UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

COS700 Research Report

Genetic Programming for the Automated Design of Cross-Domain Selection Perturbative Hyper-Heuristics

Student number: u17192968

Supervisor(s):
Prof. Nelishia Pillay

Rudo Janse van Rensburg

October 2021

Contents

Abstract

Hyper-Heuristics are used as a means of improving the generality of heuristic algorithms and have shown great success at solving discrete combinatorial optimisation problems. Creating hyper-heuristics to be applicable across different domains, however, proves to be a difficult task given they need to be domain agnostic and thus cannot incorporate domain knowledge into their design. Consequently, algorithms have been proposed to automate the process of designing these cross-domain hyper-heuristics. This paper propose a novel method of automating the design of selective perturbative Hyper-Heuristics which incorporate Genetic Programming such that the classification prowess of Genetic Programming is harnessed.

Keywords:

Automated Design, Cross-Domain, Hyper-Heuristic, Classification, Genetic Programming

1 Introduction

1.1 Precursor

The advances of searching algorithms in the realm of Computer Science has given rise to informed searches by means of heuristics; which help guide the search to mitigate exploring an entire solution space to find an optimal solution. The trend of research has since pushed to generalise algorithms from only solving a particular problem in one problem domain, to being able to solve problems from multiple domains.

To this end, the idea of Hyper-Heuristics has developed, which uses high-level heuristics to traverse a low-level heuristic space rather than a solution space directly, which is more generally applicable to multiple problems from the same problem domain.

The potential of Hyper-Heuristics have been further realised in 2011 when the Cross-Domain Heuristic Search Challenge (CHeSC 2011) [GM11] prompted competing researchers to extend the generalising capacity of their Hyper-Heuristic algorithms to problems from multiple problem domains. Furthermore, this competition gave birth to HyFlex [OGH+12], a framework by which researchers could compare the performance of their Hyper-Heuristic algorithms, and which will be used in this project to evaluate the performance of the proposed solution.

1.2 Objectives

The paper will propose a novel Selective Pertubative Hyper-Heuristic to pit against state-of-art algorithms and determine how well it compares. Thus, in accordance therewith, first objective of the proposed project is to survey and investigate existing algorithms that automate the design of Selective Perturbative Hyper-Heuristics for solving the discrete combinatorial problems encompassed specifically by the CHeSC challenge, in particular algorithms which are represented by classifiers. The algorithms investigated here, and their relative performance to the winning algorithms of the CHeSC challenge, will be used as a basis for evaluating and comparing the performance of the proposed algorithm throughout the developmental and experimental phase of the project.

The second objective of the projects encompasses the development and construction of a novel Genetic Programming solution to Automate the Design of Selective Perturbative Hyper-Heuristics, and which specifically represents the tasks as one of a classification nature. The classification capacity of Genetic Programming has been thoroughly demonstrated Espejo Espejo et al. [PSF10], and it is this capacity that the proposed algorithm will attempt to harness.

1.3 Layout

Following this section will be the Problem Statement, which will serve to refine the scope and purpose of the paper. Thereafter, Methodology will be discussed, encapsulating the research method, experimental setup and will elaborate on implementation details. Following this, will be a Background which will include a the literature survey about topics relevant to this research, ensuring the Related Work is also presented. This will be followed by a Discussion, wherein the algorithm's results will be presented. The Conclusion tie everything together and summarise the findings of the research.

2 Problem Statement

Automated Design of Selective Pertubative Hyper-Heuristics for solving discrete combinatorial problems is a relatively well covered domain, in part as a consequence of (CHeSC 2011) and the introduction of HyFlex. Little work has, however, been done on the subject of using Genetic Programming for this purpose, and none of which automate the design in terms of classification; an idea which is has been shown to be successful when using other machine learning techniques.

The sole question that this project will attempt to answer is:

How effective can a Genetic Program be used to automate the design of selection perturbative hyper-heuristics when built as a classifier?

This question will be answered by comparing the performance of the proposed genetic program against other algorithms that have been applied to the HyFlex problem domains.

