UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

COS700 Research Report

Write your Project Title here

Student number: u12345678

Supervisor(s):
Supervisor1
Supervisor2

John Doe

October 2021

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 proposes a project that will encompass a study of existing algorithms, with a focus on those which represent the task of automating the design of hyper-heuristics as a classification one. Furthermore, the project will 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 Purpose of the Study

1.2 Objectives

1.3 Layout

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 the CHeSC challenge 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. The particular domain to be tested on will be the Vehicle Routing problem.

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

1. Design and implement the proposed Genetic Program.
2. Evaluate and compare the performance of the Genetic Program to that of the top performing algorithms on the HyFlex framwork.

3 Methodology

The proposed project will follow the proof of demonstration methodology [**?**]. This entails the construction of an initial system, which will then by

iteratively refined based on its performance until a point where no significant improvements can be made to the algorithm. To validate the results for comparison, statistics tests will be utilised to ascertain statistical significance.

The project will entail 3 main stages:

3.1 Creating the Testing Environment

The first stage will involve creating an environment that mimics the HyFlex vehicle routing domain, which is the time window variant of the vehicle routing problem. The problem instances originate from two sources: the Solomon data set and the Gehring and Homberger data set. The data sets comprise of 100 customer and 1000 customer problems resepectively, and both data sets have 3 instance types; Random - customers' locations are uniformly randomly determined, Clustered - the customers' location cluster in groups, Clustered Random - combines Random and Clustered customers' locations.

3.2 Constructing a GP algorithm

The second stage will involve constructing a novel algorithm for the Automated Design of a Cross-Domain Selective Perturbative Hyper-Heurisitic by means of a Genetic Program, with the intent of harnessing the classifying power of Genetic Programming.

3.3 Evaluation and Refinement

The final stage will involve evaluating the performance of the algorithm on the aforementioned Solomon dataset as well as the Gehring and Homberger dataset. This will be done by testing it on the Time Window variant of the Vehicle Routing Problem, and it will be compared to previous work from CHeSC (2011) as well as the additional algorithms studied in the literature survey. If it does not perform favourably, it will require refinement.This could be in the form of tuning the Genetic Program's parameters, redesigning the Genetic Operators and introducing new or different Primitives.

The Genetic Program will be run on a 4 core intel i7 processor with 8 gigabytes of Memory.

4 Background

4.1 Hyper-Heuristic

4.1.1 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[**?**]. 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[**?**]. 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[**?**]. 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.

4.1.2 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(the heuristics constituting the heuristic space) in one of two ways; selection or generation [**?**]. 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.

4.1.3 Selective Perturbative Hyper-Heuristics

For a particular problem, there are a number of pre-defined low-level perturbative heuristics 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 **??** describes how selective perturbative hyper-heurstics work.

[H] an optimal solution to a particular problem domain. Create initial solution solutions have improved Select a low-level perturbative heuristic to apply Apply low-level heuristic to solution The best solution How Selective Perturbative Hyper-Heuristics work

4.2 Cross-Domain Hyper-Heuristics

4.2.1 Introduction

It should be noted that, whilst Hyper-Heuristics do offer some generalising capacity from heuristics, they are still specific to a particular problem domain. Cross-Domain Hyper-Heuristics aim to extend the capacity for generalisation, by widening the scope of a hyper-heuristic to multiple domains.[**?**]

4.2.2 Cross-Domain Heuristic Search Challenge

The Cross-Domain Heuristic Search Challenge (CHeSC 2011) was a chal-

lenge 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[**?**].

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[**?**].

4.2.3 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 for the aforementioned problems, which include Mutational heuristics, Ruin-recreate heuristics, Local search heuristics and Crossover[**?**].

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 selected 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.

4.3 Problem Domains

4.3.1 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.

4.3.2 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.

4.3.3 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.

4.3.4 Personnel Scheduling

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

4.3.5 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.

4.3.6 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.

4.4 Vehicle Routing Problem

Definition **??** formally describes the essence of the Vehicle Routing Problem. Vehicle Routing Problem is a discrete combinatorial optimization problem which tries to optimise m-many closed routes each taken by m-many vehicles to visit n - 1 customers in dispersed in a network modelled by graph G, where G = (V,A). V is a set of n vertices; of which 1 vertex represents a depot, and n - 1 vertices represent customers. A represents edges between the vertices of V, where each edge has an associated weight representing a distance or cost.

4.5 Genetic Programming

4.5.1 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 **??** provides an overview of the Genetic Program algorithm

[H] a program with optimal behaviour with respect to a particular problem domain. Create initial population a termination criteria is met Evaluate the fitnesses of the population Obtain parents by means of selection methods Apply Genetic Operators to parents to produce offspring Incorporate offspring into the population in accordance with the Control Model; The best program from the population Overview of the Genetic Program algorithm

4.5.2 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[**?**].

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.

5 Related Work

Various approaches have been proposed in literature for the problem domains in HyFlex, and this section will place emphasis on the Vehicle Routing Problem for 2 different proposed algorithms which both incorporate classifiers in their algorithm design.

In one paper, 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 [**?**] and an algorithm based on Iterated Local Search (AdOr-ILS)[**?**]. 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, 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.

[**?**] proposes a dynamic multi-armed bandit-gene expression programming hyper-heuristic for combinatorial optimisation problems(GEP-HH) to automatically evolve the hyper-heuristic acceptance criteria. The effectiveness of the algorithm is demonstrated by evaluating its performance against the top 5 ranking 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.

6 Discussion

Discuss here all your work and present your results. This can be spread over multiple sections.

7 Conclusion

Summarise here your problem, your contribution and your results. Also indicate possible future extensions to your work [**?**].