the genetic and evolutionary computation conference*, volume 2, pages 1135–1142, 1999.

[80] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

[81] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *International Conference on Learning Representations*, 2018.

[82] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *International conference on machine learning*, pages 7105–7114. PMLR, 2019.

[83] Alex Krizhevsky, Geoffrey E. Hinton, et al. Learning multiple layers of features from tiny images, 2009.

[84] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[85] Kaicheng Yu, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. *arXiv preprint arXiv:1902.08142*, 2019.

[86] Antoine Yang, Pedro M. Esperança, and Fabio M. Carlucci. Nas evaluation is frustratingly hard. In *International Conference on Learning Representations*, 2020.

[87] Darrell Whitley, Timothy Starkweather, and Christopher Bogart. Genetic algorithms and neural networks: Optimizing connections and connectivity. *Parallel computing*, 14(3):347–361, 1990.

[88] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evol. Comput.*, 10(2):99–127, June 2002.

[89] Kenneth O. Stanley, David B. D'Ambrosio, and Jason Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212, 2009.

[90] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, et al. Evolving deep neural networks. In *Artificial intelligence in the age of neural networks and brain computing*, pages 293–312. Elsevier, 2019.

[91] Travis Desell. Large scale evolution of convolutional neural networks using volunteer computing. In Peter A. N. Bosman, editor, *Genetic and Evolutionary Computation Conference, Berlin, Germany, July 15-19, 2017, Companion Material Proceedings*, pages 127–128. ACM, 2017.

[92] Kevin Swersky, David Duvenaud, Jasper Snoek, Frank Hutter, and Michael A Osborne. Raiders of the lost architecture: Kernels for bayesian optimization in conditional parameter spaces. *arXiv preprint arXiv:1409.4011*, 2014.

[93] Thomas Elsken, Jan-Hendrik Metzen, and Frank Hutter. Simple and efficient architecture search for convolutional neural networks, 2017.

[94] Martin Wistuba. Finding competitive network architectures within a day using uct. *arXiv preprint arXiv:1712.07420*, 2017.

[95] Linnan Wang, Yiyang Zhao, Yuu Jinnai, Yuandong Tian, and Rodrigo Fonseca. Neural architecture search using deep neural networks and monte carlo tree search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 9983–9991, 2020.

[96] Geoffrey F Miller, Peter M Todd, and Shailesh U Hegde. Designing neural networks using genetic algorithms. In *ICGA*, volume 89, pages 379–384, 1989.

[97] Steven Harp, Tariq Samad, and Aloke Guha. Designing application-specific neural networks using the genetic algorithm. *Advances in neural information processing systems*, 2, 1989.

[98] David J Montana, Lawrence Davis, et al. Training feedforward neural networks using genetic algorithms. In *IJCAI*, volume 89, pages 762–767, 1989.

[99] Xin Yao and Yong Liu. A new evolutionary system for evolving artificial neural networks. *IEEE transactions on neural networks*, 8(3):694–713, 1997.

[100] Jonas da Silveira Bohrer, Bruno Iochins Grisci, and Marcio Dorn. Neuroevolution of neural network architectures using codeepneat and keras, 2020.

[101] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[102] Lingxi Xie and Alan Yuille. Genetic cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1379–1388, 2017.

[103] Zhenhao Shuai, Hongbo Liu, Zhaolin Wan, Wei-Jie Yu, and Jun Zhang. A self-adaptive neuroevolution approach to constructing deep neural network architectures across different types. *Applied Soft Computing*, 136:110127, 2023.

[104] Yanan Sun, Bing Xue, Mengjie Zhang, and Gary G. Yen. Evolving deep convolutional neural networks for image classification. *IEEE Transactions on Evolutionary Computation*, 24(2):394–407, 2019.

[105] Krishna Teja Chitty-Venkata, Murali Emani, Venkatram Vishwanath, and Arun K Somani. Neural architecture search for transformers: A survey. *IEEE Access*, 10:108374–108412, 2022.