In accordance with the aforementioned question, the main outcomes of the project are as follow:

3 Background

3.1 Hyper-Heuristics

3.1.1 Introduction

Computationally hard search problems often necessitate a means of narrowing the scope of the search to avoid traversing an entire search space to find a solution. For this purpose, Heuristics are used. In principal, heuristics serve this purpose, however not without a few caveats. For one, the effectiveness of the heuristic is reliant on the implementation and expertise of it's designer, as well as domain knowledge of the problem. Furthermore, these heuristics were inherently specialised to a particular problem for which they were designed, and could not be applied generally [NP18]. To address these, researchers explored means of automating the design of heuristics. Whilst this idea of automated heuristic design can be traced back to the 1960's, the term "Hyper-Heuristic" was only coined as recently as the early 2000's [CKS00]. In laymen's terms, hyper-heuristics are heuristics for heuristics. A higher-level hyper-heuristic explores a search space of low-level heuristics which in turn describes the solution space [CDF+12]. The low-level heuristics would incorporate the difficulty or quality of a move within the solution space, and the Hyper-heuristic uses these to guide actions of the High-level heuristics.

3.1.2 Taxonomy of Hyper-Heuristics

Hyper-Heuristics can be classified by 2 primary criteria; the manner in which the hyper-heuristic interacts with the heuristic space and the function of the low-level heuristics constituting the heuristic space.

Conceptually, Hyper-Heuristics interact with low-level heuristics constituting the heuristic space in one of two ways; selection or generation.

[EMM+13]For a problem domain, either low-level heuristics are nominated to create or optimise a solution for the problem - and in this way these low-level heuristics are selected - or, new low-level heuristics are created - and this way low-level heuristics are generated. The difference between the two is in part when and where the low-level heuristic is used in the search for a solution and by these attributes can be categorised into 2 classes; constructive and perturbative.

Typically, constructive heuristics are used to guide the creation of initial solutions to a problem which will act as a basis whereupon optimisation techniques can build - and in this way is involved in the construction of solutions. On the other hand, the function of perturbative heuristics is to improve upon existing initial solutions. To explore the Heuristic Space, changes are made to an initial solution so as to perturb the solution space, which is explored by the Hyper-Heuristic by means of the low-level heuristics.

From the 2 aforementioned criteria for classifying Hyper-Heuristics, we end up with 4 classes; Selective Constructive, Selective Perturbative, Generative Constructive and Generative Perturbative - with Selective Pertubative Hyper-Heuritics being the focus of this research.

3.1.3 Selection Perturbative Hyper-Heuristics

For a particular problem, there are a number of pre-defined low-level perturbative heuristics wh that the Hyper-heuristic will have to choose from. After an initial solution is created, the selective perturbative hyper-heuristic will iteratively apply a selected low-level heuristic to the solution, producing a new refined solution. This process of iterative refinement is perpetuated so long as there are observable improvements to the solution. The metric by which solutions are evaluated for improvement are problem specific

Algorithm 1: How Selective Perturbative Hyper-Heuristics work

```
   Result: an optimal solution to a particular problem domain.
1  Create initial solution;
2  do
3      Select a low-level perturbative heuristic to apply;
4      Apply low-level heuristic to solution;
5  while solutions have improved;
6  return The best solution
```

3.2 Cross-Domain Hyper-Heuristics

3.2.1 Introduction

Cross-Domain Hyper-Heuristics aim to extend the capacity for generalisation, by widening the scope of a hyper-heuristic to multiple domains.[NP18]

3.2.2 Cross-Domain Heuristics Search Challenge

The Cross-Domain Heuristic Search Challenge (CHeSC 2011)[GM11] was a challenge that was held in 2011 which fostered new ideas in the area of research pertaining Cross-Domain Hyper-Heuristics, as well as producing a framework by which to construct these Cross-Domain Hyper-Heuristics; namingly HyFlex[OGH+12].

Given that CHeSC was a competition, it had a set of rules by which competitors had to abide. At the development phase of the competition, competitors were informed of 4 problem domains that their proposed solution would be tested against; namely Maximum Satisfiability, Bin Packing, Permutation Flowshop and Personnel Scheduling. The competition also involved 2 hidden problem domains, namingly Travelling Salesman and the Vehicle Routing Problem. These hidden problem domains were not divulged to competitors at the development phase of the competition.

During the testing phase, 3 known domains sampled from the 4 problem domains as well as both of the hidden domains were used to evaluate the performance of the competitors' solutions. It was theorised that the solution that generalised best across domains would have performed better on the unknown domains [CDF+12].

3.2.3 HyFlex

HyFlex is a Java program which acts as a framework wherein several discrete combinatorial optimisation problems can be solved; namingly Maximum Satisfiability, One-Dimensional Bin Packing, Permutation Flow Shop, Personnel Scheduling, Travelling Salesman and Vehicle Routing.

