G52GRP Interim Group Report

29th November 2009

A Visualization Tool for Selection Hyper-Heuristics

SUPERVISOR:	**Dr. Ender Ozcan**	exo

GROUP:	**gp09-exo**	
	Lao Jingqi	jxl29u
	Zhang Chao	cxz09u
	Ben Jenkinson	bxj08u
	Thomas Barton	txb18u
	Alexander Jermstad	asj08u

CONTENTS

1. INTRODUCTION

As one problem-solving algorithm, heuristic has a 40-year history (Kendall et al., 2003) and it was widely used to solve the problems such as data-mining, cutting, packing, and scheduling (Ozcan et al., 2009, Bai et al., 2007), it also has been applied in the anti-virus technology (Lakhotia & Mohammed, 2004). One successor of heuristic, meta-heuristic, was also a success that has been widely applied to many different problems. However, many heuristics are problem-dependent (Ayob & Kendall, 2003); and meta-heuristic is often fragile and unstable (Ozcan et al., 2009), hence the recent trend of research paid more attention to **hyper-heuristics**, whose motivation is to build one problem-solving system so that it can deal with a problem domain rather than one particular problem (Burke et al., 2005; Ross, 2005). Since there was an increasing importance of hyper-heuristic, it is necessary to facilitate the people understands towards this abstract algorithm. And our project aims at developing one visualization tool of hyper-heuristics. Our first prototype was finish with **3 low-level heuristics** and **Simple Random (SR)** selecting function, and we will finally extend to 5 low-level heuristics and 2 or 3 heuristic selecting methods.

2. BACKGROUND INFORMATION

2.1. Hyper-heuristic

In general, one **hyper-heuristic method** is always described as "a heuristics to choose heuristics", which would maintain some criteria to select one heuristic to a particular question from many different heuristics (low-level heuristics). And the ASAP Group (2009) gave the definition as:

"A hyper-heuristic is a search method or learning mechanism for selecting or generating heuristics to solve computational search problems."

There are several crucial components included: **heuristic selection methods**, **move acceptance** (Ozcan et al., 2009). And there are also some important components related to heuristics, such as candidate solutions, low-level heuristics, benchmark functions, etc. In addition, Ouelhadj and Petrovic (2008) thought there are two types of hyper-heuristics: **constructive** and **perturbative** hyper-heuristics; where constructive hyper heuristic construct a solution by randomly using constructive heuristic while the perturbative one begin with a complete solution which would be improved through selecting appropriate low level heuristics.

Selection Methods

Heuristic selection methods decide how a hyper-heuristic choose low level heuristics, hence their **performance** would directly affect the efficiency of hyper heuristic application. Selection processes would also take charge of applying the heuristic to the candidate solution. According to Cowling et al. (2001), some of these selection methods were defined as below:

- **Simple Random (SR)**, *which select the low level heuristics **randomly**.*
- **Greedy Random (GR)**, *which **apply all** low level heuristics to the same candidate solution separately and choose the heuristic that generates the best change on the objective function.*
- **Choice Function (CF)**, *which **dynamically** score each heuristic weighing their individual performance, combined performance with previously invoked heuristic and the time passed since the last call to the heuristic at a given step then a heuristic is chosen based on these scores.*

In this project, we will firstly use the **Simple Random (SR)** method in the prototype, and then we will extend it with the other two.

Move Acceptance

Another significant part in hyper-heuristics is the move acceptance, because it's extremely important to the **decision making** (Ozcan et al., 2009). The type of acceptance will decide whether and how a new-generated candidate solution would be accepted by the H.H. method. And also, Ozcan et al. (2008) introduced several move acceptances, such as:

- **Only Improving (OI):** *only accept the solutions that made **improvements**.*
- **Improving and Equal (IE):** *accept the solutions that made **no decrease.***
- **Exponential Monte Carlo (EMC):** *the fitness decreasing would be accepted within a **negative exponential ratio** of the probability of acceptance; as shown in* **formula (1)***:*

$$\mathbf{P}_t = e^{-\frac{\Delta f/N}{1-t/\mathrm{D}}} \qquad (1)$$

Where Δf is the fitness improvement after the t^{th} iteration, D is the maximum number of iterations; N is an expected range for the maximum fitness change.

- **Great Deluge (GD):** *all moves which generated a better or equal objective value than a* ***level computed*** *at each step are accepted. And the objective value is defined as* **formula (2)**:

$$\boldsymbol{\theta_t} = \boldsymbol{f_o} + \boldsymbol{N} \times (\mathbf{1} - \frac{\boldsymbol{t}}{\mathbf{D}}) \qquad (2)$$

Where t is the threshold level during the t^{th} iteration in a minimization problem, D is the maximum number of iterations; N is an expected range for the maximum fitness change.

And our project would use **Only Improving (OI)** method, which only accepts the new-generated solutions that have made an improvement.

Low-level Heuristics

A H.H. algorithm would hold a domain of low-level heuristics, which selection method would act on. One low-level heuristic is algorithms that will be applied on domain barrier (candidate solution) then generate a new one. Ersoy (2007) distinguished heuristic into **mutational heuristics** and **hill climbers**. Some ideas of mutational heuristics are displayed as following:

- **Inverse**: ***flip*** *the values of a particular section of one candidate solution.*
- **Reverse**: *reverse the* ***order*** *of one indicated section of the candidate solution.*
- **Shift**: ***shift*** *several bits on one section of the candidate solution, however, there exists a problem when the shift method meets an irregular section selection.*

Hill-climbing heuristic is another type of heuristic; Ozcan (2009) showed that the hill-climbing method would bring some differences to one hyper-heuristic framework. Hence, hill-climbing is not only one type of heuristic, but also an important component of hyper-heuristic framework.

