Improving Selection Hyper-Heuristics Based on Analyses of their Search Behaviour using HyVis

Interim Report

Jack Cakebread1

Supervised by Ender Özcan

School of Computer Science
University of Nottingham

December 2021

Acknowledgements

The development of HyVis was initially undertaken by Warren Jackson, a Research Fellow at the University of Nottingham, before turning into a dissertation project.

At the time of taking over the project, the data collection and post-processing tools, in their initial state, had been implemented. Development of displaying a progress plot for collected data was also started.

JavaDocs have since been created for each Java class, with Warren being credited as the first author on each class he created. A fork of the original GitHub repository was created at the start of the project, with the changes and additions to the code base being detailed in each commit, and the original code base before undertaking the project viewable.

I would like to thank Warren Jackson for his contributions to the project, as well as giving me access to the machines needed to run the hyper-heuristic trials and helping with the handover of the project.

I would also like to thank my project supervisor, Dr. Ender Özcan, for his continuous feedback, guidance, and support throughout the project.

Contents

Introduction

1.1 Background and Motivation

Over the years, heuristic algorithms have been used to find solutions to a variety of computationally complex real-world optimization problems. A heuristic is an algorithm that gives a solution to a problem within a reasonable time frame. It may not be the optimal solution to a problem, but it is achieved much faster than an exhaustive method, which must search the whole solution space to say with certainty what the optimal solution to a problem is.

Hyper-heuristics are one of the more recent innovations in the field [5]. The key concept of a hyper-heuristic is that it operates on a set of low-level heuristics, which are able to be changed depending on the problem which is being solved. These low-level heuristics operate directly on the solution. An objective function, usually being minimised, indicates the quality of the solution, and the objective value returned by it is passed back to the hyper-heuristic after the low-level heuristic is applied to the problem.

The most common hyper-heuristic framework is single point based search, in which a single solution is initialised and updated through the run time of the hyper-heuristic. Once a stopping condition is reached, the final solution is returned. An iterative selection hyper-heuristic (Figure 1.1) applies a chosen low-level heuristic to the current solution at each step of a search, before deciding whether to accept or reject the newly created solution [7]. A generation hyper-heuristic generates new heuristics from components of existing heuristics [4]. A real-world example of their use is a Reinforcement Learning hyper-heuristic for the optimisation of flight connections [11].

In order to analyse the performance of a heuristic, a variety of different visualisations can be used. Progress plots show how the objective value of the current solution changes over the run-time of a heuristic, charts can be used to display utilization rates with hyper-heuristics (how often a hyper-heuristic selects a given low-level heuristic throughout its execution), and evolutionary activity plots are used with multi-memetic algorithms to show the accumulation of meme concentration up until a given generation [9].

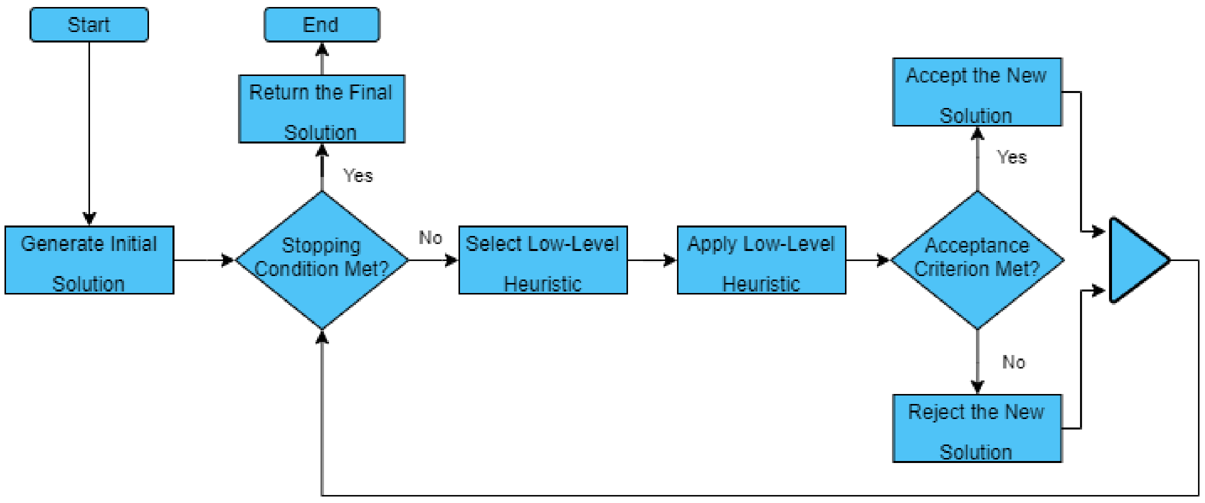

Start
End
Return the Final Solution
Yes
Accept the New Solution
Yes
Generate Initial Solution
Stopping Condition Met?
No
Select Low-Level Heuristic
Apply Low-Level Heuristic
Acceptance Criterion Met?
No
Reject the New Solution

Figure 1.1: Flowchart showing how an iterative selection hyper-heuristic functions

HyFlex is a flexible framework for the implementation and analysis of hyper-heuristics across different domains [2]. It was initially designed for the rapid development of selection hyper-heuristics and there are currently nine problem domains (0-1 Knap Sack, Bin Packing, Flow Shop Scheduling, Max-Cut, MAX-SAT, Personnel Scheduling, Quadratic Assignment, Travelling Salesman Problem, Vehicle Routing Problem) which are made up of several problem instances, a set of low-level heuristics that are applicable to the respective domain and the initialisation method for the solutions [2].

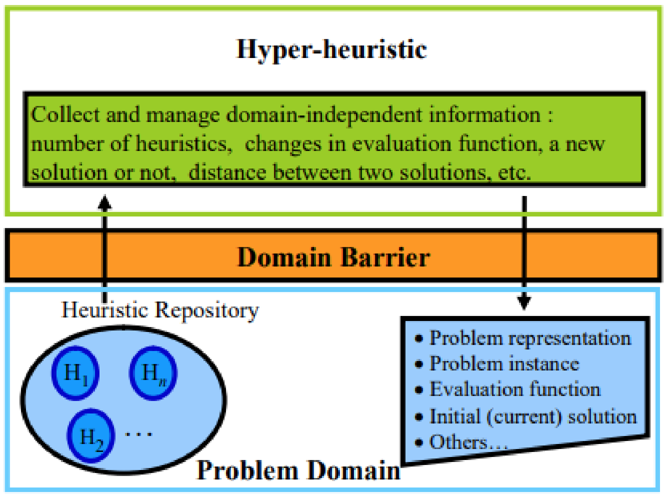

Hyper-heuristic
Collect and manage domain-independent information : number of heuristics, changes in evaluation function, a new solution or not, distance between two solutions, etc.
Domain Barrier
Heuristic Repository
H1
Hn
H2
...
• Problem representation
• Problem instance
• Evaluation function
• Initial (current) solution
• Others...
Problem Domain

Figure 1.2: Hyper-heuristic conceptual framework featuring the domain barrier [3].

HyFlex makes it simple to apply a hyper-heuristic to a problem domain and provides a layer of abstraction (Figure 1.2), in the form of a software interface, between the hyper-heuristic and problem domain layers. It enables the created hyper-heuristic to be applied without it having knowledge of the problem domain itself, with the hyper-heuristic only knowing the objective value returned by the objective function and the set of low-level heuristics it has available. Furthermore, it also removes the burden on the user of designing and implementing a problem domain and enables the cross-domain usage of the hyper-heuristic, as shown in the CHeSC (Cross-domain Heuristic Search Challenge) competition [3].

