G52GRP Final Group Report

29th March, 2010



***A Visualization Tool for Selection Hyper-Heuristics***

## Supervisor: **Dr. Ender Ozcan** exo

## Group**:** **gp09-exo**

Jingqi Lao jxl29u  
Chao Zhang cxz09u  
Thomas Barton txb18u  
Ben Jenkinson bxj08u  
Alexander Jermstad asj08u

Contents

[Supervisor:  **Dr. Ender Ozcan** exo 1](#_Toc257720671)

[Group**:**  **gp09-exo** 1](#_Toc257720672)

[1. Introduction 3](#_Toc257720673)

[2. Background Information 3](#_Toc257720674)

[2.1. Hyper-heuristic 3](#_Toc257720675)

[2.2. What is **VSH**? 7](#_Toc257720676)

[3. Requirements Achieved 8](#_Toc257720677)

[3.1. Functional Requirements: 8](#_Toc257720678)

[3.1.1. Input 8](#_Toc257720679)

[3.1.2. Output 9](#_Toc257720680)

[3.2. Non-functional Requirements: 10](#_Toc257720681)

[4. Updated design and user interface 12](#_Toc257720682)

[4.1. State the problem description 12](#_Toc257720683)

[4.2. Construct H.H object & design patterns 12](#_Toc257720684)

[4.3. The user interface 14](#_Toc257720685)

[4.4. Use this tool 19](#_Toc257720686)

[5. Project Mangement 21](#_Toc257720687)

[References 22](#_Toc257720688)

**ii**

**ii**

# 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. The finished version of our program was extended to 5 low level heuristics, 3 selection methods, 3 move acceptances and 3 benchmark functions, which is more powerful than the first prototype.

# Background Information

## 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 the final version, our program contains **all of those 3 Selection Methods** above, hence it become more flexible to deal with different situations.

##### 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:

* **All Moves (AM):*****all*** *movements were accepted.*
* **Only Improving (OI):** *only accept the solutions that made* ***improvements****.*
* **Improving and Equal (IE):** *accept the solutions that made* ***no decrease.***
* **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)**:

(1)

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

* **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)***:*

(2)

*Where is the fitness improvement after the**iteration, D is the maximum number of iterations; N is an expected range for the maximum fitness change.*

Here we chose the first three move acceptance: **All Moves (AM), Only Improve (OI) and Improve and Equal (IE)**.

##### 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.*
* **Flip one bit:** *just randomly* ***flip only one bit*** *of candidate solution.*

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.*

We use all the low-level heuristics above in the project, which provide a lot of choices to the users.

##### The Benchmark Functions

One important component related to Hyper-Heuristic testing is benchmark function. In a sense, the benchmark functions consisted “problem domain” of the hyper heuristic methods. On the other hand, benchmark functions are also helpful to the algorithm optimization and evaluation (Ozcan, 2008). There are 3 benchmark functions in this project; and they all have simple but effective performance.

The first benchmark is defined as:

(1)

and we also have:

(2)

finally, the most complicated one is:

(3)

##### Hyper-heuristic Process

In a simple hyper heuristic, 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 1.1** shows, one candidate solution () 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 cannot meet the requirements (Ozcan et al., 2009).

**Domain Bearer**

**Candidate Solution** ()

**Select from low-level**

**& Apply ()**

**Low-level Heuristics**

**…**

**Heuristic Selection**

**If accepted, = ()**

**If rejected, =**

**Move Acceptance**

**Check satisfaction**

**Does meet the requirements?**

No

**Return best solution**

Yes

**Figure 2.1** *Operation of a selection hyper-heuristic framework based on a single point search*

And **Figure 2.2** 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 2.2**  *Pseudo code of a selection hyper-heuristic based on a single point search*

In this project, a **candidate solution** is represented by array contained **15** **binary bits**, which would be operated by the heuristic selection method; after that, it would be translated to one particular decimal value and sent to the comparing step, known as the move acceptance.

## 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 version **JAVA SE 6.0**. 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.