- **Hill-climbing**: *change the value of one candidate solution* ***bit-by-bit****, and accepted the improved changes, and finally get the solution.*

There were 3 simple heuristic methods in the first-step sample, and our project will finally extend these methods into **5**.

Hyper-heuristic Process

In a simple hyperheuristic, a single candidate solution representing a problem at hand is continuously processed in an iterative cycle until some termination criteria are satisfied at each step, a heuristic is selected based on some problem independent criteria, such as the fitness change. Then it is applied to the candidate solution producing a new one. This new solution is accepted or rejected based on a strategy within the hyper heuristic (Ersoy, 2007).

We use the single point search process as the example. As **Figure 1** shows, one candidate solution (**S_t**) is initialized into a new solution (or solutions) using a selected heuristic (or heuristics). Then, a move acceptance method is employed to decide whether to accept or reject a resultant solution. This process will continue if $\boldsymbol{S_t}$ cannot meet the requirements (Ozcan et al., 2009).

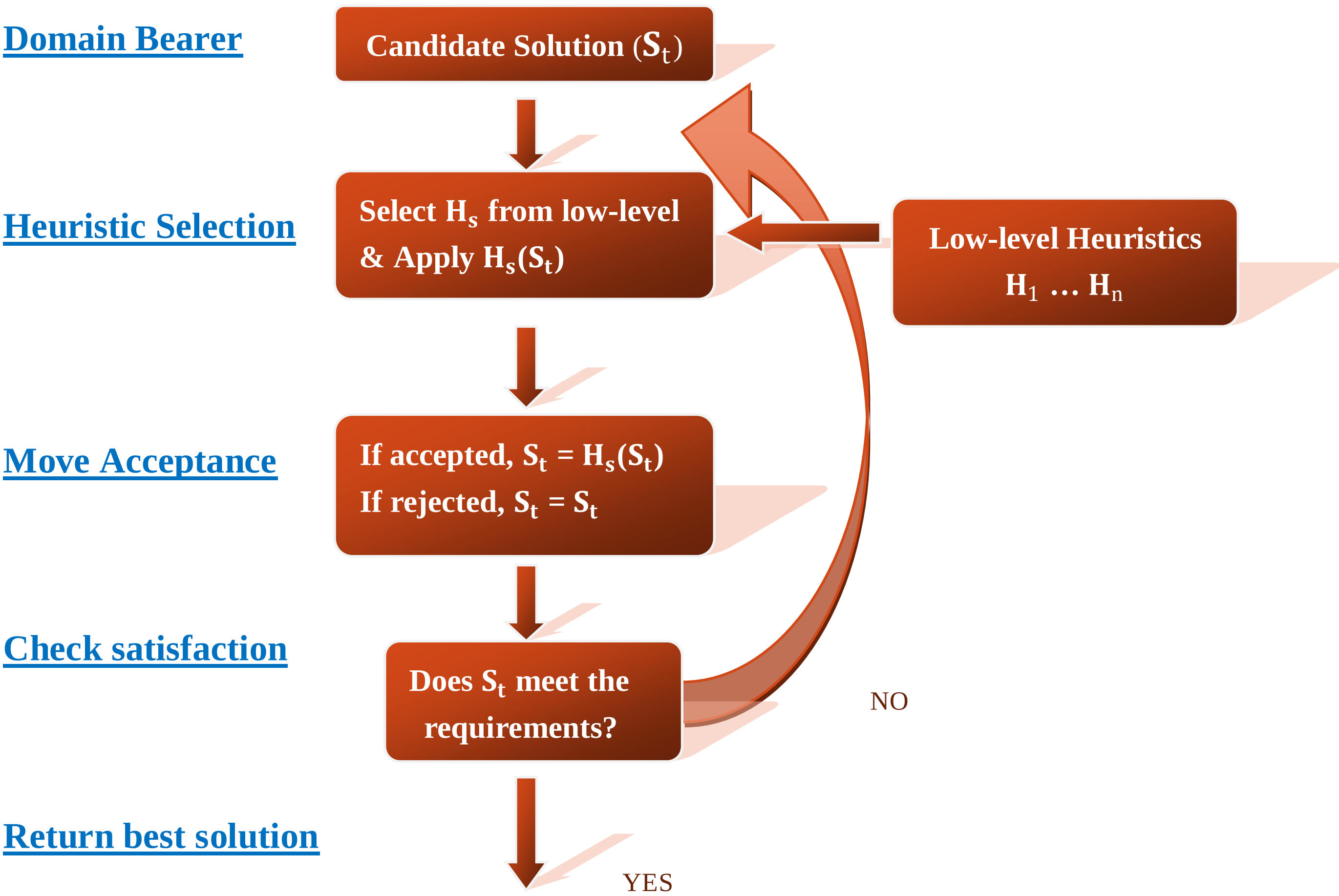


Figure 1. *Operation of a selection hyper-heuristic framework based on a single point search*

And **Figure 2** shows the pseudo code for this process:

```
Hyper-Heuristic Algorithm
     candidateSolution <- generateCS(randomSeed);
     while(!meetRequirement(candidateSolution))
          choosed_H <- selecting_H(lowLevel_Hs);
          newCS <- choosed_H(candidateSolution);
          Acceptance(candidateSolution, newCS);
     return candidateSolution;
```

Figure 2. *Pseudo code of a selection hyper-heuristic based on a single point search*

2.2. What is **VSH**?

VSH is to be the name of our application. VSH is literally defined as "**Visualization for Selection Hyper-heuristics**". As its name suggested, this project is a visualization tool to facilitate people's understanding about hyper-heuristics. VSH is developed with purpose that to give a direct and clear expression of hyper-heuristics to the users; and this project will show how a hyper-heuristic works through simulating the working processes of an exact hyper-heuristic. The framework of VSH is an object oriented one which is written in JAVA SE 6.0 version. The advance of using an object oriented language is that it can benefit the code minimising and reusing. In addition, VSH will be an open source project to response the increasing interest for hyper-heuristics at present. In general, VSH can provide a good understanding of hyper-heuristics.