However, analysis of the performance of the hyper-heuristics, in the form of visualisations, is left to the user as HyFlex does not contain any visualisation tools. Visualisation tools will enable the user to easily make informed decisions on any modifications made to their hyper-heuristics, in order to increase their performance. They will also help the user to understand the inner workings of the hyper-heuristic and enable them to be graphically compared to other hyper-heuristics based on different components.

1.2 Aims and Objectives

The aim of this research project is to compare and improve selection hyper-heuristics based on analysis of their search behaviour, with the aid of graphs and visualisations. In order to do this, HyVis, a visualisation tool extending HyFlex, will be developed to enable the filtering and visualisation of hyper-heuristic components. The end goal of HyVis is to observe the search behaviour of the hyper-heuristics and to discover hidden patterns regarding decisions made by each hyper-heuristic component. HyVis will serve as a system to build systems enabling the construction (generation) of new, effective, and efficient search methods through the observations by experts or automatically, using collected filtered data and the application of machine learning methods.

The key objectives consist of:

- Collecting raw data at every step as a hyper-heuristic is being executed on an instance of a problem domain, including but not limited to:
  - The selected low-level heuristic in the heuristic selection phase of the hyper-heuristic

  - The time taken to execute the selected heuristic on the current solution
  - The objective value from the new solution generated from applying the heuristic
  - The accept/reject decision in the move acceptance phase of the hyper-heuristic.

- Generating a variety of graphs/plots to enable the visualisation and filtering of the collected data from the running of hyper-heuristics.
- Creating innovative visualisations for hyper-heuristics, building on research from other heuristic topics, and displaying information that is not available from current visualisations.
- Implementing at least two chosen selection hyper-heuristics and collecting raw data from them for analysis with HyVis.
- Analysing the visualisations generated by HyVis and using the information learned to adjust and improve the performance of the hyper-heuristics.
- Designing an intuitive GUI, which takes into consideration certain disabilities such as colour-blindness, enabling the tool to be accessible to a wide range of people.

Related Work

This section will discuss related visualisations that are relevant to the project.

2.1 Visualisation Methods

2.1.1 Progress Plots

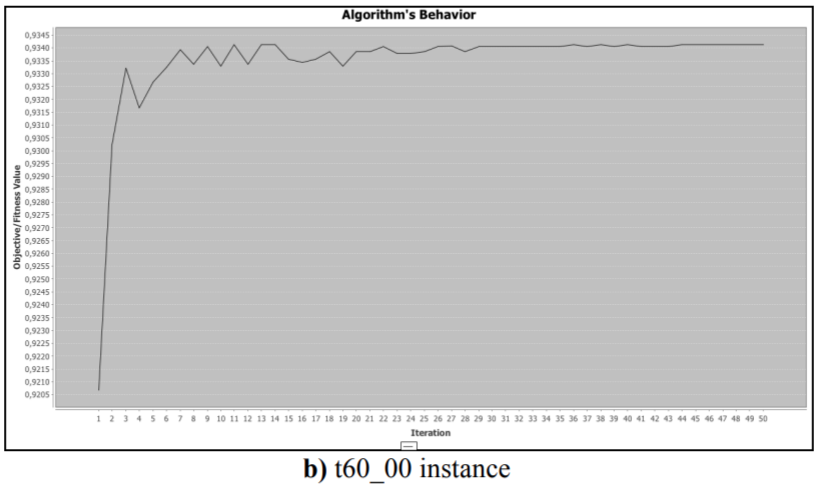


Figure 2.1: Progress plot using VisTHAA [6].

Progress plots show how the objective value of the current solution changes over the run time of a hyper-heuristic. As the name suggests, they are useful to show how the hyper-heuristic is progressing over time in finding a better solution. They can also provide insight into the move acceptance method of the hyper-heuristic and its effectiveness of balancing exploration of the search space with the exploitation of the areas of the search space which represent promising solutions.

2.1.2 Evolutionary Activity Plots

An evolutionary activity plot indicates how the evolutionary activity of a meme changes over generations in a memetic algorithm (Figure 2.2). A memetic algorithm is a population-based approach, keeping track of a population of individuals (solutions) for each generation (iteration). A meme encodes into an individual of the population information such as how, where, when, and to what degree to change an individual in order to generate novel individuals for the population.

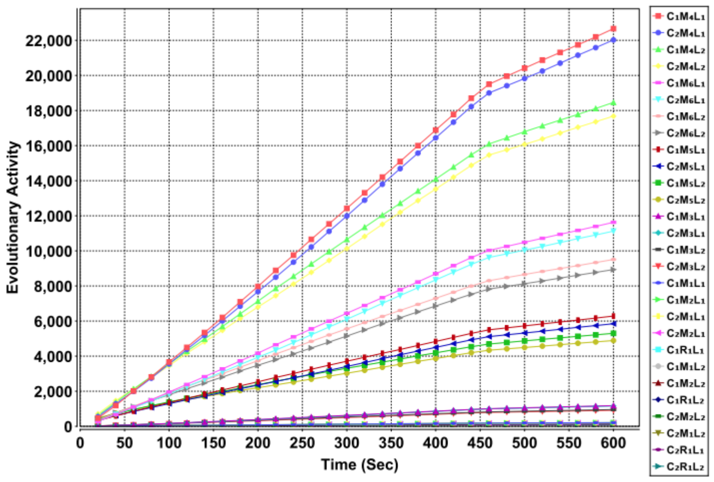


Figure 2.2: Evolutionary activity plot of a multi-memetic algorithm [13]

Defined by Krasnogor, the concentration of a meme is ”the number of individuals in the population that carry the meme” and the evolutionary activity of a meme is the accumulation of meme concentration until a given generation [9].

Although evolutionary activity plots are not used for the visualisation of hyper-heuristics, the key concepts and ideas behind them are transferable to the field. A new visualisation, inspired by evolutionary activity plots, will be proposed in this project.

2.1.3 Heuristic Utilization Rate Chart

A heuristic utilization rate chart shows how often each low-level heuristic is selected during the run-time of a hyper-heuristic. These charts are a useful way of visualising which low-level heuristics were favoured by the hyper-heuristic and enable the comparison of the popularity of each low-level heuristic across multiple instances of a problem domain.

2.1.4 Heuristic Acceptance Rate Chart

A heuristic acceptance rate chart shows how often the application of each low-level heuristic leads to an accepted move during the run-time of a hyper-heuristic. These charts are a useful way of visualising how effective the application of each heuristic was, and can be used for the comparison of the effectiveness of each low-level heuristic across different instances of a problem domain.

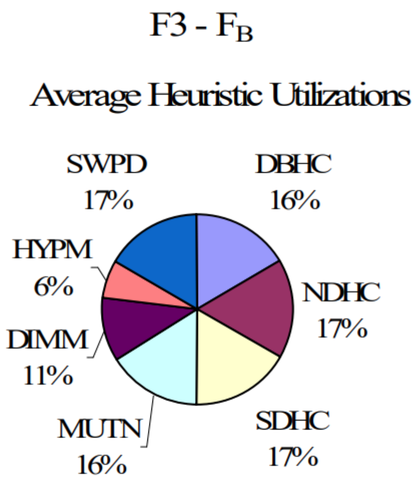


Figure 2.3: Pie chart showing the average heuristic utilization rates of each low-level heuristic during the run-time of a hyper-heuristic [12].