# Requirements Achieved

## Functional Requirements:

We have defined the functional requirements in the interim report, and after that we extended and modified them as below:

* 1. **Simulate** – the program can correctly simulate and implement the processes of a Hyper-Heuristic method.
  2. **Animate –** the program can demonstrate how Hyper-Heuristics work, which including dynamical graphs and the animations.
  3. **Draw curves** – enable the program to draw the curves in accord with the benchmark functions.
  4. **GUI** – program should contain a Graphical User Interface (GUI)
  5. **Generate result table –** implement a form of the Travelling Salesperson Problem using permutation representation.
  6. **Construct H.H.** – enable the user to choose the *selection methods* and *move acceptance* from a domain in the interface.
  7. **Define parameters** **–** enable the users to choose the range of low level heuristics, the benchmark functions from the list in the interface.
  8. **Set termination –** enable the users to set the termination situations for the hyper heuristic porcess
  9. **Control the processes** – enable the users of running, stopping and pausing operations upon the animation.
  10. **Inform** – the program shall inform the users about the hyper heuristic processes and give tips.

Our completed version has achieved **all the functional requirements** described above, which means, the program finished with functional integrity; on the other hand, it also welcomes the further modifications and extending.

### Input

The user should be able to choose the benchmark function to be operated upon and specify the parameters before initialising the hyper heuristics. This is in the form of list of buttons that displayed in a specified panel. Additionally Java has the capability to create classes on the fly thereby making it possibly for users to possibly enter their own functions.

And they should also be able to choose which hyper-heuristic selection methods (simple random, greedy random and choice function) and the move acceptances are applied before initialising the visualisation. And the program would automatically generate the H.H. method after that.

For the lower level heuristics algorithms such as the inverse, reverse, shift, flip and hill-climbing should be implemented. And the user shall be able to choose the range of the low-level heuristics that they want. The 5 low level heuristics were displayed in the panel left of the GUI; users can add one to the domain by click the button, and an additional click would remove it from the candidate heuristics.

The user should also be able to define the termination criterion so the program knows then to stop. This was 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.

A candidate solution is segment represented by **a series of binary values** which 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.

### Output

The function was appropriately 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]](#footnote-1)

The encoding and decoding of the data and the manipulation of the binary representation is to be illustrated as a part of the H.H. method in GUI. The binary representation was graphically represented and interacted 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. In the formal prototype we put the calculating and drawing process into two different threads, which, however, slow down the running after a large scale of calculations. Hence we tried a new technology that to put them together in one single thread, within wait-notify skill. And this brought a significant improvement to the running efficiency. 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 is to be stored so that it can be later viewed in a tabular form for easy viewing and possible statistical analysis if needed. And all resulting heuristically generated data should also be stored as the programs runs so that a table of results can be displayed for analysis at the end.

## Non-functional Requirements:

Different from the functional, non-functional requirements are more abstract; hence it is hard to control its outputs. On the other hand, because our developing module is prototyping, the requirements from the client or supervisor become quite important and changeable.

1. **Appearance of interface –** the GUI should be simple, acknowledgeable, powerful, friendly and usable
2. **Size of interface –** the interface could fit different size of screen
3. **Length of candidate solution** – the length should be long enough so that the hyper heuristic process could be more obvious
4. **Appropriate H.H. methods** – the *Selection Methods* and *Move Acceptances* should be powerful enough to explain the H.H. processes
5. **Benchmark functions –** the benchmark functions should be clear but complex (or effective) enough

Our finished interface has successfully met the first requirements, and the details would be shown in the design part. In fact, the JAVA default GUI can hardly fit the requirements; however, ***Jingqi Lao*** has found an effective way to improve its appearance, made the finished interface looked perfect. To make the interface acknowledgeable, we put the flow chart of H.H. Process in the initial animation panel, and put hints and explanations in the panel.

To the second requirement, we design the interface to two sizes: **general size** and **full-screen**. This is because of the nature of the background picture. We’ve tried to make the interface auto-justified while drag the frame of the interface; however the background picture can hardly fit the changed frame, so we finally fixed the size of the window, and prepared two groups of background pictures.