HyFlex includes 4 problem specific categories of low-level perturbative heuristics 6for the aforementioned problems, which include Mutational heuristics, Ruin- recreate heuristics, Local search heuristics and Crossover [NP18]. Mutational Heuristics function by performing swapping, changing, adding or deletion solution components in accordance with the improvements as determined by an evaluation function.

Ruin-recreate Heuristics function by removing parts of a solution, then using problem-specific low-level construction heuristics to build replacements for the removed parts.

Local search Heuristics function by making minute alterations to randomly se- lected components of a solution and then accepting the changes that are at least as good as the original solution.

Crossover function by applying crossover operators to a pair of selected solutions, producing one offspring which retains aspects of both selected solutions.

3.3 Problem Domains

Maximum Satisfiability - This is a problem whereby the satisfiability (whether it can evaluate as True) of a given boolean formula needs to be determined by means of assigning any boolean value to variables within the formula.

Bin Packing - This is a problem whereby a certain number of items holding different sizes need to be placed in a finite number of bins having limited capacity.

Permutation Flowshop - This is a problem whereby a number of jobs need to be scheduled and processed on a fixed number of machines with respect to a fixed sequence or order of machines.

Personnel Scheduling - This is a problem whereby a Personnel schedule needs to be constructed given a plethora of constraints.

Travelling Salesman - This is a problem whereby a Hamiltonian circuit needs to be constructed, where cities represent vertexes, and edges are the walks between cities.

Vehicle Routing Problem - This a problem whereby the a number of closed routes are scheduled, each taken by a vehicle, to perform a sequence of tasks, such that the vehicles start and end at a depot.

3.4 Genetic Programming

3.4.1 Introduction

Genetic Programming is a class of Evolutionary Algorithms which, like other Evolutionary Algorithms, is a multi-point search which relies on the Darwinian principal of Survival of the fittest to iteratively and incrementally evolve a population of unfit programs into progressively fitter programs with respect to a given problem.

Genetic Programming is conceptually very similar to Genetic Algorithms, but there is one significant difference; Genetic Programs explore a search space composed of Programs(i.e. a Program Space), where the objective is to obtain optimal behaviours.

In order to move the search, Genetic Operators are applied to candidates that are selected by means of a selection methods which considers the individual's fitness function such that fitter individuals have a greater chance of being selected. In this way, the search is guided towards optimal areas of the Program Space.

Algorithm 2: Overview of the Genetic Program algorithm

```
  Result: a program with optimal behaviour with respect to a
          particular problem domain.
1 Create initial population;
2 repeat
3     Evaluate the fitnesses of the population;
4     Obtain parents by means of selection methods;
5     Apply Genetic Operators to parents to produce offspring;
6     Incorporate off-spring into the population;
7 until a termination criteria is met;
8 return The best program from the population
```

3.4.2 Classification

The process of automating the design of a Hyper-Heuristic will in someway need to incorporate the capacity to identify which low-level heuristic to apply given a particular solution. With the addition of one layer abstraction, where solutions in the solution space are labelled with arbitrary classes, and associate each class with predefined low-level heuristic, then the problem becomes one of classification; a function that Genetic Programs have been shown to perform well in when compared to a variety of other techniques [PSF10].

There are various representations that can be used to build classifiers with Genetic programs; namingly Arithmetic, Logical, Decision and Production Rule.

Arithmetic Trees are trees that are built with Arithmetic operators as their function set, and attributes of the data-set as their terminal set.

Logical Trees are trees that are built with Logical operators as their function set, and attributes of the data-set as their terminal set.

Decision Trees are trees where nodes represent attributes of the data-set, and edges represent value ranges for the attribute that the node from which they stem represents.

Production Rule Trees are trees that are built with condition logic, comprising of nodes that represent a logical ”If” operation. These take as arguments a condition, an action to take should the condition hold true and finally an action to take should the condition not hold. It is this kind of classifier that will be used.

4 Related Work

This section will place emphasis on 2 different proposed algorithms which both incorporate classifiers in their algorithm design.