[106] Gustavo-Adolfo Vargas-Hakim, Efren Mezura-Montes, and Hector-Gabriel Acosta-Mesa. A review on convolutional neural network encodings for neuroevolution. *IEEE Transactions on Evolutionary Computation*, 26(1):12–27, 2021.

[107] Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao. A genetic programming approach to designing convolutional neural network architectures. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, page 497–504, New York, NY, USA, 2017. Association for Computing Machinery.

[108] Julian F. Miller. Cartesian genetic programming: its status and future. *Genetic Programming and Evolvable Machines*, 21(1):129–168, 2020.

[109] Filipe Assunção, Nuno Lourenço, Penousal Machado, and Bernardete Ribeiro. Evolving the topology of large scale deep neural networks. In Mauro Castelli, Lukas Sekanina, Mengjie Zhang, Stefano Cagnoni, and Pablo García-Sánchez, editors, *Genetic Programming*, pages 19–34, Cham, 2018. Springer International Publishing.

[110] Filipe Assunção, Nuno Lourenço, Penousal Machado, and Bernardete Ribeiro. Fast denser: Efficient deep neuroevolution. In *european conference on genetic programming*, pages 197–212. Springer, 2019.

[111] Filipe Assunção, Nuno Lourenço, Bernardete Ribeiro, and Penousal Machado. Fast-denser: Fast deep evolutionary network structured representation. *SoftwareX*, 14:100694, 2021.

[112] Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Freeze-thaw bayesian optimization. *arXiv preprint arXiv:1406.3896*, 2014.

[113] Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. Accelerating neural architecture search using performance prediction. *arXiv preprint arXiv:1705.10823*, 2017.

[114] Mohammed Imed Eddine Khaldi and Amer Draa. Surrogate-assisted evolutionary optimisation: a novel blueprint and a state of the art survey. *Evolutionary Intelligence*, pages 1–31, 2023.

[115] Shiqing Liu, Haoyu Zhang, and Yaochu Jin. A survey on computationally efficient neural architecture search. *Journal of Automation and Intelligence*, 1(1):100002, 2022.

[116] Boyang Deng, Junjie Yan, and Dahua Lin. Peephole: Predicting network performance before training. *arXiv preprint arXiv:1712.03351*, 2017.

[117] Bryson Greenwood and Tyler McDonnell. Surrogate-assisted neuroevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1048–1056, 2022.

[118] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Twenty-fourth international joint conference on artificial intelligence*, 2015.

[119] Liang Fan and Handing Wang. Surrogate-assisted evolutionary neural architecture search with network embedding. *Complex & Intelligent Systems*, 9(3):3313–3331, 2023.

[120] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. graph2vec: Learning distributed representations of graphs. *arXiv preprint arXiv:1707.05005*, 2017.

[121] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, pages 89–96, 2005.

[122] Shen Yan, Colin White, Yash Savani, and Frank Hutter. Nas-bench-x11 and the power of learning curves. *Advances in Neural Information Processing Systems*, 34:22534–22549, 2021.

[123] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 4780–4789. AAAI Press, 2019.

[124] Gonglin Yuan, Bing Xue, and Mengjie Zhang. An evolutionary neural architecture search method based on performance prediction and weight inheritance. *Information Sciences*, page 120466, 2024.

[125] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1314–1324, 2019.

[126] Agoston E. Eiben and Jim Smith. From evolutionary computation to the evolution of things. *Nature*, 521:476–482, 28 May 2015.

[127] Edgar Galván. Neuroevolution in deep learning: The role of neutrality. arXiv preprint arXiv: 2102.08475, 2021.

[128] Edgar Galván-López, James McDermott, Michael O'Neill, and Anthony Brabazon. Defining locality in genetic programming to predict performance. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2010, Barcelona, Spain, 18-23 July 2010*, pages 1–8, 2010.

[129] Edgar Galván-López, James McDermott, Michael O'Neill, and Anthony Brabazon. Defining locality as a problem difficulty measure in genetic programming. *Genetic Programming and Evolvable Machines*, 12(4):365–401, 2011.