The length of candidate solutions was extended to **15** rather than 10 in the first prototype. The problem here is, with the increasing of length, the difficulty of representing has also increased; and in this case, we finally chose 15 as a result of compromise.

In addition, the current methods in hyper heuristic are successfully fit the animated simulation. However, it is critical that to apply **Great Deluge** method in the move acceptance. Actually, the implement of this algorithm is not such a hard thing; however we meet a problem while representing it.

Finally, the three benchmark function would provide a variety of performance to this visualisation tool, the program contain both simple and more complex functions. However, one imperfection existing while drawing those curves in animation panel, which is, the curves seemed ‘compressed’ and in some cases, the curves would overlap the coordinate axes.

##### Hardware & Software Requirements

1. **Programming language**: JAVA SE 6.0
2. **Project subversion**: Use of subversion to track changes.
3. **OS environment**: Primarily Windows, although due to widespread use and flexibility the Java can be used on many other operating systems if required.

Besides the requirement in problem description, 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 ***Eclipse*** is our development program of choice due to its powerful functionality and effective plug-in boards.

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 ([http://code.google.com/p/VSH/](http://code.google.com/p/vch/)) 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.

# Updated design and user interface

## State the problem description & implementations

Below is the problem description, as stated on the course website:

*“****Hyper-heuristics*** *can be defined as "methodologies to choose heuristics". There is a growing interest in hyper-heuristics as powerful tools in search and optimisation. A randomly generated initial candidate solution is improved iteratively using a set of low level heuristics in a simple choice hyper-heuristic framework. At each iteration, a given solution passes through two successive stages: heuristic selection and acceptance. The heuristic selection mechanism chooses and applies a low level heuristic to a candidate solution producing a new solution. Then, the acceptance strategy decides whether to continue the search process using the new solution or the one at hand. This project involves in designing and implementing a* ***Java applet*** *(or application) that* ***demonstrates*** *how a choice hyper-heuristic works on an optimisation problem that requires binary representation.”*

After analysing about that, we got the two crucial implementations:

1. *HH algorithm simulation module(heuristic selections, move acceptance, etc)*
2. *The Visualisation module (user interface, animation, etc)*

The first keyword was “hyper-heuristics”, which was described in *section 2*; and the algorithm was implemented by the OO-Design. Our final version contains an improved H.H simulation part with new-designed structures and extended functions.

On the other hand, since our project is a visualization tool, the interface should and need to be nice-looked, well-designed and human friendly. Fortunately, the final version of interface has been significantly improved and optimised (*see section 3.2*).

## Construct H.H object & design patterns

To implement the H.H algorithm is one thing, while to integrate it into the simulation animation is another. In the prototype last semester, we just nest the algorithm into animation and graphical interface section, which brought some confusion to the understanding of software structure. Fortunately, benefited from module G52OBJ, our newly designed structure became clearer and more reliable.

Firstly, we go through low-level heuristics, heuristic selection methods and move acceptance, for their properties, functions, and interactions. Due to the independence of low level heuristics, we defined one abstract class LowLevelHeuristic as the super class, which allows the further addition, deletion, or change. As the definitions mentioned in *section 2.1*, heuristic selection methods and move acceptance consist one specified heuristic method, while each of them is a property that need to be implemented; so the decision is, defining two JAVA interface to represent these two components, and then construct one new hyper heuristic algorithm via implementing these two interfaces and inheritance the *HyperHeuristic* class. Thus, we should also have a *HyperHeuristic* object to be inheritance due to its concrete features.  **Figure 4.2.1** shows the relations between these three components:

|  |
| --- |
| HyperHeuristic |

//defined parameters

|  |
| --- |
| HyperHeuristic-1 |
| HSPerformance**()** |
| MAPerformance**()** |

|  |
| --- |
| HyperHeuristic-2 |
| HSPerformance**()** |
| MAPerformance**()** |

|  |
| --- |
| <<interface>>  HeuristicSelection |
| HSPerformance**()** |
|  |

|  |
| --- |
| <<interface>>  MoveAcceptance |
| MAPerformance**()** |
|  |

**Figure 4.2.1** *the basic idea of implementations and inheritances for HH methods*

The above work significantly improved the clearness and reusing of code; however, things were more complex than it seemed to be. The following problems occurred while constructing different hyper heuristic algorithms; that is, for example, if we implemented a lot of concrete objects of either *MoveAcceptance* or *HeuristicSelection* interface, how many concrete *HyperHeuristic* classes should we create? The problem is that, to create all the *HyperHeuristic* classes is not only inefficient but impossible (because there maybe additional heuristic selection methods, etc). Thus, we brought the builder and factory design patterns to solve this problem.