In a paper by S. Asta and E. Özcan [AO14], a proposed solution makes use of an Apprenticeship Learning Hyper-Heuristic (ALHH) approach for the Vehicle Routing problem, and compares its performance to that of the winning algorithm of CHeSC 2011, Adaptive Hyper-Heuristic(AdapHH) as well as 2 newer approaches - the P-Hunter algorithm Ip et al. [ICXC12] and an algorithm based on Iterated Local Search (AdOr-ILS) proposed by Walker et al. [WOGB12] . The result of this experiment shows that ALHH outperforming all the competing algorithms and conclusively demonstrates the ALHH's comparative success in generalizing actions of the expert, even in for unknown problem instances.

In another paper Raras et al. [RER17] , a proposed solutions makes use of a Time Delay Neural Network (TDNN) as an apprenticeship learning hyper-heuristic adhering to a learning by demonstration approach. The result of the experiment alludes empirically to that fact that exposing richer information to the expert has the capacity to generate hyper-heuristics with improved generalising capabilities. Another paper by Sabar et al. [NGR15] proposes a dynamic multi-armed bandit-gene expression programming hyper-heuristic for combinatorial optimisation problems(GEP-HH) to automat- ically evolve the hyper-heuristic acceptance criteria. The effectiveness of the algorithm is demonstrated by evaluating its performance against the top 5 rank- ing hyper-heuristics from CHeSC on the problem domains in HyFlex where it ranks 4 out of 6; a promising result.

Whilst Gene expression programming is a variant of Genetic Programming, the approach taken in this paper is removed from the approach of using a classifier to select low-level heuristics as observed in the previous 2 papers. This presents a gap in research, whereby a Genetic Programming algorithm is used to automate the design of Selective Perturbative Hyper-Heuristics for of a classifier.

5 Methodology

5.1 Hyper-Parameters

Maximum Generation

This parameter represents the termination criteria of the Genetic Program search. When determining a maximum generation it is important not to use too few generations are the search may not have converged and consequently result in underfitting. Using a maximum generation that is to large may result in overfitting.

Training Percent

This parameter refers to the portion of the data-set to use when evaluating the quality of invdividuals. A value that is too large will result in the model overfitting and thus hinder generalisation. A value that is too small will result in underfitting.

Main Tree Max Depth

This parameter represents the maximum depth the main program tree (refer to Representation) of the Genetic Program may reach. A maximum depth that is to shallow may hinder the learning capacity of the search, whereas a maximum depth that is too deep may result in learning on noise present in the data-set.

Condition Tree Max Depth

This parameter represents the maximum depth the condition program subtrees (refer to Representation) of the Genetic Program may reach. A maximum depth that is to shallow may hinder the learning capacity of the search, whereas a maximum depth that is too deep may result in learning on noise present in the data-set.

Population Size

This parameter represents the number of individual present in each generation (refer to Control Model) and remains fixed throughout the search. The population size impacts the Genetic Diversity of a population, as more individuals can represent more points in the search space. Thus, a population size that is too small may result in premature convergence on local optima. However, a population size that is too large may be unnecessary and result a longer run-times.

Tournament Size

This parameter represents the size of a tournament (refer to Selection Method). A tournament size that is too small may approaches random selection which hinders exploitation. A tournament size that is too large results in greater elitism, and thus results in premature convergence.

Crossover Application Rate

This parameter reprents the percentage of the population that will be produced from Crossover (refer toGenetic Operators), and is proportionate to the ease of convegence.

Mutation Application Rate

This parameter reprents the percentage of the population that will be produced from Mutation (refer toGenetic Operators), and is proportionate to the difficulty of convegence.

Hoist Application Rate

This parameter reprents the percentage of the population that will be produced from Hoist (refer toGenetic Operators), and is proportionate to the difficulty of convegence.

5.2 Genetic Program Algorithm

Control Model

Generational Control Model was used to automate the design of the Selective Perturbative Hyper-Heuristic. Broadly speaking, it entails taking an initial population of candidate individuals representing points in a search space, and iteratively applying genetic operators to a produce new populations, and in this way, traverse the the search space and converging on optimal points therein.

Algorithm 3: Generational Control Model

```
Result: A Genetic Program with optimal behaviour.
1 Create Initial Population;
2 do
3     Apply Genetic Operators;
4 while Termination Criteria not met;
5 return An Optimal Program ;
```

Each iteration of the loop on line 2 is referred to as a Generation, hence the name.

Initial Population Creation

Ramped-Half-and-Half was used to create the initial population (refer to 4). This is a hybrid approach of the Full method and Grow method so as to produce a genetically diverse initial method, or more technically, represent a greater area of of the program space.