On the other hand, though there were several visualization tools to represent heuristic processes, there were no such applications for hyper-heuristics at present. Undoubtedly, the blank of similar software in the market would provide an obvious opportunity to our project. The increasing attention on hyper-heuristics would also raise its requirements. Hence VSH would bring optimistic benefit either as free software or commercial software.

3. REQUIREMENTS SPECIFICATION

3.1. Functional Requirements:

- To be able to **demonstrate** how hyper-heuristics work using a visualization tool, the idea being to show the search space as a function and how it changes.
- Implement a form of the Travelling Salesperson Problem using permutation representation.
- Create a binary representation of the function.
- Create a simple GUI with animation to represent the operations on the function.

3.1.1. Input

The user should be able to choose the function to be operated upon and specify the parameters before initialising the hyper heuristics. This may be in the form of predefined drop down boxes. Additionally Java has the capability to create classes on the fly thereby making it possibly for users to possibly enter their own functions. They should also be able to choose which hyper-heuristic selection methods (simple random, greedy random and choice function) are applied before initialising the visualisation. Initially for the prototype we'll just use simple random while we develop the framework, then it can be expanded to include the additional hyper heuristic selection methods.

For the lower level heuristics algorithms such as the inverse, reverse and hill-climbing should be implemented. Similarly with the lower level heuristics initially on the prototype we can implement the flip or swap, eventually including both along with the combination.

All results should also be stored as the programs runs so that a table of results can be displayed for analysis at the end.

The user should also be able to define the termination criterion so the program knows then to stop. This will be done by specifying a number of iterations for the program to apply the heuristics, or specifying the size of the number below which the program shouldn't continue to perform calculations, or a combination of both. Other constraints will also be considered such as a time constraints.

A candidate solution must initially be randomly generated which can be fed into the heuristic process so there's a starting point to compare the results of applying the heuristics.

An overview of how the different classes should be able to interact can be found in a UML sequence diagram in the appendix.

3.1.2. Output

The function should be illustrated graphically through the GUI.

The graphical display should be scalable, along with the rest of the user interface for use on monitors of different resolutions. Although it can be optimised for a screen resolution of 1024x768 seeing as this is currently one of, if not the most common screen resolution in use today.[1]

The encoding and decoding of the data and the manipulation of the binary representation is to be illustrated as a part of the GUI. The binary representation is to be graphically represented and will interact with different parts of the screen to illustrate the operations being performed upon it at any one time, allowing the user to follow the process.

Key to representing the progress of the hyper heuristic calculations is making sure that the visualisation runs at a reasonable speed so that the user can actually digest the information being presented to them. The speed can be arranged with respect to the speed of the machine it is operating on, although this could cause issues with vastly different specked machines and how quickly the visualisation is rendered so it should be possible for the user to also choose the animation time or time between screens. This will require slowing down the process, however this could cause flickering and an uncomfortable viewing experience so the implementation of z-buffering is also

[1] http://www.w3schools.com/browsers/browsers_display.asp

necessary. Z buffering is an algorithm often used in 3D graphics to ensure perspective of objects are maintained on a computer screen as in real life, i.e. if there's a solid object (or picture or pixels) in the foreground then they will block the view of the one behind it. This can be used to create the next screen prior to displaying it, storing it in the z or depth buffer and then layer it on top of the previous screen, similar to flick book. This will remove the flickering effect. The use of threads could help in facilitating this and will be investigated.

For the actual graphs themselves the axis will not need to excess the 0 – 1024 range due to the binary nature of the data we are dealing with.

All changes to the input data and resulting heuristically generated data is to be stored so that it can be later viewed in a tabular form for easy viewing and possible statistical analysis if needed.

3.2. Non-functional Requirements:

- Programming language: JAVA
- Project website: http://code.google.com/p/VSH/ Use of subversion to track changes.
- OS environment: Primarily Windows, although due to widespread use and flexibility the Java can be used on many other operating systems if required.

3.2.1. Hardware & Software Requirements

It was easy to decide on Java as the language of choice for the project given everyone's common experience of at least one year working with the language. This gives us a strong core understanding of the language between us to give us sound starting point and also grants us the ability to improve our knowledge and adapt how we're able to apply it together to best utilise the language not only for this project, but future projects too. Its flexibility also bodes well meaning we could edit using any text editor although JCreater is our development program of choice due to familiarity from the first year.

As the project will likely develop into a something larger than any of us have worked on before at this stage and work will be continuous over a period of months a tool to amalgamate all of our work in one place as well as track any changes we make is required. We decided to use Google Code for this as it is free, easy to use, comes well recommended and makes great use of subversion to track changes. This way each member of the group can upload new code or amendments to current files and these changes will immediately be available to everyone. This minimises the chances of overlapping code and issues with working on different sections of code at the same time, and breaking it due to changes that are not in keeping with the other files also being worked on. As the VSH project will be open source it's also important to ensure that access to our source code is made public and available to the community to help encourage the development of the field of HH and encourage further understanding of this field. Therefore hosting the project somewhere where people will be able to gain access is also important.