[130] Alberto Moraglio, Krzysztof Krawiec, and Colin G. Johnson. Geometric semantic genetic programming. In Carlos A. Coello Coello, Vincenzo Cutello, Kalyanmoy Deb, Stephanie Forrest, Giuseppe Nicosia, and Mario Pavone, editors, *PPSN (1)*, volume 7491 of *LNCS*, pages 21–31. Springer, 2012.

[131] Edgar Galván-López, Efrén Mezura-Montes, Ouassim Ait ElHara, and Marc Schoenauer. On the use of semantics in multi-objective genetic programming. In Julia Handl et al., editors, *Parallel Problem Solving from Nature – PPSN XIV: 14th International Conference, Edinburgh, UK, September 17-21, 2016, Proceedings*, pages 353–363. Springer, 2016.

[132] Nicholas Freitag McPhee, Brian Ohs, and Tyler Hutchison. Semantic building blocks in genetic programming. In *Proceedings of the 11th European conference on Genetic programming*, EuroGP'08, pages 134–145, Berlin, Heidelberg, 2008. Springer-Verlag.

[133] Quang Uy Nguyen, Xuan Hoai Nguyen, and Michael O'Neill. Semantic aware crossover for genetic programming: the case for real-valued function regression. In *Genetic Programming: 12th European Conference, EuroGP 2009 Tübingen, Germany, April 15-17, 2009 Proceedings 12*, pages 292–302. Springer, 2009.

[134] Edgar Galván-López, Brendan Cody-Kenny, Leonardo Trujillo, and Ahmed Kattan. Using semantics in the selection mechanism in genetic programming: A simple method for promoting semantic diversity. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2013, Cancun, Mexico, June 20-23, 2013*, pages 2972–2979, 2013.

[135] Stefan Forstenlechner, David Fagan, Miguel Nicolau, and Michael O'Neill. Towards effective semantic operators for program synthesis in genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '18, pages 1119–1126, New York, NY, USA, 2018. ACM.

[136] Nguyen Quang Uy, Michael O'Neill, Nguyen Xuan Hoai, Bob Mckay, and Edgar Galván-López. *Semantic Similarity Based Crossover in GP: The Case for Real-Valued Function Regression*, pages 170–181. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[137] Alberto Moraglio and Riccardo Poli. *Topological Interpretation of Crossover*, pages 1377–1388. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[138] Frederico JJB Santos, Ivo Gonçalves, and Mauro Castelli. Neuroevolution with box mutation: An adaptive and modular framework for evolving deep neural networks. *Applied Soft Computing*, page 110767, 2023.

[139] Ivo Gonçalves, Sara Silva, and Carlos M Fonseca. Semantic learning machine: a feedforward neural network construction algorithm inspired by geometric semantic genetic programming. In *Progress in Artificial Intelligence: 17th Portuguese Conference on Artificial Intelligence, EPIA 2015, Coimbra, Portugal, September 8-11, 2015. Proceedings 17*, pages 280–285. Springer, 2015.

[140] Lawrence Beadle and Colin G Johnson. Semantically driven mutation in genetic programming. In *2009 IEEE Congress on Evolutionary Computation*, pages 1336–1342. IEEE, 2009.

[141] Lawrence Beadle and Colin G. Johnson. Semantically driven crossover in genetic programming. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2008, June 1-6, 2008, Hong Kong, China*, pages 111–116. IEEE, 2008.

[142] William B Langdon and Riccardo Poli. *Foundations of genetic programming*. Springer Science & Business Media, 2013.

[143] Edgar Galván-López, Tom Curran, James McDermott, and Paula Carroll. Design of an autonomous intelligent demand-side management system using stochastic optimisation evolutionary algorithms. *Neurocomputing*, 170:270–285, 2015.

[144] Edgar Galván-López, Adam Taylor, Siobhán Clarke, and Vinny Cahill. Design of an automatic demand-side management system based on evolutionary algorithms. In *Proceedings of the 29th Annual Symposium on Applied Computing, SAC '14*, pages 525 – 530, Gyeongju, Korea, 26-28 March 2014. ACM.