Our secondary optimisation also began with analysing the features of current desing. And also, there are two reasons to choose builder pattern which met the criteria of using builder pattern:

* Complex object structures
* Data represented by many objects

Applying builder pattern means to implement one *HHDriector* object as well as the *HHBuilder* object; moreover, concrete builder will invoke a HeuristicSelectionFactory object and one MoveAcceptanceFactory object (which belongs to then factory pattern), and finally return the constructed hyper heuristic algorithm. And this process can be represent as that showed in **Figure 4.2.2**.

|  |
| --- |
| HHDriector |
|  |
| +Construct() |

|  |
| --- |
| HHBuilder |
|  |
| +BuildPart() |

+GetFactorys()

Client

|  |
| --- |
| HHAlgorithm |
|  |
|  |

|  |
| --- |
| MAFactory |
|  |
| +RetrunMA() |

|  |
| --- |
| HSFactory |
|  |
| +ReturnHS() |

**Figure 4.2.2** *diagram of HH method construction process & the combined design pattern*

When a client program calls the director, it would invoke the builder class to create one concrete Hyper-Heuristic implementation that can be directly used by the animation process. Furthermore, factory pattern would facilitate the choosing process for builder.

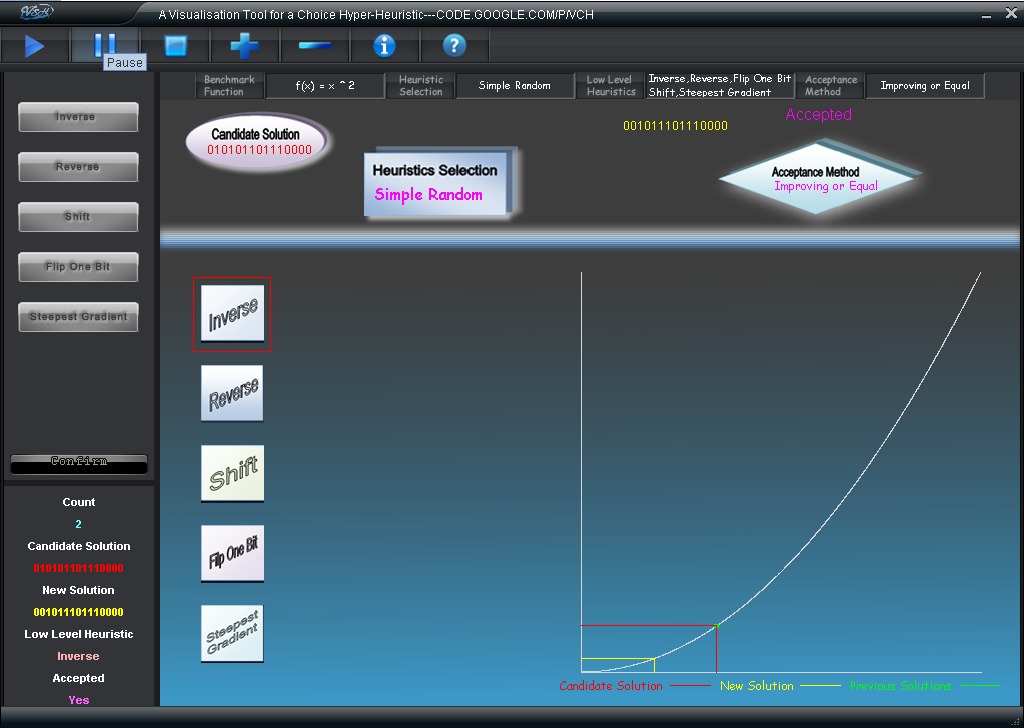
## The user interface

Our final design of GUI was a compromise between integration of core functions, size of animation, clearness of window and its appearance, as that showed in **Figure 4.3.1**. Here we put the logo in the title pane instead of words; right of the logo is the link connected to our project website: [http://code.google.com/p/VSH/](http://code.google.com/p/vch/).