4. INITIAL DESIGN

4.1. Software & Hardware

One of the first things decided about the project was the choice of programming language. We immediately settled on Java due to our shared knowledge of the language. With our language decided, the hardware took care of itself; our program would quite happily run on any operating system that supported the Java runtime.

4.2. Key implementation

- A **user interface** class
  This class would mainly contain the user interface; and user can also choose some running parameters such as the hyper-heuristic methods (i.e. Simple Random, Greedy Random, and Choice Function), set domain bearer and set object functions.
- A **dynamic graphic-drawing panel** class
  Here the code would work out the location of each new candidate solution and represent them in the object function curve.
- A **process-display panel** class
  This is class where the main computing processes located in; and it also can show how heuristic and hyper-heuristic interacted. The parameters get form the

user interface would be past here and this class would invoke the hyper-heuristic implementation class (the following one) to get the running data and then show the processes dynamically. It would also invoke the graphic-drawing class.

- A **hyper-heuristic implementation** class

  The main function to apply one specific hyper-heuristic method and very related to the process-display class. Thus, this class would get the computational parameters from the display process and then choose one heuristic through which to generate a new solution, compare those solutions, and finally return all the related data.

- **Low level heuristic Object** class

  This is an object class that can represent a heuristic method. As an object, these low level heuristic methods can be operated in an array; which means we can add and delete any of the methods conveniently.

4.3. User Interface

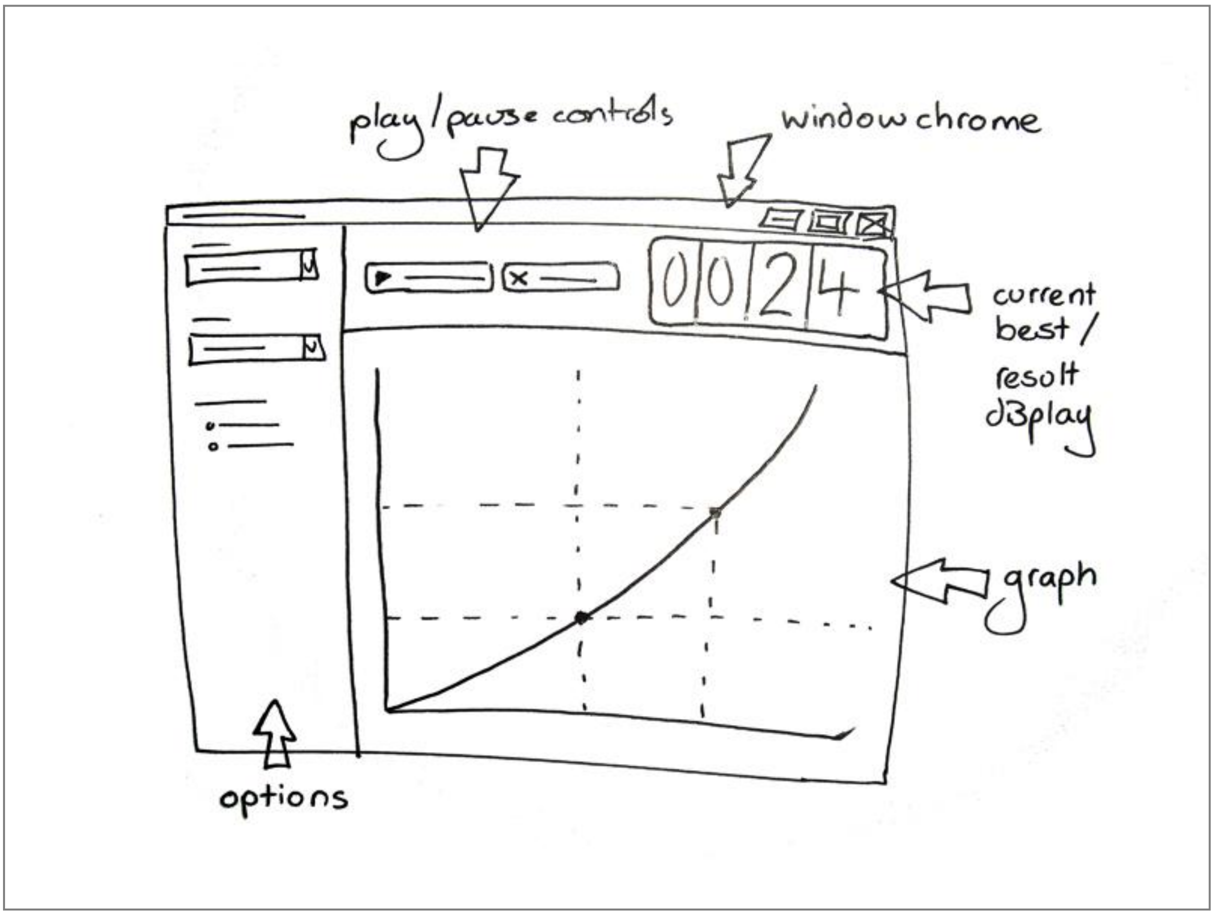
The current interface that graces our program is mainly for testing purposes, a relatively blank slate upon which our visualisation components can be placed, and a space for the control elements.

Since the aim of this project is to improve the user understands of Hyper-heuristics; the user interface should be designed to be as human-friendly as possible and our final interface will be very different from the current one.

Our original idea for the interface was to construct a 3D rotating cube upon which sections of the interface would be displayed on each face. After looking into this, we decided it was completely extravagant and unnecessary and was far too difficult to accomplish. The UI of a program should be invisible; it should be able to fade into the background, merely facilitating the desires of the user.