Full - this will produce a parse tree where each and every branch extends to an imposed maximum depth.

Grow - this will produce a parse tree where each branch extends at most to an imposed maximum depth.

At each depth greater than 2, equal numbers of individuals are created with that imposed depth using Full and Grow. As such, an initial population with parse trees with a wider variety of depths are produced

Representation

The Genetic Program are production-rule classifiers, whereby the tree is comprised of conditions and and actions to perform when the conditions evaluate true and false respectively.

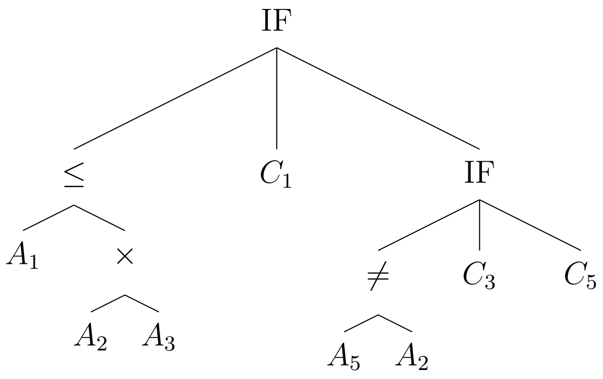

Figure 1: A simple production rule tree

The tree is implemented as a main program tree, where each *IF* has a condition sub-tree associate with it. The output of the condition sub-tree is to be interpreted as a boolean value and this boolean value determines whether the *true* branch *false* gets executed. The condition sub-tree performs relational, arithmetic, bitwise and logical operators on the values of attributes in a data-instances. It is this conditional functionality that allows for learning.

It should be noted that the main program tree and the condition sub-trees have their own set of primitives (refer to 5.2).
[1]

[1]

Definition 5.1 *Arity - The number of arguments a primitive takes.*

Definition 5.2 *Primitive - The nodes of a parse tree.*

Definition 5.3 *Terminal - A primitive that takes no arguments,i.e has an arity of 0.*

Definition 5.4 *Function - A primitive that takes arguments,i.e has an arity of at least 1.*

Main Program Tree
The sole function(refer to 5.4) of the main program tree is the *If* function. It takes in 3 arguments; a condition body to evaluate, a body of code to execute if the condition evaluates true and a body of code to execute if the condition evaluates false. The sole terminal(refer to 5.3) is the class terminal, which represents a class of the classifier.

Condition Sub-Tree
The sole terminal of the condition sub-tree is the attribute terminal, which represents the value of an attribute in the instance being classified. The function set is comprised of 4 categories of operations - relational, arithmetic, bitwise and logical - and each take 2 arguments. Relational functions include $>, <, \leq, \geq, \neq$. Arithmetic functions include $+, -, \div, \times$. Bitwise functions include $\|, \&$. Logical functions include OR, AND.

Function	**Arity**	**Tree**
IF	3	Main
$>$	2	Condition
$<$	2	Condition
$\geq$	2	Condition
$\leq$	2	Condition
$+$	2	Condition
$-$	2	Condition
$\div$	2	Condition
$\times$	2	Condition
$\|$	2	Condition
&	2	Condition
$\oplus$	2	Condition
OR	2	Condition
AND	2	Condition

(a) Functions

Terminal	**Arity**	**Tree**
Class	0	Main
Attribute	0	Condition

(b) Terminals

Figure 2: Summary of Primitives

Genetic Operators

Crossover - Crossover is a local-search operator that drives convegence by exploiting what has been explored by the search space. It accomplishes this by selecting (refer to Selection Method) a pair of candidate parents from the population, uniformly randomly selecting crossover-points on each parent, then swapping the sub-trees rooted at those points on each tree. In this wasy, genetic material is combined from both parents. To ensure that trees maintain there validity strong-typing is enforced, whereby only nodes in the main tree can be crossed over with one another, and only nodes in the condition sub-trees can be crossoved over with one another.

Mutation - Mutation is a global-search operator that drives divergence by exploring areas of the search space. It does this by selecting a sole candidate parent from the population, uniformly randomly selecting a mutation-point, then recreating the sub-tree rooted at that point. In this way, new genetic material is introduced and thereby enhancing genetic diversity of the population.