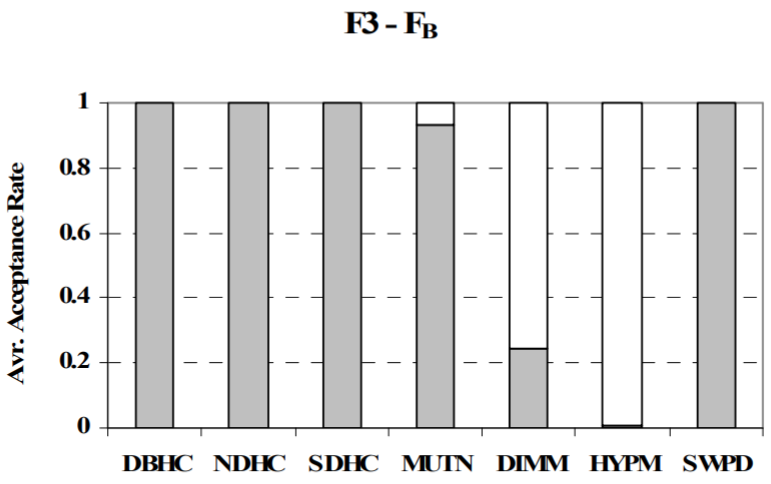


Figure 2.4: Bar chart showing the average acceptance rates of each low-level heuristic during the run-time of a hyper-heuristic [12].

Methodology

3.1 Cross-domain Heuristic Search Challenge

In this project, the hyper-heuristics being analysed will be run in the same way as the competing hyper-heuristics were during the Cross-domain Heuristic Search Challenge (CHeSC) competition [3]. That is, running the hyper-heuristics across 5 instances of each of the 6 problem domains chosen for the CHeSC competition. For each instance, 31 trials will be run, and the average performance of the hyper-heuristics across those trials will be used to evaluate their performance on the instance. Each trial will be run for the equivalent of 10 minutes on the CHeSC competition machine, with the benchmarking tool provided for the competition being used to calculate how long the hyper-heuristics need to be run on a machine in order to achieve this.

The 6 HyFlex problem domains that were used in the CHeSC competition were Bin Packing, FlowShop, Personnel Scheduling, SAT (Boolean Satisfiability Problem), TSP (Travelling Salesperson Problem), and Vehicle Routing.

The motivation behind running the hyper-heuristics on the CHeSC instances is due to the CHeSC competition being an established and extensive way of comparing HyFlex implemented hyper-heuristics, and there is no need to reinvent the wheel in this case. This also enables the performance of each hyper-heuristic being analysed to be compared to the hyper-heuristics that competed in the CHeSC competition.

3.2 Hyper-Heuristics

This section will detail the hyper-heuristics which will be compared in this project and explain the motivation behind selecting them.

3.2.1 Simple Random Improving or Equal Acceptance

The simple random improving or equal acceptance hyper-heuristic randomly selects a low-level heuristic to be applied to the current solution at each iteration and accepts the move if the candidate solution generated from the application is an improvement on the current solution or equal to the current solution.

This hyper-heuristic has been chosen as it is very basic and simple to understand. The move acceptance method should ensure that the solution never gets worse, but it may lead to a stagnation in the progress of the hyper-heuristic, due to not considering worsening solutions at all. This can lead to the hyper-heuristic not being able to explore the solution space as extensively as it could and result in it getting stuck at a local minimum. Therefore, this should act as a baseline hyper-heuristic for evaluating heuristics with more complex heuristic selection and move acceptance methods.

3.2.2 Simple Random Naive Acceptance

Similar to the simple random improving or equal acceptance hyper-heuristic, the simple random naive acceptance hyper-heuristic also selects a low-level heuristic at random during the heuristic selection stage and accepts moves that result in improving or equal solutions. However, when a worsening solution is encountered, there is a 50% chance that it is accepted.

This hyper-heuristic has been chosen as it is building on the idea of the simple random improving or equal acceptance hyper-heuristic, with additional complexity in the move acceptance method. Accepting worsening solutions will lead to a more diverse search of the solution space for the global minima, and can stop the hyper-heuristic from being stuck at a local minimum. However, it may also hinder the exploitation of a promising area of the solution space, instead searching another area with less promising solutions. Therefore, HyVis will be a good medium for evaluating the effectiveness of this added layer of complexity.

3.2.3 Lean-GIHH

LeanGIHH [1] is an adaptation of the Generic Intelligent Hyper-heuristic (GIHH), which aims to streamline the hyper-heuristic by removing unnecessary complexity without affecting its performance. GIHH (originally Adap-HH) [10] is the CHeSC competition winner, performing the best in the cross-domain search compared to all the other hyper-heuristics that were entered. GIHH is considered to be a state-of-the-art HyFlex implemented hyper-heuristic, with it being far more complex than both the simple random hyper-heuristics discussed. It makes use of methods such as excluding poorly performing heuristics, applying heuristics based on how well they have performed previously, and restarting the search (re-initialising the current solution) if a condition is met. A full, in-depth explanation is available in [10] .

This hyper-heuristic has been chosen due to its impressive cross-domain performance in the CHeSC competition. As the hyper-heuristics will be compared based on their performance on the CHeSC instances, it makes sense to include the winner in the analyses. Additionally, using HyVis to analyse a complex hyper-heuristic, which may be hard to trace or follow, will give the user a deeper insight into how the hyper-heuristic functions.

3.2.4 Modified Choice Function All Moves Acceptance

The Modified Choice Function All Moves Acceptance (MCFAM) hyper-heuristic [8] uses a Modified Choice Function for heuristic selection, and accepts all solutions that are generated by the application of the chosen low-level heuristics. As the name suggests, the Modified Choice Function is an adaptation of the Choice Function. The motivation behind the creation of a Modified Choice Function was due to the poor cross-domain performance of the Choice Function.

The Modified Choice Function selects which low-level heuristic to apply based on three performance metrics: the performance of the low-level heuristics in previous iterations, the success of heuristics when they are applied consecutively, and how much time has elapsed since the last time the low-level heuristic had been selected. One of the modifications it makes to the Choice Function is the use of crossover heuristics, which take in two solutions (the current solution and a randomly initialised solution, for example) and returns a new solution based on a combination of the parent solutions. A key difference between the original and Modified Choice Function is that the Modified Choice Function focuses on the exploitation of a promising solution over the exploration of the search space. A full, in-depth explanation is available in [8].

The motivation behind choosing this hyper-heuristic was in part due to its impressive cross-domain performance when tested against the CHeSC competitors. Additionally, as the moves are automatically accepted regardless of the change in the quality of the current solution, the intelligence of the hyper-heuristic is concentrated in the heuristic selection method. Although all moves are accepted, worsening moves are penalised by the Modified Choice Function, reducing the likelihood of selecting the heuristic which led to the worsening move in future iterations. HyVis will offer a deeper insight into the functioning of the Modified Choice Function, and how it makes decisions during the run-time of the hyper-heuristic.

3.3 Visualisations

This section details the visualisations which HyVis will be able to generate and explains the motivation behind each one.

3.3.1 Progress Plots

Progress plots, as detailed in subsection 2.1.1, will be used to visualise how the objective value of the current solution changes over the run-time of a hyper-heuristic.