[145] Edgar Galván-López and Riccardo Poli. Some steps towards understanding how neutrality affects evolutionary search. In Thomas Philip Runarsson, Hans-Georg Beyer, Edmund K. Burke, Juan Juli'an Merelo Guerv'os, L. Darrell Whitley, and

Xin Yao, editors, *Parallel Problem Solving from Nature - PPSN IX, 9th International Conference, Reykjavik, Iceland, September 9-13, 2006, Procedings*, volume 4193, pages 778–787. Springer, 2006.

[146] Edgar Galván-López, Stephen Dignum, and Riccardo Poli. The effects of constant neutrality on performance and problem hardness in GP. In Michael O'Neill, Leonardo Vanneschi, Steven M. Gustafson, Anna Esparcia-Alcázar, Ivanoe De Falco, Antonio Della Cioppa, and Ernesto Tarantino, editors, *Genetic Programming, 11th European Conference, EuroGP 2008, Naples, Italy, March 26-28, 2008. Proceedings*, volume 4971 of *Lecture Notes in Computer Science*, pages 312–324. Springer, 2008.

[147] Riccardo Poli and Edgar Galván-López. On the effects of bit-wise neutrality on fitness distance correlation, phenotypic mutation rates and problem hardness. In Christopher R. Stephens, Marc Toussaint, Darrell Whitley, and Peter F. Stadler, editors, *Foundations of Genetic Algorithms*, pages 138–164, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[148] Riccardo Poli and Edgar Galván-López. The effects of constant and bit-wise neutrality on problem hardness, fitness distance correlation and phenotypic mutation rates. *IEEE Trans. Evolutionary Computation*, 16(2):279–300, 2012.

[149] Edgar Galván-López, Riccardo Poli, Ahmed Kattan, Michael O'Neill, and Anthony Brabazon. Neutrality in evolutionary algorithms... what do we know? *Evolving Systems*, 2(3):145–163, 2011.

[150] Edgar Galván-López, James McDermott, Michael O'Neill, and Anthony Brabazon. Towards an understanding of locality in genetic programming. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, GECCO '10, pages 901–908, New York, NY, USA, 2010. ACM.

[151] Edgar Galván-López, Riccardo Poli, and Carlos A. Coello Coello. Reusing code in genetic programming. In Maarten Keijzer, Una-May O'Reilly, Simon M. Lucas, Ernesto Costa, and Terence Soule, editors, *Genetic Programming, 7th European Conference, EuroGP2004, Coimbra, Portugal, April 5-7, 2004, Proceedings*, volume 3003 of *Lecture Notes in Computer Science*, pages 359–368. Springer, 2004.

[152] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[153] Krzysztof Krawiec and Tomasz Pawlak. Locally geometric semantic crossover: a study on the roles of semantics and homology in recombination operators. *Genetic Programming and Evolvable Machines*, 14:31–63, 2013.

[154] Urvesh Bhowan, Mark Johnston, Mengjie Zhang, and Xin Yao. Evolving diverse ensembles using genetic programming for classification with unbalanced data. *IEEE Transactions on Evolutionary Computation*, 17(3):368–386, 2012.

[155] Urvesh Bhowan, Mengjie Zhang, and Mark Johnston. Multi-objective genetic programming for classification with unbalanced data. In Ann Nicholson and Xiaodong Li, editors, *AI 2009: Advances in Artificial Intelligence*, pages 370–380, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[156] Arthur Asuncion, David Newman, et al. Uci machine learning repository, 2007.

[157] Carlos A. Coello Coello, Gary B. Lamont, and David A. Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[158] Zhixing Huang, Yi Mei, and Jinghui Zhong. Semantic linear genetic programming for symbolic regression. *IEEE Transactions on Cybernetics*, 2022.

[159] Stefan Forstenlechner. *Program synthesis with grammars and semantics in genetic programming*. PhD thesis, University College Dublin, 2019.

[160] Shuchao Deng, Zeqiong Lv, Edgar Galván, and Yanan Sun. Evolutionary neural architecture search for facial expression recognition. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 7(5):1405–1419, 2023.

[161] Waseem Rawat and Zenghui Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural computation*, 29(9):2352–2449, 2017.

[162] Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. (With contributions by J. R. Koza).