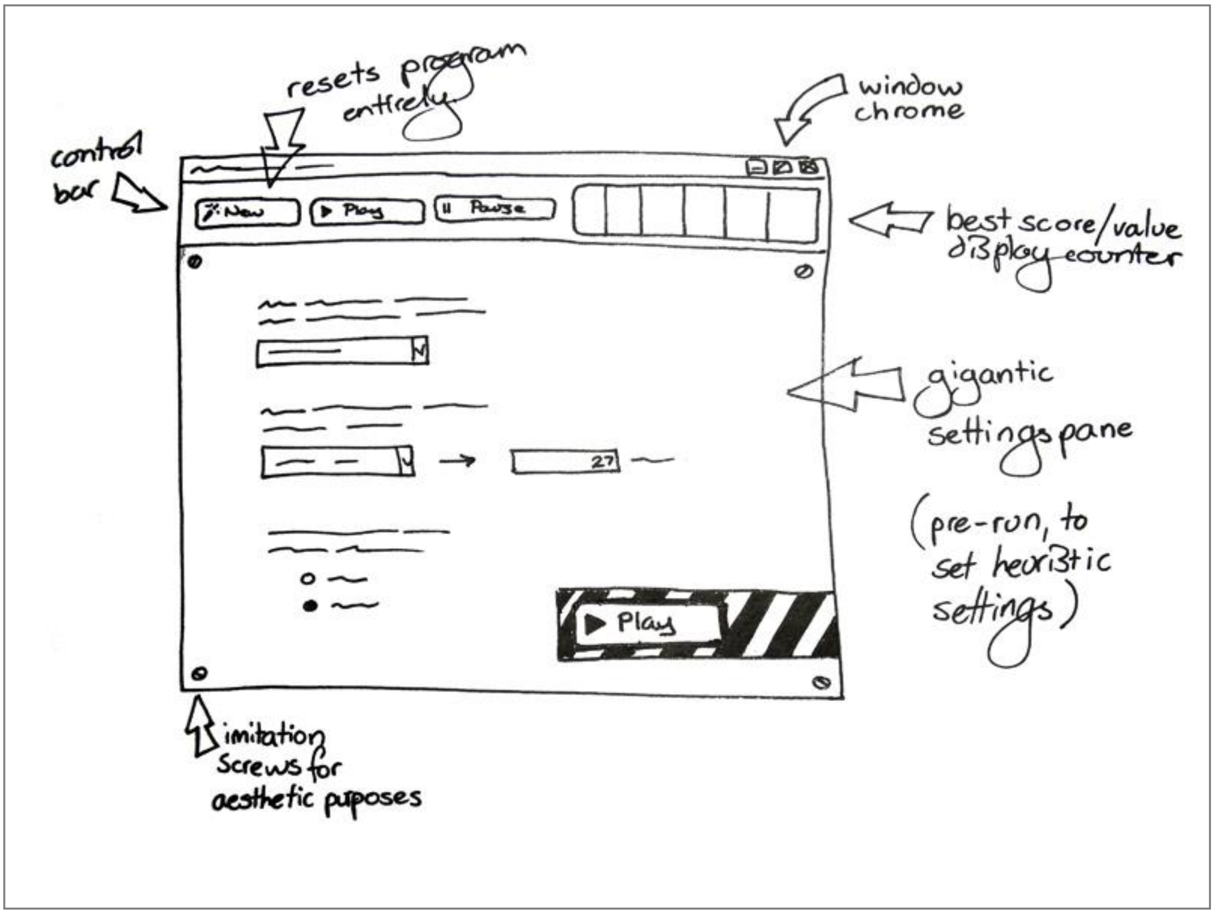
The following are preliminary sketches of our user-interface: The sketch above shows an initial idea for the application, you can see the sidebar that would contain settings for the hyper-heuristic being used and the problem to be solved, and the control bar at the top to play/pause/step-through the heuristic’s runtime. Finally, there is the large graph pane to visualise the results and the large counter display to show the current best result at every point.

This idea was then refined slightly into the one below, showing the first view of the application, immediately after initialisation. The screen is split into two parts, for reference purposes the upper area is henceforth dubbed the ‘control bar’ and the lower area shall be known as the ‘display pane’.



At this point, we can assume that the user wants to start a visualisation of a hyper-heuristic; that is after all the point of the program. We can then present them with all the settings that they may wish to alter, and that we need to know in order to create a visualisation. These would all be displayed in the display pane. It is important to note that these are all pre-filled with default values or options in order to expedite the process. If the user does not understand an option, they can just ignore it and leave it at the default value. There’s also a good opportunity here to incorporate explanatory tooltips.

Since you don't need to alter the settings while the process is running, you will notice that this design has removed the sidebar that was present in the previous sketch. Instead the user can specify all the settings before running the process, and they can return to this pane at any time by clicking the reset/new button on the control bar.



Although we removed the sidebar for the visualisation display, some sort of static indicator may still be present to indicate what acceptance criteria and other settings are currently operating on the process.

Although the control bar primarily concerns itself with playback, it is still visible when the settings pane is occupying the display pane. Most of the buttons that would be present are greyed out however, visually distinct from the still-active ones. This may seem superfluous, but this bar would provide a constant visual indicator of which options are available at each time.

When the user has selected their problem domain and other options for the process, they may click the immediately obvious Continue/Play button on the settings pane to initialise their hyper-heuristic. The playback controls on the control bar are then enabled and the settings pane disappears, replaced by the visualisation pane which contains the results graph and the visualisation of the heuristic.

The current best result of the hyper-heuristic is displayed in the top-right corner in a dedicated section. This is updated in real-time as the process progresses.