The motivation behind choosing this type of visualisation is due to it offering an easy way to crudely assess how well a hyper-heuristic is performing over time. It enables the objective value of the current solution at the beginning of the execution of the hyper-heuristic to be easily compared to the objective value at the end of the execution of the hyper-heuristic. It also enables the user to visualise the acceptance of improving and worsening solutions. Additionally, it makes it simple to compare the performance of different hyper-heuristics on the same problem instance.

3.3.2 Average Heuristic Utilization Rate Charts

Heuristic utilization rate charts, as detailed in subsection 2.1.3, will be used to visualise the average utilization rate of each low-level heuristic during the run-time of a hyper-heuristic on a problem instance.

The motivation behind selecting this type of visualisation is due to it offering a clear and compact way of visualising the most popular heuristics chosen during the run-time of a hyper-heuristic. Conversely, this can also give information on which low-level heuristics were avoided during the run-time of the hyper-heuristic and can be used to compare how different hyper-heuristics utilize different low-level heuristics across instances of a problem domain.

3.3.3 Average Heuristic Acceptance Rate Charts

Heuristic acceptance rate charts, as detailed in subsection 2.1.4, shows how often the application of each selected low-level heuristic leads to an accepted move.

The motivation behind choosing this type of visualisation is due to it offering a side-by-side comparison of the average acceptance rate for each low-level heuristic on a problem instance. This enables it to be easily seen which low-level heuristics were the most effective at generating an acceptable solution. Additionally, combined with the information from a heuristic utilization rate chart, if it is seen that a heuristic has a very high utilization rate and a very low acceptance rate, it may indicate that the heuristic selection method is performing poorly in progressing the search and improvements to the hyper-heuristic can be made based on this information.

3.3.4 Average Heuristic Improvement-Time Plots

An average heuristic improvement-time plot is a novel visualisation proposed in this project. An average heuristic improvement-time plot displays the average improvement rate of a low-level heuristic against the average application time of the low-level heuristic, in the form of a scatter plot. The improvement rate of a low-level heuristic is the rate at which the application of the heuristic results in an improving solution. The application time of a low-level heuristic is the time taken to apply the heuristic to the solution.

The motivation behind choosing this visualisation is to give a deeper insight into the application of the low-level heuristics. Although utilization rate and acceptance rate charts are useful to visualise how each low-level heuristic is used and how successful they were at generating an acceptable solution, respectively, they do not offer insight into whether or not the solution was actually improved. For example, a naive move acceptance mechanism has a 50% chance of selecting a worsening solution, which means that the acceptance rate on its own can be a deceiving measure of how effective a hill-climbing low-level heuristic is.

Therefore, an average heuristic improvement-time plot can be used in conjunction with a utilization rate and acceptance rate chart to enable the user to make an informed analysis of the performance of a low-level heuristic. For example, with just a utilization rate and acceptance rate chart, a high utilization and acceptance rate for a given hill-climbing heuristic may lead the user to conclude that the heuristic is performing well in advancing the search towards a global minimum. However, the improvement-time plot may reveal that the heuristic has a very low average improvement rate and a long average application time, meaning the low-level heuristic is not as effective as initially thought.

3.3.5 Heuristic Stability Plots

A heuristic stability plot is a novel visualisation that is proposed in this project. Building on the concepts of an evolutionary activity plot, as detailed in subsection 2.1.2, a heuristic stability plot displays the cumulative selection frequency of each low-level heuristic until a given time. Similar to the concept of evolutionary activity, a heuristic stability plot aims to visualise the activity of the low-level heuristics over the run-time of a hyper-heuristic.

The motivation behind choosing this visualisation is to provide a simple and informative way for the user to examine which low-level heuristics were chosen by the hyper-heuristic, and how the choice of which low-level heuristic to apply changes over time. This can be useful to gain a greater understanding of how the heuristic selection method is working in practice and can be used in conjunction with other visualisation methods to make informed improvements to the selection method of the hyper-heuristic.

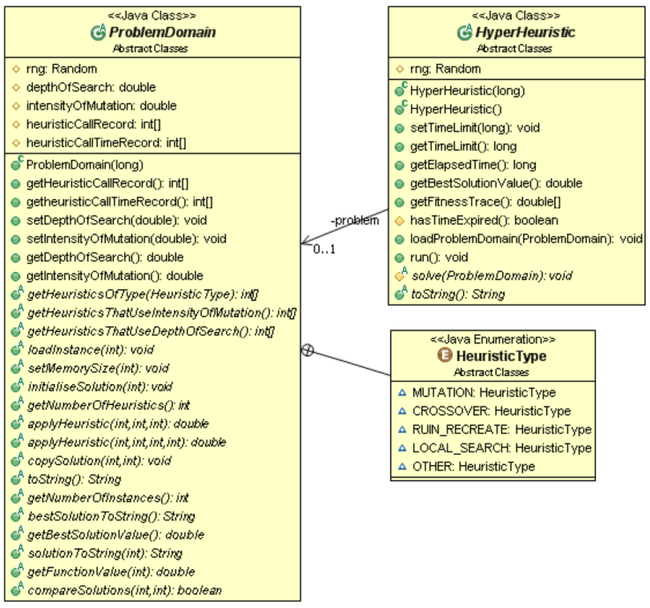
Design

4.1 HyVis

HyVis will be made up of 5 main components: the data collection system, post-processing system, filtering system, visualisation system, and graphical user interface.

4.1.1 Data Collection System

The data collection system is tasked with collecting data on the running of a hyper-heuristic. The goal of the data collection system is to collect as much data as possible about the running of the hyper-heuristic without needing to modify its code, enabling HyVis to be used with any existing hyper-heuristic designed for HyFlex.



<<Java Class>>
ProblemDomain
Abstract Classes
rng: Random
depthOfSearch: double
intensityOfMutation: double
heuristicCallRecord: int[]
heuristicCallTimeRecord: int[]
ProblemDomain(long)
getHeuristicCallRecord(): int[]
getheuristicCallTimeRecord(): int[]
setDepthOfSearch(double): void
setIntensityOfMutation(double): void
getDepthOfSearch(): double
getIntensityOfMutation(): double
getHeuristicsOfType(HeuristicType): int[]
getHeuristicsThatUseIntensityOfMutation(): int[]
getHeuristicsThatUseDepthOfSearch(): int[]
loadInstance(int): void
setMemorySize(int): void
initialiseSolution(int): void
getNumberOfHeuristics(): int
applyHeuristic(int,int,int): double
applyHeuristic(int,int,int,int): double
copySolution(int,int): void
toString(): String
getNumberOfInstances(): int
bestSolutionToString(): String
getBestSolutionValue(): double
solutionToString(int): String
getFunctionValue(int): double
compareSolutions(int,int): boolean
-problem
0..1
<<Java Class>>
HyperHeuristic
Abstract Classes
rng: Random
HyperHeuristic(long)
HyperHeuristic()
setTimeLimit(long): void
getTimeLimit(): long
getElapsedTime(): long
getBestSolutionValue(): double
getFitnessTrace(): double[]
hasTimeExpired(): boolean
loadProblemDomain(ProblemDomain): void
run(): void
solve(ProblemDomain): void
toString(): String
<<Java Enumeration>>
HeuristicType
Abstract Classes
MUTATION: HeuristicType
CROSSOVER: HeuristicType
RUIN_RECREATE: HeuristicType
LOCAL_SEARCH: HeuristicType
OTHER: HeuristicType

Figure 4.1: Class diagram for the HyFlex framework [2].