[163] Yuqiao Liu, Yanan Sun, Bing Xue, Mengjie Zhang, Gary Yen, and Kay Tan. A survey on evolutionary neural architecture search. *IEEE transactions on neural networks and learning systems*, PP, 08 2021.

[164] Paul Templier, Emmanuel Rachelson, and Dennis G Wilson. A geometric encoding for neural network evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 919–927, 2021.

[165] Anthony M Zador. A critique of pure learning and what artificial neural networks can learn from animal brains. *Nature communications*, 10(1):1–7, 2019.

[166] Julian F. Miller. *Cartesian Genetic Programming*, pages 17–34. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[167] Garnett Wilson and Wolfgang Banzhaf. A comparison of cartesian genetic programming and linear genetic programming. In *European Conference on Genetic Programming*, pages 182–193. Springer, 2008.

[168] Dario Izzo, Francesco Biscani, and Alessio Mereta. Differentiable genetic programming. In *Genetic Programming: 20th European Conference, EuroGP 2017, Amsterdam, The Netherlands, April 19-21, 2017, Proceedings 20*, pages 35–51. Springer, 2017.

[169] Marcus Märtens and Dario Izzo. Neural network architecture search with differentiable cartesian genetic programming for regression. In *Proceedings of the genetic and evolutionary computation conference companion*, pages 181–182, 2019.

[170] Mohamed Amine Bouhlel, John T. Hwang, Nathalie Bartoli, Rémi Lafage, Joseph Morlier, and Joaquim R. R. A. Martins. A python surrogate modeling framework with derivatives. *Advances in Engineering Software*, page 102662, 2019.

[171] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[172] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13:455–492, 1998.

[173] Fabio Alexandre Spanhol, Luiz S Oliveira, Caroline Petitjean, and Laurent Heutte. Breast cancer histopathological image classification using convolutional neural networks. In *2016 international joint conference on neural networks (IJCNN)*, pages 2560–2567. IEEE, 2016.

[174] Yassir Benhammou, Boujemaa Achchab, Francisco Herrera, and Siham Tabik. Breakhis based breast cancer automatic diagnosis using deep learning: Taxonomy, survey and insights. *Neurocomputing*, 375:9–24, 2020.

[175] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[176] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[177] Silvia Cascianelli, Raquel Bello-Cerezo, Francesco Bianconi, Mario L Fravolini, Mehdi Belal, Barbara Palumbo, and Jakob N Kather. Dimensionality reduction strategies for cnn-based classification of histopathological images. In *Intelligent Interactive Multimedia Systems and Services 2017 10*, pages 21–30. Springer, 2018.

[178] Vibha Gupta and Arnav Bhavsar. An integrated multi-scale model for breast cancer histopathological image classification with joint colour-texture features. In *Computer Analysis of Images and Patterns: 17th International Conference, CAIP 2017, Ystad, Sweden, August 22-24, 2017, Proceedings, Part II 17*, pages 354–366. Springer, 2017.

[179] Mukta Sharma, Rahul Singh, and Mahua Bhattacharya. Classification of breast tumors as benign and malignant using textural feature descriptor. In *2017 Ieee international conference on bioinformatics and biomedicine (bibm)*, pages 1110–1113. IEEE, 2017.

[180] Abdullah-Al Nahid and Yinan Kong. Histopathological breast-image classification using local and frequency domains by convolutional neural network. *Information*, 9(1):19, 2018.

[181] Abdullah-Al Nahid, Mohamad Ali Mehrabi, Yinan Kong, et al. Histopathological breast cancer image classification by deep neural network techniques guided by local clustering. *BioMed research international*, 2018, 2018.

[182] R Karthiga and K Narasimhan. Automated diagnosis of breast cancer using wavelet based entropy features. In *2018 Second international conference on electronics, communication and aerospace technology (ICECA)*, pages 274–279. IEEE, 2018.

[183] Sawon Pratiher and Subhankar Chattoraj. Diving deep onto discriminative ensemble of histological hashing & class-specific manifold learning for multi-class breast carcinoma taxonomy. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1025–1029. IEEE, 2019.