4.4. Prototyping

The visualisation prototype for a Selection Hyper-Heuristic is already available for user to try and feedback. Generally, the interface of prototype can be divided into three parts:

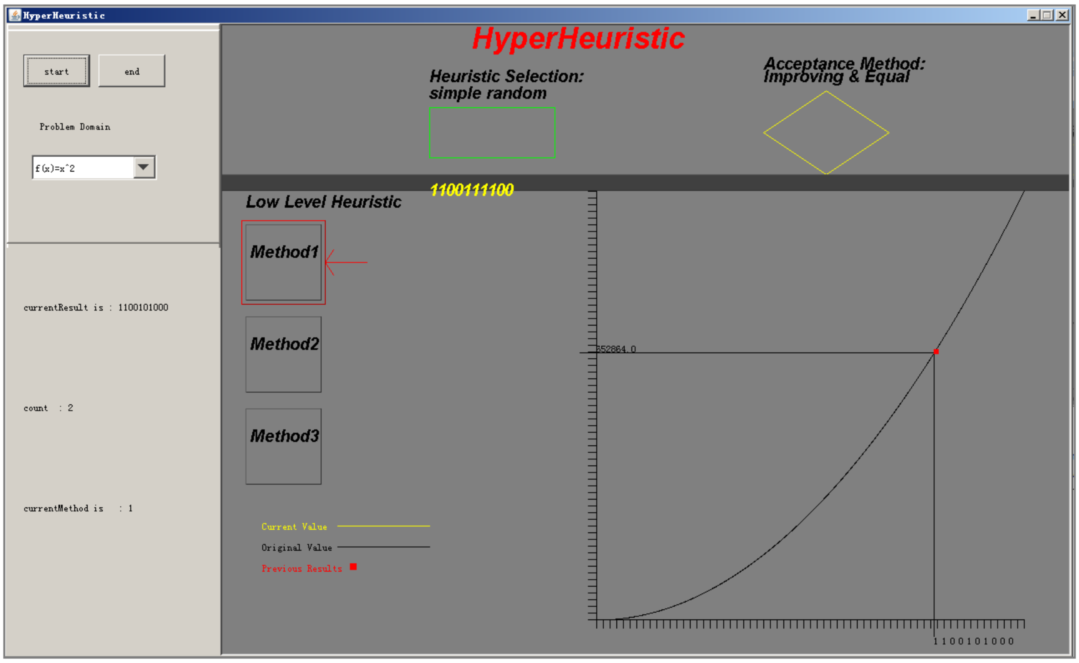
The **button area** consists of three buttons: start button, end button and the problem domain button. Each button processes specific functions. The start and end button provide switch so that the user can control the work process of a choice hyper-heuristic. As for the problem domain button, it allows users to choose a function that a choice hyper-heuristic will work on.

The **bottom left** panel shows the current status. It states the current result, the number of iterations and which low-level heuristic method is chosen.

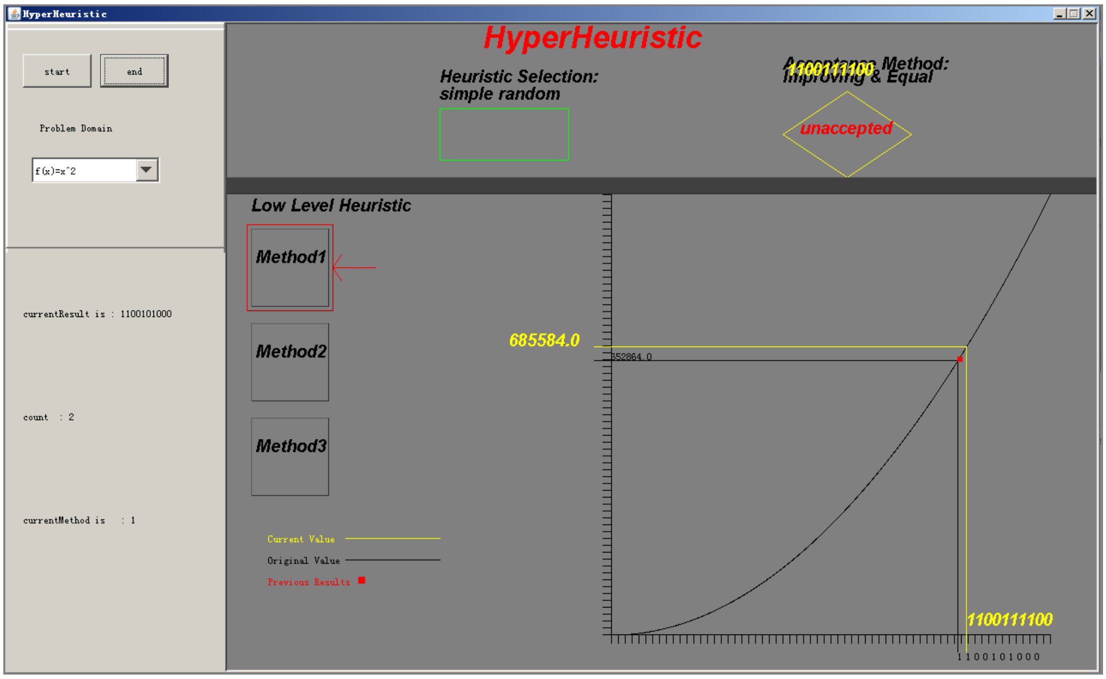
The **right area** mainly provides the visualization on how the choice hyper-heuristic works on an optimisation problem. This area can also be divided into three parts. The top area including the heuristic selection and acceptance method boxes shows the main process of hyper-heuristics. The bottom left area is used to show which low-level heuristic is chosen, how it works and the new solution produced by the low-level heuristic. As for the graphic in the bottom right area, it displays whether to continue the search process using the new solution or the one at hand.