Figure 4.1 shows a class diagram for the HyFlex framework. To create a HyFlex compatible hyper-heuristic, all the user has to do is extend the HyperHeuristic abstract class and override the *solve* and *toString* methods. A runner class can then be used to create an instance of the hyper-heuristic and call its methods to set how long it should run for, load the problem domain and instance it is operating on, and then run the hyper-heuristic. To evaluate its progress after it has been run, *getBestSolutionValue* can be called, which will return the objective value of the best solution found. The runner class has a very limited amount of intelligence of the execution of the hyper-heuristic, therefore, data collection is not possible within the class.

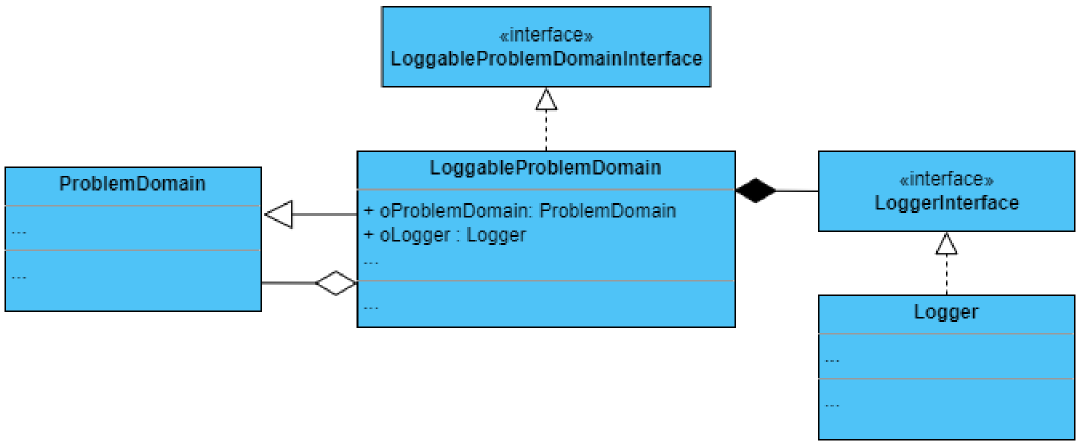


Figure 4.2: Class diagram showing how the decorator pattern is used with LoggableProblemDomain.

The proposed solution to this problem is using the decorator pattern to wrap a ProblemDomain object and extend its functionality (Figure 4.2). The LoggableProblemDomain class extends the ProblemDomain class, in order for its objects to be used in place of ProblemDomain objects. Aggregation is also used to enable the class to take an existing ProblemDomain object in its constructor, store it as a class member, and use it as required.

Additionally, LoggableProblemDomain will make use of composition, using a Logger object to log the collected data at each step of the hyper-heuristic's execution. LoggableProblemDomain will override the required methods in the ProblemDomain class in order to add the logging functionality and will preserve the original functionality of the methods.

The collected data will be stored in a .csv file, with each line representing a step in the hyper-heuristic. The steps that will be recorded are the initialisation of a solution, the copying of a solution from one memory index to another, and the application of a low-level heuristic on the current solution.

4.1.2 Post-Processing System

After the data has been collected, it must be processed to scale the execution time of the trials to 10 minutes, in order to match the CHeSC competition time.

Additionally, as the data will be collected from the LoggableProblemDomain class, it will not be able to directly log whether or not a move was accepted or rejected. This is due to the move acceptance stage being inside the solve method of the hyper-heuristic, and the LoggableProblemDomain object has no knowledge of the hyper-heuristics code. However, it does have knowledge of the memory indices of the current and candidate solutions for each step, and we can work out whether or not the solution was accepted or rejected based on how they change as the hyper-heuristic is run.

When a move is accepted, the candidate solution replaces the current solution. One way of replacing the current solution with the candidate solution is to simply copy the solution stored at the candidate solution memory location into the current solution memory location, overwriting the current solution. A more efficient way of doing it is to swap the candidate and current solution memory indices, which avoids having to copy the actual solution representation.

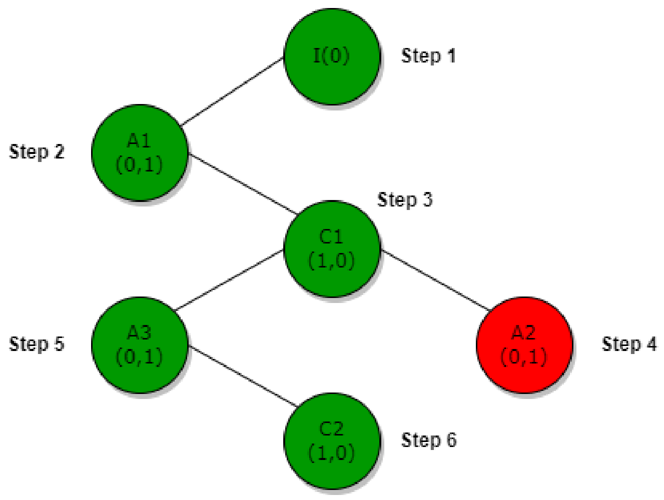


Figure 4.3: Graph of the running of a hyper-heuristic with two moves accepted and one move rejected. I(initialisationIndex) = Initialisation of a solution in memory, A(indexFrom, indexTo) = Application of a low-level heuristic, C(indexFrom, indexTo) = Copying of a solution from one memory location to another. Green = Visited node, Red = Unvisited node.

The proposed solution to this problem is to model the execution of the hyper-heuristic as a graph, with each step of the hyper-heuristic added sequentially as a node. The edges of the graph are dependent on how the memory indices are accessed over time, with dead branches indicating rejected moves. When the final node is added, the following algorithm will be applied to the graph:

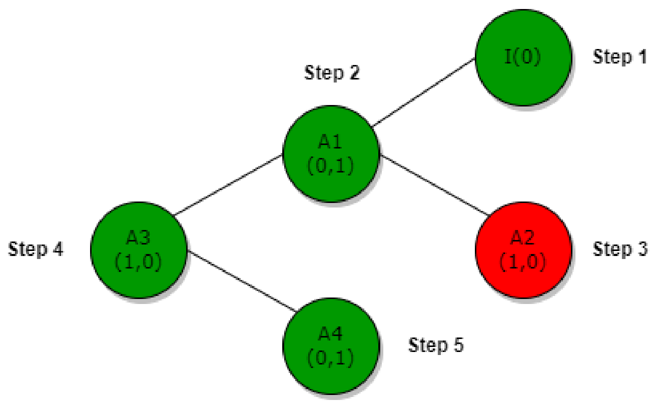


Figure 4.4: Graph of the running of a hyper-heuristic with three moves accepted and one move rejected. The candidate and current solution indices are swapped when a move is accepted in this implementation.

Algorithm 1 calculateAcceptanceDecisions

```
push the final node to an empty stack
mark the final node as visited

while the stack is not empty do
    $n \leftarrow$ pop node from stack
    if $n$ is not the root node then
        for each parent node of $n$ do
            if the parent node is not visited then
                push the parent node to the stack
                mark the parent node as visited
```

This algorithm calculates the acceptance decisions by visiting the parent nodes of each node, starting at the final node. Each visited application node counts as an accepted move, and each unvisited application node counts as a rejected move.