Our user-interface is designed in such a way that contains 5 parts in the final pattern: one processing-control bar (**Figure 4.3.2**), one panel that display the parameters (i.e. benchmark functions, move acceptance, etc) and the related parameter-confirmation panel in the left side, one statement tracing panel in the bottom-left (**Figure 4.3.4**), and the most important panel – visualising panel (**Figure 4.3.7**). We will introduce these components individually.

**Processing-control panel**

**Running parameters displaying panel**



**Animation panel**

**Running states tracing**

**Change and confirm parameters**

**Figure 4.3.1** *one example of the user interface in processing*

Firstly, the control bar takes charge of the animation controlling, included **play**, **pause**, **stop**, **speed up**, **slow down**, **information** and **help** functions, and its example was given as following:



**Figure 4.3.2** *example of control bar*

Secondly, parameter-displaying bar shows the crucial information about the current hyper heuristic algorithm; which definitely are the benchmark function, heuristic selection method, low-level heuristic domain and the move acceptance. User shall click the button right of each cell and get the information from the left text-field.

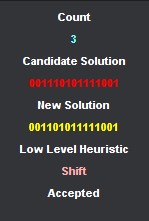




**Figure 4.3.4** *the parameter-displaying bar; the focus shows the structure of each cell, whose left is a button that to be clicked and the right is a text-field that shows the item details.*

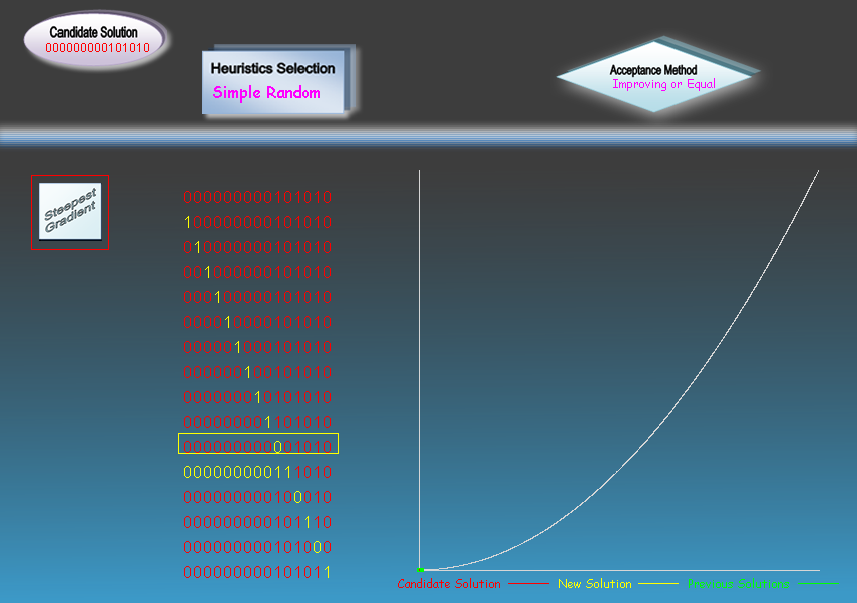
Clicking the buttons in the parameter-displaying panel would lead to a change of display to the panel in its right, which is known as the parameter-choosing panel. Users could change these settings from it, and then click “confirm” to change. To the components like Benchmark functions, heuristic selection and move acceptance, it allows only one selection at one time; and the selected item would appear a “faded” looking as that showed in **Figure 4.3.5.** Multiple choices were allowed for choosing low level heuristics, due to the principle of problem domain described before. What convenient here is that when click “confirm” button, all the parameter would be sent to the algorithm module.