There is example of our prototype demonstrating how a simple random choice hyper-heuristic works on finding the most suitable 10-bit binary representation makes the value of function $y=x^2$ as small as possible. There are two stages for the iteration: the **selection of low-level heuristics** and **acceptance method**.

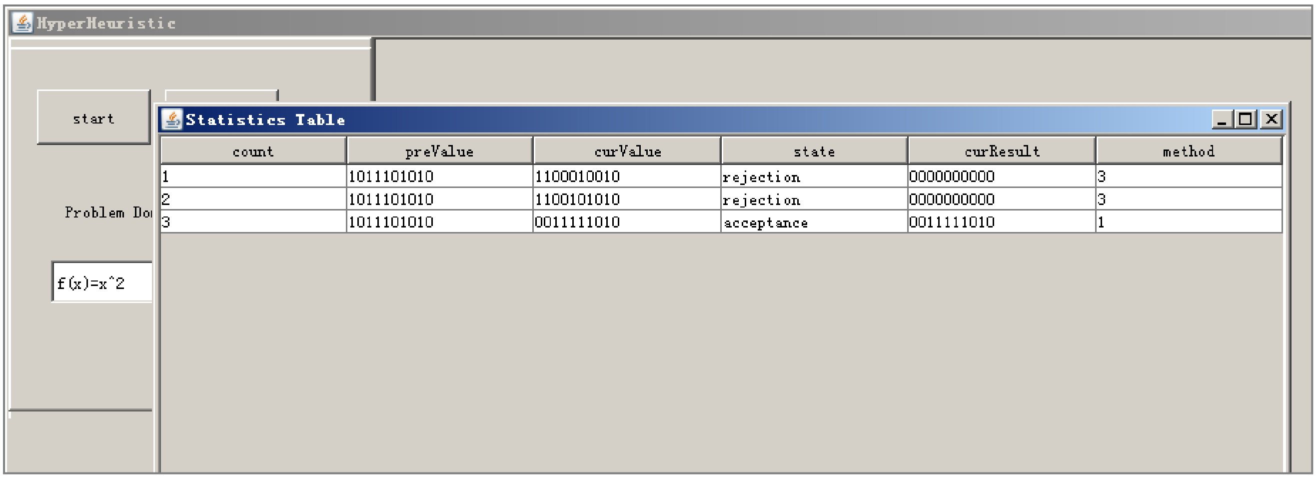
In this iteration, when the number enters the box of heuristics selection, a low-level heuristics will be chosen randomly and it will produce a new number which will be returned to the box. When a heuristic method is selected, it will be boxed in red (as below). Now a new candidate solution is rising to the green box; and the green box will contain the new result that was generated by the low-level heuristic.



When the new number enters the box of acceptance method, the new number's corresponding value of function will be draw on the graphic by yellow line. Then the program will make a decision whether accept the new number or keep the old one by comparing the positions of two numbers. The yellow box would give information about the acceptance; and the result would also be highlighted in the dynamic diagram. Moreover, the red dots are used to record the previous results.



Finally, if the end button is pressed, there will be a pop-up window of the statistics table; and this history record can be used to draw the history diagram as an extension.



5. DISCUSSION OF PROBLEMS

Being that this was the first time any member of the group had ever worked alongside so many others, the first problems we encountered were organizing the workload, keeping to deadlines and making sure everyone understood what needed to be done. This was solved by having two weekly group meetings, one formal; with the project supervisor Ender Ozcan, the other, informal which was usually carried out in the Computer Labs.

The formal meetings underpinned what was to be done over the upcoming weeks, keeping to a timeline that allowed us not to get caught behind. The informal meetings allowed us to keep in contact mid week to settle any problems we had with any of the work and help other group members out.

Initially one of the problems we face is the GUI and how to draw it efficiently, currently the graph is drawn using 1024 points, this requires that on every redraw 1024 individual dots must be created on screen, the x y coordinates of these dots is calculated using the problem domain, the graph offset and the size of the graph. Currently this is the most efficient and accurate way we can think of for drawing the graph, it was suggested that you can draw curves in java using a curve function; however for the prototype we have just used the direct method of calculating the points and drawing them.

Another problem we encountered was when implementing the graph it dawned upon us that we need to choose where to run the calculations for the selected problem domain, being that our GUI is laid out using panels it seemed wise that each problem domain had its own panel, and that the calculations for the heuristic were run in its own panel, as opposed to the run section of the main program. Currently in the prototype everything is very procedural and the calculations for the hard coded problem domain is done in the main function, this would make adding other problem domains harder and so will be re coded in a more object orientated way in the final revision.

When coding with others it's hard to know what you can and can't edit as you don't want to get in each other's way, when a lot of code is uploaded to the SVN at one time it is very hard to keep up with other group members work, especially if it is directly affecting the coding you had previously worked on. Everyone has access to the code and once you have done some programming you may leave it and come back to find it has been changed by another group member. This is both beneficial and problematic, splitting the code between a numbers of people will decrease the programming time per person, however another member may change your code, and without proper protocols could confuse the original writer. In order to resolve this problem code must be well documented with reasonable variable names and comments explaining what each function does and, if ambiguous, any other piece of code that isn't simply understood, for example, when drawing the graph it uses a lot of offset and scale variables, this makes reading the code hard however by using static variables for these values, once the code is written it is easy to edit the inputs, and then in this case a comment can be used explaining what each static variable does. By documenting and commenting as we program it makes debugging and further maintenance easier, so the time taken to document is well spent.