Figure 4.3 shows a hyper-heuristic in which the first low-level heuristic to be applied had its resulting solution accepted. The second move was rejected and the third was accepted. We can see the first move was accepted as after the heuristic was applied to the current solution, the next step was to copy the resulting solution into the memory location of the current solution. We can see the second move was rejected as it lead to a dead branch, with the solution not being copied before the next heuristic was applied. Finally, we can see that the last move was accepted as the resulting solution was copied into the memory location of the current solution in the next step. Figure 4.4 shows a hyper-heuristic that does not copy the resulting solution into the memory location of the current solution when a move is accepted but swaps the memory indices of the candidate and current solution.

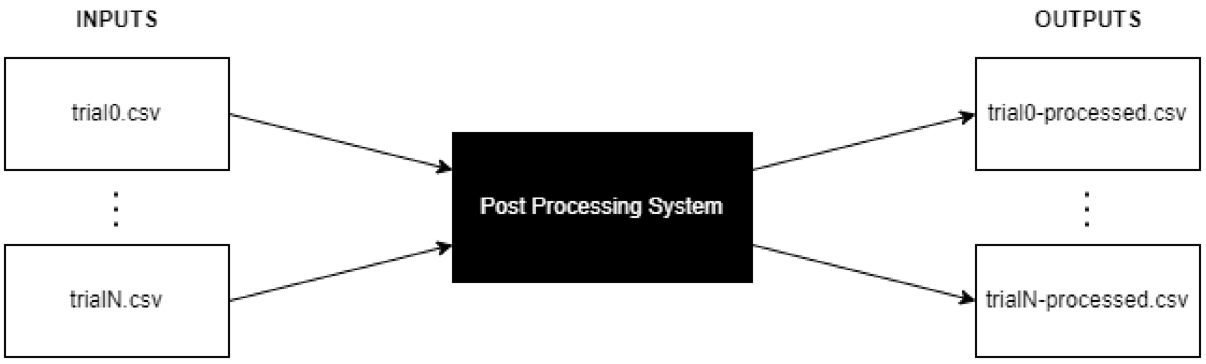


Figure 4.5: Black box input-output diagram of the post processing system, taking as input the data collected from running a hyper-heuristic on a single instance of a problem domain.

Figure 4.5 offers a black box input-output diagram on how the system will function. The post processing system takes in files for N trials and outputs N processed files for each trial.

4.1.3 Data Filtering System

Now that the data has been processed to include the accept/reject decisions of the hyper-heuristics and the execution time of the trials has been scaled to 10 minutes to match the CHeSC competition machine time, the data is ready to be filtered. The data filtering system is needed to average the results across every trial for a given problem instance and remove unnecessary data that is not needed for each visualisation. This is a vital step as the size of the processed data can become considerably large depending on the number of trials, and the data filtering system overcomes the two main problems which arise from this.

The first problem that arises from this is that without the data filtering system, the full set of processed data will be needed to run the visualisation tool, which may require a significant amount of storage space. The second problem that arises from this is that having to average and filter the data on the fly during the execution of the visualisation tool may require a significant amount of processing time, which makes this approach unfeasible as it would greatly hinder the usability of the tool.

As can be seen in Figure 4.6, The data filtering system overcomes these problems by generating a single filtered file per visualisation for a problem instance, which averages the

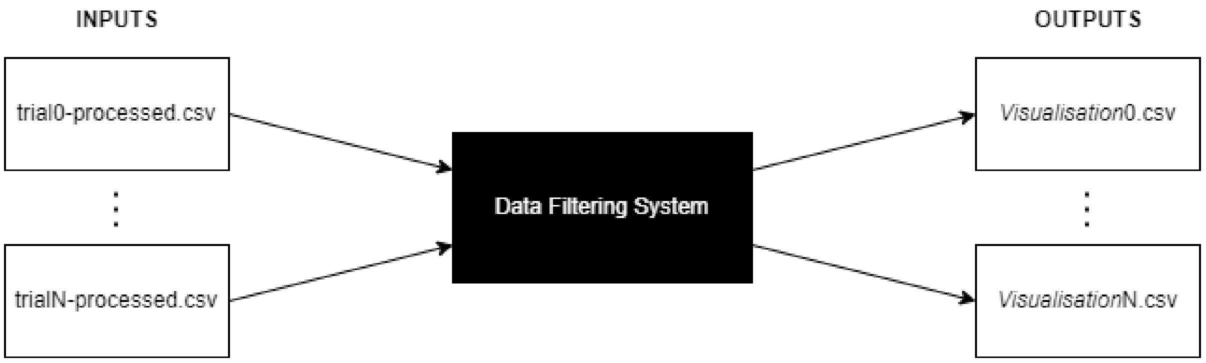


Figure 4.6: Black box input-output diagram of the data filtering system, taking as input the processed data for a single instance of a problem domain.

data per trial and contains only what is necessary for the visualisation in question. This significantly decreases the amount of data required to run the tool, as well as making the processing required to generate the visualisation much less intensive and more importantly, much faster.

4.1.4 Visualisation System

The visualisation system is tasked with generating visualisations for each filtered visualisation file. When the user selects which visualisation they would like to see of a hyper-heuristics performance on a problem instance, the system will fetch the relevant filtered file. Then, it will parse the file based on the visualisation required and the visualisation will be displayed to the user via the graphical user interface.

4.1.5 Graphical User Interface

The Graphical User Interface (GUI) is tasked with displaying the visualisations to the user, as well as giving them options on which hyper-heuristic, problem domain, problem instance, and visualisation that they can view data for.

Figure 4.7 shows a mock-up of how the GUI may look for HyVis. The visualisation being displayed is a progress plot, and the title of the visualisation informs the user that it is displaying data for the Simple Random Naive Acceptance hyper-heuristic on instance 7 of the Bin Packing problem domain.

The menu bar at the top of the display contains the options ”File”, ”Hyper-Heuristic”,

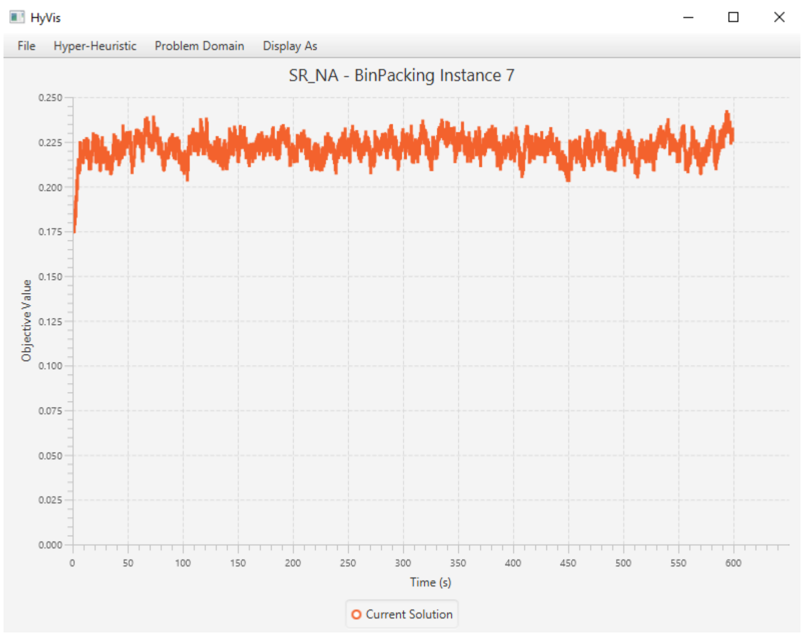


Figure 4.7: Mock-up of how the GUI may look for HyVis, created in JavaFX.