Hoist - Hoist is an operator with both explorative and exploitative qualities. It accomplishes this by selecting a parent from the population, uniformly randomly selecting a hoist-point, then making that point the root of the off-spring. In this way, it exploits the genetics of its parent, while exploring the neighbourhood thereof.

Selection Method

A selection method is a means by which to nominated candidates to apply genetic operators to - producing offspring. The selection method is a function of the the fitness (refer to Objective Function), such that fitter individuals have a greater chance of being nominated. Note that poorer candidates can be nominated, however unlikely, so as to mitigate *Elitism*, which ultimately causes premature convergence by impeding genetic diversity.

For the purpose of selection, **Tournament Selection** is used in this experiment. A number of individuals (refer to Tournament Size) are uniformly randomly and exhaustively chosen to enter a tournament. The individual with the best fitness (largest in the case of a maximisation, smallest in the case of minimisation) is the winner of the tournament, and it is used as the candidate parent.

Objective Function

The objective function is used as the ”fitness” of the individual, and it is this that the Selection Method uses to nominate candidates for partipation in Genetic Operators.

Given that the Genetic Program represent a classifier, it’s fitness should assess it’s classifying ability. For this purpose, the classifier’s **F1 Score** [She21] is used.

$$F_1 = \frac{tp}{tp + \frac{1}{2}(fp + fn)}$$

tp – true positives

fp – false positives

fn – false negatives

Figure 3: The F_1 score

5.3 Data-set

The data-set used to train the classifiers is quintessential in allowing the classifiers to make inferrences and learn about the Heuristic space, and by extension, how effective the Hyper-Heuristic will be.

Data-Engineering

The data-set is not wholely original, as it was derived from the data-set used by Raras et al. [RER17], whereby a memory of the last 8 changes in objective function where used to select one of 12 low-level heuristics as specified within HyFlex and described in a paper by Walker et al. [WOGB12]. However, their proposed algorithm (and the data-set) where specific to the HyFlex Open Vehicle Routing problem. For the purpose of producing a cross-domain Hyper-Heuristic, the data-set required to be adjusted.

One glaring problem with using the data-set used by Raras et al. [RER17] is that not all Problem Domains in the CHeSC 2011 competition have the same number of low-level heuristics, nor are the low-level heuristics necessarily the same. However, each and every problem domain has the same number of low-level heuristic categories. Thus, in order to make the data-set more generally applicable (to different problem domains), the low-level heuristics need to be mapped to their respective categories.

To further enhance the data-set, a memory buffer of the last 8 low-level heuristic selected are engineered, in line with the memory buffer of the last 8 deltas (change in objective function).

Parameter	**Category**
Two-Opt	Mutational
Or-Opt	Mutational
Shift	Mutational
Interchange	Mutational
Time-based radial ruin	Ruin and Recreate
Location-based radial ruin	Ruin and Recreate
Shift	Local Search
Interchange	Local Search
Two-opt	Local Search
GENI	Local Search
Combine	Crossover
Longest Combine	Crossover

Table 1: The 8 low-level heuristics and their Categories.

Finally, we have a dataset in which a state s_i is defined in equation 1

$$s_i = \{d_i, h_i, d_{i-1}, h_{i-1}, ..., d_{i-n}, h_{i-n}\} \tag{1}$$

Where d_i is the change in the objective function from iteration $i-1$ to i and h_i is the heuristic applied to s_{i-1} to get to state s_i.

5.4 Algorithm

The algorithm has 2 phases; the training phase and the problem-solving phase.

Training Phase

Algorithm 4: Ramped Half-and-Half Method

```
CreateInitialPopulation ($P, D_{max}$)
    inputs : Population size $P$;Max Depth $D_{max}$
    output: An Population denoted by $G$
    $G \leftarrow \emptyset$ ;
    foreach $d \in \{x \| x \in \mathbb{Z} \wedge 2 \leq x \leq D_{max}\}$ do
        $i \leftarrow 0$ ;
        for $i < \frac{1}{2}\frac{P}{D_{max}-2}$ do
            $G \leftarrow G \cup Grow(d)$ ;
            $G \leftarrow G \cup Full(d)$ ;
            $i \leftarrow i + 2$;
        end
    end
    return $G$;
```

First an initial population is created by means of the Ramped-Half-and-Half method so as to provide the search with a good starting point, by maximising the genetic diversity and thus the dispersion of points in the search space.