We are trying to create a visualisation tool so it would only make sense that the application looks good and displays information in an easy to understand and reasonable manner, by using animations we have tried to make the heuristic as easy to follow as possible, but there are restrictions when using Java as it implements its own UI, this makes the GUI not very appealing and is more for function and information as opposed to outstanding graphics. To aid the display of information we have implemented a statistics window that pops up when the heuristic ends, this allows you to see the calculations, methods used and acceptances of all the previous iterations.

The subject matter of heuristics is quite hard to understand without prior knowledge, and so the first few weeks were spent understanding the specification, we looked for similar applications but only found a few that were applicable, the closest we could find was a genetic binary algorithm applet, this gave us a better understanding of the problem but wasn't perfect. The lack of already existing applications meant we had nothing to base our program off; this made starting harder as we had no founding program we could compare to and improve on.

Finding time in the week that each group member was free was hard as we all have slightly different time tables, coursework deadlines and one member had a part time job which meant some meetings just weren't viable as the time had to be spent meeting other coursework deadlines. To manage the work we set weekly/biweekly goals, this meant that people could slot the work into when it was good for them, the progress would then be presented to the group at the next formal meeting and any advice could be given by other group members and the project supervisor. Allowing each member the choice of what they wanted to work on ensured that everyone was happy with what work they needed to get done, the bigger tasks such as the report was split up equally amongst the group this meant everyone contributed equally and helped us to work alongside each other.

Finally to make sure everyone had some foundation to work on most parts were initially worked on by the whole group, brainstorming, writing notes and bullet points that were distributed to each member using the SVN, this meant that each member had notes that they could work off for their individual sections.

REFERENCES

1) E. Ozcan, M. Mısır, G. Ochoa, E.K. Burke, **A Reinforcement Learning Great-Deluge Hyper-heuristic for Examination Timetabling,** University of Nottingham, United Kingdom, 2009
2) E. Özcan, B. Bilgin, E.E. Korkmaz, **A Comprehensive Analysis of Hyper-heuristics** Yeditepe University Department of Computer Engineering, Intelligent Data Analysis, 12:1, pp. 3-23, 2008.
3) E. Özcan, E. Ersoy, A.S. Uyar, **Memetic Algorithms and Hyperhill-climbers**, Yeditepe University Department of Computer Engineering, 2007
4) E. Özcan, S. E. Uyar and E. K. Burke, **A greedy hyper-heuristic in dynamic environments**, Proc. of the 11th annual conference companion on Genetic and Evolutionary Computation COnference, WORKSHOP SESSION: Automated heuristic design: crossing the chasm for search methods, pp. 2201-2204, 2009.
5) E.K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, J. R. Woodward, **A Classification of Hyper-heuristic Approaches,** Automated Scheduling, Optimisation and Planning (ASAP) Group, School of Computer Science, University of Nottingham, UK.. 2009.
6) E. K. Burke, E. Hart, G. Kendall, J. Newall, P. Ross, and S. Schulenburg, **Hyper-heuristics: An emerging direction in modern search technology, Handbook of Metaheuristics** *(F. Glover and G. Kochenberger, eds.), Kluwer, 2003, pp. 457–474.*
7) E. Burke, G. Kendall, O. Ross, S. Petrovic, E. Soubeiga, **ASAP Project: An Investigation of Hyper-Heuristic Methods** University of Nottingham , *June, 2003* ***http://www.asap.cs.nott.ac.uk/projects/hh/hhbg.shtml***
8) R. Bai, E. K Burke, G. Kendall, & B. McCollum (2007) **Memory length in hyper-heuristics: an empirical study.** *In Proceeding of 2007 IEEE Symposium on Computational Intelligence in Scheduling (CISched2007), (pp.173-178)*
9) Sabar N. R., Ayob M. and Kendall G. (2009) **Tabu Exponential Monte-Carlo with Counter Heuristic for Examination Timetabling**, in proceedings of 2009 IEEE**.**Symposium on Computational Intelligence in Scheduling, 30 Mar - 2 Apr, 2009, Nashville, Tennessee, USA, pp 90-94
10) Djamila Ouelhadj & Sanja Petrovic, **A Cooperative Distributed Hyper-heuristic Framework for Scheduling** University of Nottingham *IEEE International Conference on Systems, Man and Cybernetics (SMC 2008).*
11) A. Lakhotia and M. Mohammed **Imposing Order on Program Statements to Assist Anti-Virus Scanners** University of Louisiana at Lafayette, The *Netherlands, November 2004, pp.161-171.*

TIME PLAN

#	Info	Title	Given Planned Work	Flag Status	# Predecessors	Expected Start	Assigned Resources	% Complete
0		Gantt Chart Report				01/10/2009		40%
1		Background study	10 days			01/10/2009		100%
2		Problem specification	10 days			01/10/2009		100%
3		Requirements specification	15 days			01/10/2009		100%
4		A simple hyper-heuristics implementation	15 days			14/10/2009		100%
5		Three low-level heuristics animated	10 days			26/10/2009		100%
6		Function draw	10 days			30/10/2009		100%
7		Prototype integration	15 days			09/11/2009		100%
8		Prototype improvement	10 days			23/11/2009		100%
9		Interim report	10 days			23/11/2009		100%
10		Redesign and specification improvement	10 days			01/01/2010		0%
11		Program integration	20 days			07/01/2010		0%
12		Testing and debugging	30 days			21/01/2010		0%
13		Final group report	20 days			05/03/2010		0%
14		Final individual reports	20 days			05/03/2010		0%
15		Open day	30 days			25/03/2010		0%
16		Project presentation	30 days			29/03/2010		0%