"Problem Domain", and "Display as". The "File" option will enable the user to exit the program and export the currently displayed visualisation as a PNG file, for use outside of the tool. The "Hyper-Heuristic" option will enable the user to select the hyper-heuristic to be visualised. The "Problem Domain" option will allow the user to select the problem domain and then the instance that they would like to be visualised. The "Display As" option will allow the user to select between the different visualisations for the hyper-heuristic and problem instance selected.

Progess

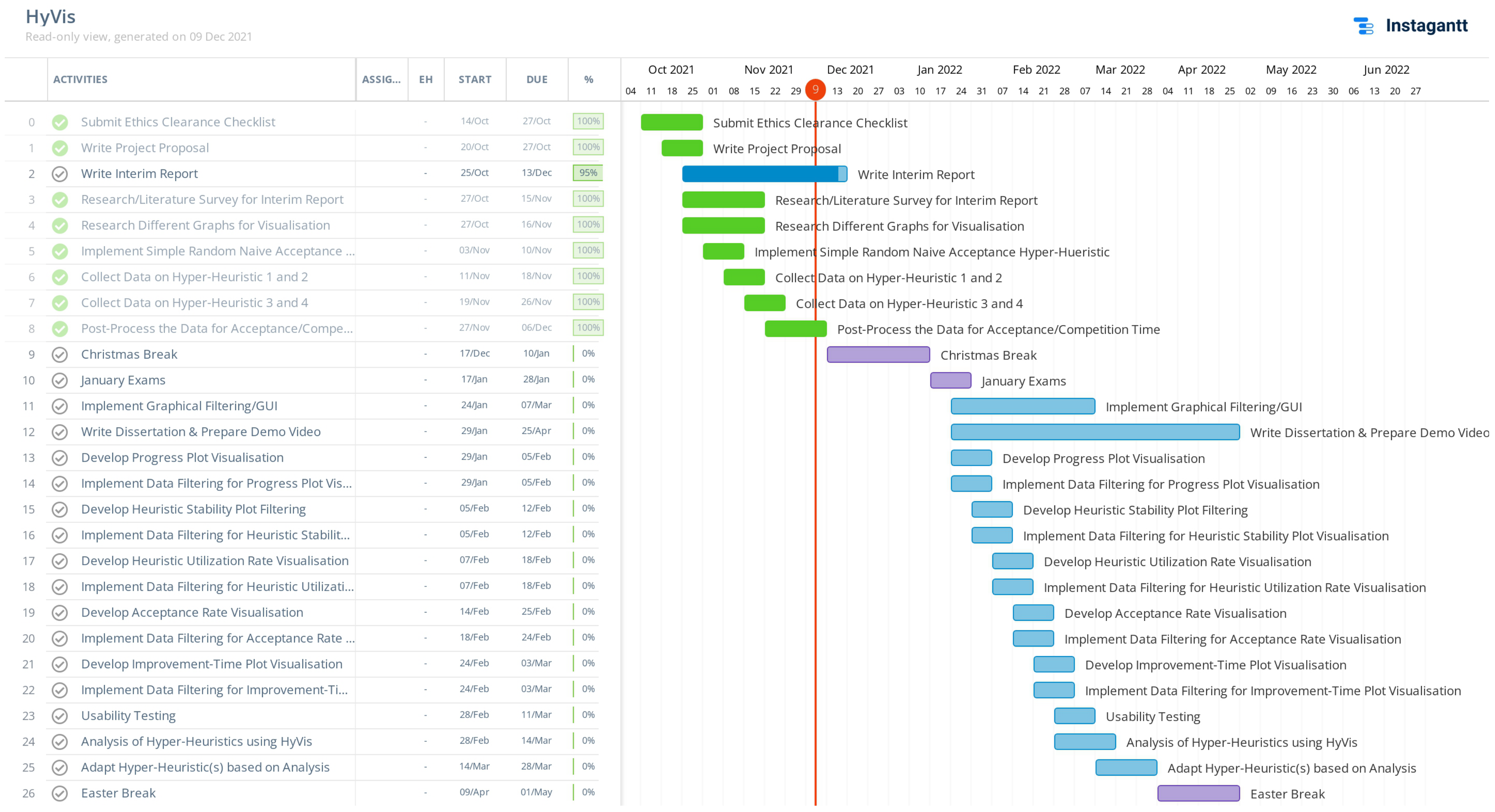
5.1 Project Management

The waterfall methodology has been used for the management of the project, with a Gaant chart being utilized to decompose the project into a sequence of ordered tasks. The project is currently on track, with all of the tasks required up until this point in time completed. However, as the project has developed, events leading to the adjustment of the Gaant chart have occurred. These include research into different visualisations that can be used with Hyvis, deciding and coming up with the visualisations that the tool will offer, and deciding on how many different problem domains/instances to collect data for.

Therefore, as more time has been spent on the project, an improved vision of the future tasks has been gained. This has enabled existing tasks to be decomposed further into more descriptive components, which better describe the breakdown of the project and enable a more accurate time frame to be set for each of them. For example, in the initial Gaant chart, the plan was to implement two hyper-heuristics and collect data on both of them in order to compare their performance. However, as the project developed and the decision was made that the hyper-heuristics would be run in the same way as the competing hyper-heuristics in the CHeSC competition, it was decided that it would be better to compare four hyper-heuristics, including the competition winner, and this lead to a change in the implementation and data collection processes.

Another example is the expansion of the two tasks ”Develop Visualisation Tools” and ”Implement Graphical Filtering”. After the visualisations had been decided, it did not make sense to keep these tasks vague as they were able to be expanded, leading to a more detailed plan and a better idea of the time frame it will take to complete them. It turns out that due to choosing to include five different types of visualisations, the tasks will take longer than initially planned for in the chart, and this change has enabled this to be taken into account earlier on in the project.

A full list of changes are reflected in the newly updated Gaant chart for the project, as shown in Figure 5.1.


HyVis
Read-only view, generated on 09 Dec 2021

Instagantt

	ACTIVITIES	ASSIG...	EH	START	DUE	%
0	✓ Submit Ethics Clearance Checklist		-	14/Oct	27/Oct	100%
1	✓ Write Project Proposal		-	20/Oct	27/Oct	100%
2	✓ Write Interim Report		-	25/Oct	13/Dec	95%
3	✓ Research/Literature Survey for Interim Report		-	27/Oct	15/Nov	100%
4	✓ Research Different Graphs for Visualisation		-	27/Oct	16/Nov	100%
5	✓ Implement Simple Random Naive Acceptance ...		-	03/Nov	10/Nov	100%
6	✓ Collect Data on Hyper-Heuristic 1 and 2		-	11/Nov	18/Nov	100%
7	✓ Collect Data on Hyper-Heuristic 3 and 4		-	19/Nov	26/Nov	100%
8	✓ Post-Process the Data for Acceptance/Compe...		-	27/Nov	06/Dec	100%
9	✓ Christmas Break		-	17/Dec	10/Jan	0%
10	✓ January Exams		-	17/Jan	28/Jan	0%
11	✓ Implement Graphical Filtering/GUI		-	24/Jan	07/Mar	0%
12	✓ Write Dissertation & Prepare Demo Video		-	29/Jan	25/Apr	0%
13	✓ Develop Progress Plot Visualisation		-	29/Jan	05/Feb	0%
14	✓ Implement Data Filtering for Progress Plot Vis...		-	29/Jan	05/Feb	0%
15	✓ Develop Heuristic Stability Plot Filtering		-	05/Feb	12/Feb	0%
16	✓ Implement Data Filtering for Heuristic Stabilit...		-	05/Feb	12/Feb	0%
17	✓ Develop Heuristic Utilization Rate Visualisation		-	07/Feb	18/Feb	0%
18	✓ Implement Data Filtering for Heuristic Utilizati...		-	07/Feb	18/Feb	0%
19	✓ Develop Acceptance Rate Visualisation		-	14/Feb	25/Feb	0%
20	✓ Implement Data Filtering for Acceptance Rate ...		-	18/Feb	24/Feb	0%
21	✓ Develop Improvement-Time Plot Visualisation		-	24/Feb	03/Mar	0%
22	✓ Implement Data Filtering for Improvement-Ti...		-	24/Feb	03/Mar	0%
23	✓ Usability Testing		-	28/Feb	11/Mar	0%
24	✓ Analysis of Hyper-Heuristics using HyVis		-	28/Feb	14/Mar	0%
25	✓ Adapt Hyper-Heuristic(s) based on Analysis		-	14/Mar	28/Mar	0%
26	✓ Easter Break		-	09/Apr	01/May	0%