Thereafter, the search is moved by applying Genetic Operators to that initial population, and in that way propagating good genetics. The number of individuals in a particular generation produced by each of the Genetic Operators are determined by the respective application rates and these are user-defined hyper-parameters. The search is terminated once it hits a threshold maximum generation, at which point, the best program of the final generation is returned from the search. It is important to note that the data-set is shuffled before the start of each new generation, so as to ensure that the programs do memorise noise in the dataset.

Algorithm 5: Evolutionary Process

```
Evolution ($T, S, D$)
    inputs : Max Generation $T$;Population size
             $S$;Max Depth $D$
    output: Optimal program $P$
    $t \leftarrow 0$ ;
    $G \leftarrow CreateInitialPopulation(S, D)$ ;
    while $t < T$ do
        shuffleDataset() ;
        $G' \leftarrow \emptyset$;
        $c \leftarrow 0$;
        for $c <$ crossoverApplicationRate do
            $p_a \leftarrow$ TournamentSelection ($G$) ;
            $p_b \leftarrow$ TournamentSelection ($G$) ;
            $p'_a, p'_b \leftarrow$ Crossover ($p_a, p_b$) ;
            $G' \leftarrow G' \cup p'_a \cup p'_b$;
            $c \leftarrow c + 2$ ;
        $m \leftarrow 0$;
        for $m <$ mutationApplicationRate do
            $p \leftarrow$ TournamentSelection ($G$) ;
            $p' \leftarrow$ Mutate ($p$) ;
            $G' \leftarrow G' \cup p'$;
            $m \leftarrow m + 1$ ;
        $h \leftarrow 0$;
        for $h <$ hoistApplicationRate do
            $p \leftarrow$ TournamentSelection ($G$) ;
            $p' \leftarrow$ Hoist ($p$) ;
            $G' \leftarrow G' \cup p'$;
            $h \leftarrow h + 1$ ;
        $G \leftarrow G'$ ;
        $t \leftarrow t + 1$ ;
    $P \leftarrow BestOfGeneration(G)$ ;
    return $P$;
```

Problem-Solving Phase

Algorithm 6: GP-based Selection Hyper-Heuristic

```
1  SolveProblem ($\rho, \varsigma, \omega, \phi, \delta$)
       inputs : Problems $\rho$; Time Limit $\varsigma$; Max Generation $\omega$;
                Population size $\phi$; Max Depth $\delta$
       output: Optimal Solutions $\chi$
       /* Acquire an optimal Program $H$                                */
2      $P \leftarrow$ Evolution($\omega, \phi, \delta$) ;
       /* Start with an empry set of Solutions for $\rho$                 */
3      $\chi \leftarrow \emptyset$;
4      foreach $\rho_i \in \rho$ do
5          initialiseSolutionInMemory ($\chi_i$);
6          loadProblem($\rho_i$) ;
7          $O \leftarrow \emptyset$ ;
8          $C \leftarrow \emptyset$ ;
           /* Start time                                                 */
9          $l \leftarrow 0$ ;
10         do
11             $O \leftarrow O \cup$ getObjectiveValue($\rho_i$);
12             $c \leftarrow$ getHeuristicCategory($P, O, C$) ;
13             $h \leftarrow$ getRandomLowLevelHeuristicFromCategory($c$);
14             Move acceptance: All-Moves;
15             applyLowLevelHeuristic($h$);
16             $C \leftarrow C \cup c$ ;
17         while currentTime()$-l < \varsigma$;
18         $\rho \leftarrow \rho \cup \rho_i$;
19     end
20     return $\chi$;
```

From Algorithm 1, it can be seen that the evolutionary process is used to automate the design of a Selection Perturbative Hyper-Heuristic, and the manner in which the Genetic Program uses aspects of the fitness landscape and memory of applied heuristics to determine which heuristic to apply next. Furthermore, it is important to note that the **Move Acceptance** policy is to accept all the low-level heuristics.
An implementation detail to consider is that the random generator used to pick a random low-level heuristic from a category is seeded, and thus results are repeatable.