[184] Joke A. Badejo, Emmanuel Adetiba, Adekunle Akinrinmade, and Matthew B. Akanle. Medical image classification with hand-designed or machine-designed texture descriptors: a performance evaluation. In *Bioinformatics and Biomedical Engineering: 6th International Work-Conference, IWBBIO 2018, Granada, Spain, April 25–27, 2018, Proceedings, Part II 6*, pages 266–275. Springer, 2018.

[185] Abdullah-Al Nahid, Aaron Mikaelian, and Yinan Kong. Histopathological breast-image classification with restricted boltzmann machine along with backpropagation. *Biomedical Research*, 29(10):2068–2077, 2018.

[186] Kausik Das, Sailesh Conjeti, Abhijit Guha Roy, Jyotirmoy Chatterjee, and Debdoot Sheet. Multiple instance learning of deep convolutional neural networks for breast histopathology whole slide classification. In *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*, pages 578–581. IEEE, 2018.

[187] Kundan Kumar and Annavarapu Chandra Sekhara Rao. Breast cancer classification of image using convolutional neural network. In *2018 4th International Conference on Recent Advances in Information Technology (RAIT)*, pages 1–6. IEEE, 2018.

[188] MA Aswathy and M Jagannath. An svm approach towards breast cancer classification from h&e-stained histopathology images based on integrated features. *Medical & biological engineering & computing*, 59(9):1773–1783, 2021.

[189] Atharv Bhosekar and Marianthi Ierapetritou. Advances in surrogate based modeling, feasibility analysis, and optimization: A review. *Computers & Chemical Engineering*, 108:250–267, 2018.

[190] Loïc Lannelongue, Jason Grealey, and Michael Inouye. Green algorithms: quantifying the carbon footprint of computation. *Advanced science*, 8(12):2100707, 2021.

[191] Sungheon Park and Nojun Kwak. Analysis on the dropout effect in convolutional neural networks. In *Computer Vision–ACCV 2016: 13th Asian Conference on Computer Vision, Taipei, Taiwan, November 20-24, 2016, Revised Selected Papers, Part II 13*, pages 189–204. Springer, 2017.

[192] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[193] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017.

[194] Guoping Xu, Xiaxia Wang, Xinglong Wu, Xuesong Leng, and Yongchao Xu. Development of skip connection in deep neural networks for computer vision and medical image analysis: A survey. *arXiv preprint arXiv:2405.01725*, 2024.

[195] Kaspar Riesen and Horst Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision computing*, 27(7):950–959, 2009.

[196] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of*

the IEEE conference on computer vision and pattern recognition, pages 2818–2826, 2016.

[197] Daniel H. Stolfi and Enrique Alba. Epigenetic algorithms: A new way of building gas based on epigenetics. *Information Sciences*, 424:250–272, 2018.

[198] William La Cava, Thomas Helmuth, Lee Spector, and Kourosh Danai. Genetic programming with epigenetic local search. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 1055–1062, 2015.

[199] Gunter Meister and Thomas Tuschl. Mechanisms of gene silencing by double-stranded rna. *Nature*, 431(7006):343–349, 2004.

[200] Ronald L Iman and Michael J Shortencarier. Fortran 77 program and user's guide for the generation of latin hypercube and random samples for use with computer models. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 1984.

[201] Daniel Kermany, Kang Zhang, Michael Goldbaum, et al. Labeled optical coherence tomography (oct) and chest x-ray images for classification. *Mendeley data*, 2(2):651, 2018.

[202] Daniel S Kermany, Michael Goldbaum, Wenjia Cai, Carolina CS Valentim, Huiying Liang, Sally L Baxter, Alex McKeown, Ge Yang, Xiaokang Wu, Fangbing Yan, et al. Identifying medical diagnoses and treatable diseases by image-based deep learning. *cell*, 172(5):1122–1131, 2018.

[203] Bastian Rieck, Matteo Togninalli, Christian Bock, Michael Moor, Max Horn, Thomas Gumbsch, and Karsten Borgwardt. Neural persistence: A complexity measure for deep neural networks using algebraic topology. *arXiv preprint arXiv:1812.09764*, 2018.