Oct 2021
Nov 2021
Dec 2021
Jan 2022
Feb 2022
Mar 2022
Apr 2022
May 2022
Jun 2022
04 11 18 25 01 08 15 22 29 9 13 20 27 03 10 17 24 31 07 14 21 28 07 14 21 28 04 11 18 25 02 09 16 23 30 06 13 20 27
Submit Ethics Clearance Checklist
Write Project Proposal
Write Interim Report
Research/Literature Survey for Interim Report
Research Different Graphs for Visualisation
Implement Simple Random Naive Acceptance Hyper-Hueristic
Collect Data on Hyper-Heuristic 1 and 2
Collect Data on Hyper-Heuristic 3 and 4
Post-Process the Data for Acceptance/Competition Time
Christmas Break
January Exams
Implement Graphical Filtering/GUI
Write Dissertation & Prepare Demo Video
Develop Progress Plot Visualisation
Implement Data Filtering for Progress Plot Visualisation
Develop Heuristic Stability Plot Filtering
Implement Data Filtering for Heuristic Stability Plot Visualisation
Develop Heuristic Utilization Rate Visualisation
Implement Data Filtering for Heuristic Utilization Rate Visualisation
Develop Acceptance Rate Visualisation
Implement Data Filtering for Acceptance Rate Visualisation
Develop Improvement-Time Plot Visualisation
Implement Data Filtering for Improvement-Time Plot Visualisation
Usability Testing
Analysis of Hyper-Heuristics using HyVis
Adapt Hyper-Heuristic(s) based on Analysis
Easter Break

Figure 5.1: Updated Gaant chart detailing the sequence of tasks for the project.

5.2 Contributions and Reflections

Reflecting on the project, I believe that good progress has been made towards the realisation of HyVis. So far, existing visualisations and visualisation tools have been analysed, the visualisations that HyVis will offer have been chosen, the hyper-heuristics to be analysed have been implemented/selected, data has been collected and then processed for each of the hyper-heuristics across the CHeSC problem instances, and the design of HyVis has been detailed for each component of the tool. The main tasks to be completed involve the implementation of the tool and using the visualisations it generates to analyse and compare the performance of the hyper-heuristics.

The key achievements/contributions that have been made so far in the project are the proposal of both the average heuristic stability plot and the average improvement-time plot, both novel visualisations for use with HyVis. The hope for these visualisations is that they are able to offer further information about the operation of hyper-heuristics than what is currently available, and can be used by experts in order to analyse the performance of a hyper-heuristic, using the information learnt from the visualisations to improve them.

Although the project is currently on track and progressing well, the initial hope was that the project would be ahead of schedule at this current phase. This is due to the large workload for the project after the January exams, and completing the tasks ahead of schedule would allow for more time to be spent refining the project, and more time to deal with potential unexpected challenges. With that being said, the project is by no means progressing slower than planned and is on track to being completed when expected. Overall, I am pleased with the progress made so far and HyVis is looking like it will be a valuable tool for the analyses and improvement of Hyflex implemented hyper-heuristics.

Bibliography

[1] Adriaensen, S., and Now'e, A. Case study: An analysis of accidental complexity in a state-of-the-art hyper-heuristic for hyflex. In *Evolutionary Computation (CEC), 2016 IEEE Congress on* (2016), IEEE.

[2] Burke, E., Curtois, T., Hyde, M., Kendall, G., Ochoa, G., Petrovic, S., and Antonio Vázquez Rodríguez, J. Hyflex: A flexible framework for the design and analysis of hyper-heuristics. *Proceedings of the 4th Multidisciplinary International Scheduling Conference: Theory & Applications (MISTA 2009)* (2009), 790–797.

[3] Burke, E., Gendreau, M., Hyde, M., Kendall, G., McCollum, B., Ochoa, G., Parkes, A., and Petrovic, S. The cross-domain heuristic search challenge – an international research competition. *LION 2011: Learning and Intelligent Optimization* (2011), 631–634.

[4] Burke, E. K., Hyde, M. R., Kendall, G., Ochoa, G., Özcan, E., and Woodward, J. R. International series in operations research and management science. *A classification of hyper-heuristic approaches: Revisited 272* (2018), 453–477.

[5] Cowling, P., Kendall, G., and Soubeiga, E. A hyperheuristic approach to scheduling a sales summit. *Practice and Theory of Automated Timetabling III* (2001), 176–190.

[6] Cruz Reyes, L., Santillán, C., Castillo-García, N., Quiroz, M., Ochoa, A., and Hernández-Hernández, P. A visualization tool for heuristic algorithms analysis. *Advances in Intelligent Systems and Computing* (2013), 515–524.

[7] Drake, J. H., Kheiri, A., Özcan, E., and Burke, E. K. Recent advances in selection hyper-heuristics. *European journal of operational research* (2020), 405–428.

[8] Drake, J. H., Özcan, E., and Burke, E. K. A modified choice function hyper-heuristic controlling unary and binary operators. In *2015 IEEE Congress on Evolutionary Computation (CEC)* (2015), pp. 3389–3396.

[9] Krasnogor, N. *Studies in the Theory and Design Space of Memetic Algorithms.* PhD thesis, University of the West of England, Bristol, U.K., 2002.

[10] Misir, M., Verbeeck, K., De Causmaecker, P., and Vanden Berghe, G. An intelligent hyper-heuristic framework for chesc 2011. In *Learning and Intelligent OptimizatioN* (2012), Springer, pp. 461–466.

[11] Pylyavskyy, Y., Kheiri, A., and Ahmed, L. A reinforcement learning hyper-heuristic for the optimisation of flight connections. *2020 IEEE Congress on Evolutionary Computation (CEC)* (2020), 1–8.

[12] Özcan, E., Bilgin, B., and Korkmaz, E. A comprehensive analysis of hyper-heuristics. *Intelligent Data Analysis* (2008), 3–23.

[13] Özcan, E., Drake, J. H., Altintaş, C., and Asta, S. A self-adaptive multi-meme memetic algorithm co-evolving utility scores to control genetic operators and their parameter settings. *Applied Soft Computing* (2016), 81–93.