5.5 Experimental Setup

Training and Testing for Convergence

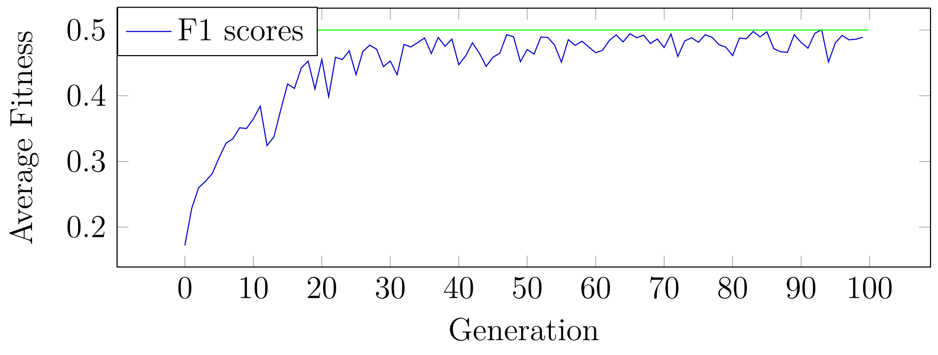

Figure 4: Graph showing the average score of each generation

From the graph, it is clear that the algorithm converged as after about generation 40 the graph plateaus.

Parameter Values

The parameter values were determined empirically.

Parameter	**Value**
Maximum Generation	100
Training Percent	0.7
Main Tree Max Depth	10
Condition Tree Max Depth	5
Population Size	2000
Tournament Size	4
Crossover Application Rate	0.5
Mutation Application Rate	0.475
Hoist Application Rate	0.5

Table 2: Parameter values used in the experiment.

6 Discussion

7 Conclusion

References

[AO14] Shahriar Asta and Ender Ozcan. *An Apprenticeship Learning Hyper-Heuristic for Vehicle Routing in HyFlex.* IEEE, 2014.

[CDF$^+$12] T. Cichowics, M. Drozdowski, M. Frankiewicz, G. Pawlak, F. Rytwinski, and J. Wasilewski. Hyper-heuristics for cross-domain search. *Bulletin of the Polish Academy of Sciences*, pages 801–802, 2012.

[CKS00] P. Cowling, G. Kendall, and E. Soubeiga. A hyperheuristic approach for scheduling a sales summit. *Selected Papers of the Third International Conference on the Practice And Theory of Automated Timetabling(PATAT 2000)*, pages 176–190, 2000.

[EMM$^+$13] Burke E., Gendreau M., Hyde M., Kendall G., Ochoa G., and Özcan E. Hyper-heuristics: A survey of the state of the art. *Journal of Operational Research Society 64*, pages 1695–1724, 2013.

[GM11] Ochoa G and Hyde M. The cross-domain heuristic search challenge, 2011. Accessed: 2021-10-30.

[ICXC12] W. Ip, C. Chan, F. Xue, and C. Cheung. *A Hyper-Heuristic inspired by pearl hunting.* Springer, 2012.

[NGR15] Sabar N.R., Kendall G., and Qu R. A dynamic multi-armed bandit-gene expression programming hyper-heuristic for combinatorial optimization problems. *IEEE Transactions on Cybernetics*, pages 217–228, 2015.

[NP18] Rong Qu Nelishia Pillay. *Hyper-Heuristics: Theory and Applications.* Springer, Gewerbestrasse 11, 6330 Cham, Switzerland, 2018.

[OGH$^+$12] Ochoa, G., Hyde, M., Curtois, T., Vazquez-Rodriguez, J.A., Walker, J., Gendreau, M., Kendall, G., McCollum, B., Parkes, A.J., Petrovic, S., Burke, and E.K. HyFlex: A Benchmark Framework for Cross-domain Heuristic Search. In J.-K. Hao and M. Middendorf, editors, *European Conference on Evolutionary Computation in Combinatorial Optimisation(EvoCOP 2012)*, volume 7245 of *LNCS*, pages 136–147, Heidelberg, 2012. Springer.

[PSF10] Espejo P.G., Ventura S., and Herrera F. A survey on the application of genetic programming to classification. *IEEE Transactions on Systems, Man, and Cynbernetics, Part C (Applications and Reviews)*, pages 121–144, 2010.

[RER17] Tyasnurita Raras, Ozcan Ender, and John Robert. Learning heuristic selection using a time delay neural network for open vehicle routing. *IEEE Congress on Evolutionary Computation 2017*, pages 1–7, 2017.

[She21] V. Sheela. What is the f1-score and what is its importance to machine learning?, 2021. Accessed: 2021-11-01.

[WOGB12] D. Walker, G. Ochoa, M. Gendreau, and E. K. Burke. Vehicle routing and adaptive iterated local search within the hyflex hyper-heuristic framework. *LION*, pages 265–276, 2012.