In addition, the panel of running state was set up for optimising the running data tracing. As it showed below, this panel would record how many new solutions were generated, what the current candidate solution is, what the new one is and whether the new candidate solution was accepted.

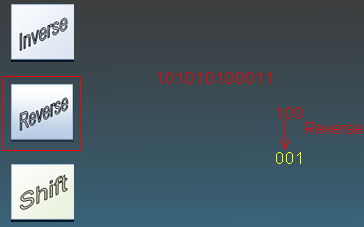
**Figure 4.3.5** *example of the selecting panel (left) and statement-tracing bar (right)*

Finally, animation panel included “the two crucial components of hyper heuristics”: heuristic selection algorithms and move acceptance process. Users can get the information about the current best candidate solution from the ellipse in top-left, and see how one candidate was operated via the low-level heuristic(s) from the left part; and then the right panel would draw the new generated solution onto the curve to show whether it was accepted. When the user clicks the “*pause*” button, this panel would be frozen until the user continues. Moreover, the demonstrating will stop if meet any terminal conditions, for example, the user’s stopping operation. The following **Figure 4.3.6** shows how these components were organized.



**Figure 4.3.6** *one real-time running example of the animation panel*

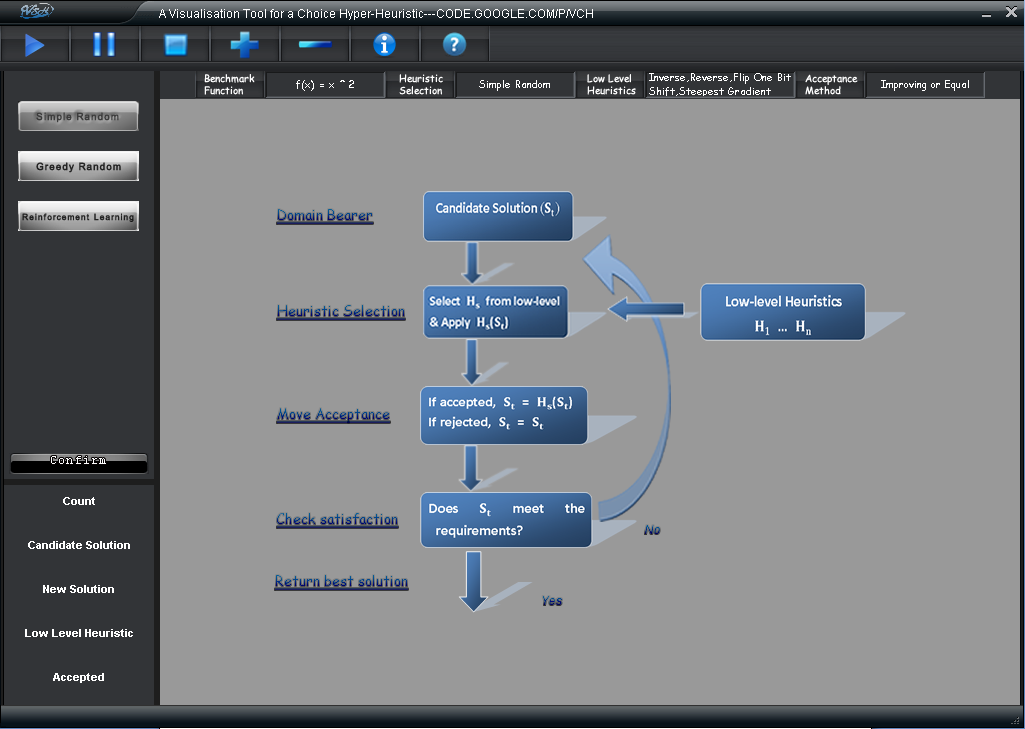
In the HS part, the rectangle boxed with red was the currently processing heuristic. Because this is a simulation under Hill-climber algorithm, the candidate solution boxed with yellow means the chosen one from all those solutions the low-level heuristic generated. For other low-level heuristics, system would demonstrate how a candidate solution was changed (**Figure 4.3.7**)**.**



**Figure 4.3.8** *one real-time running example of applying reversing heuristic*

## Use this tool

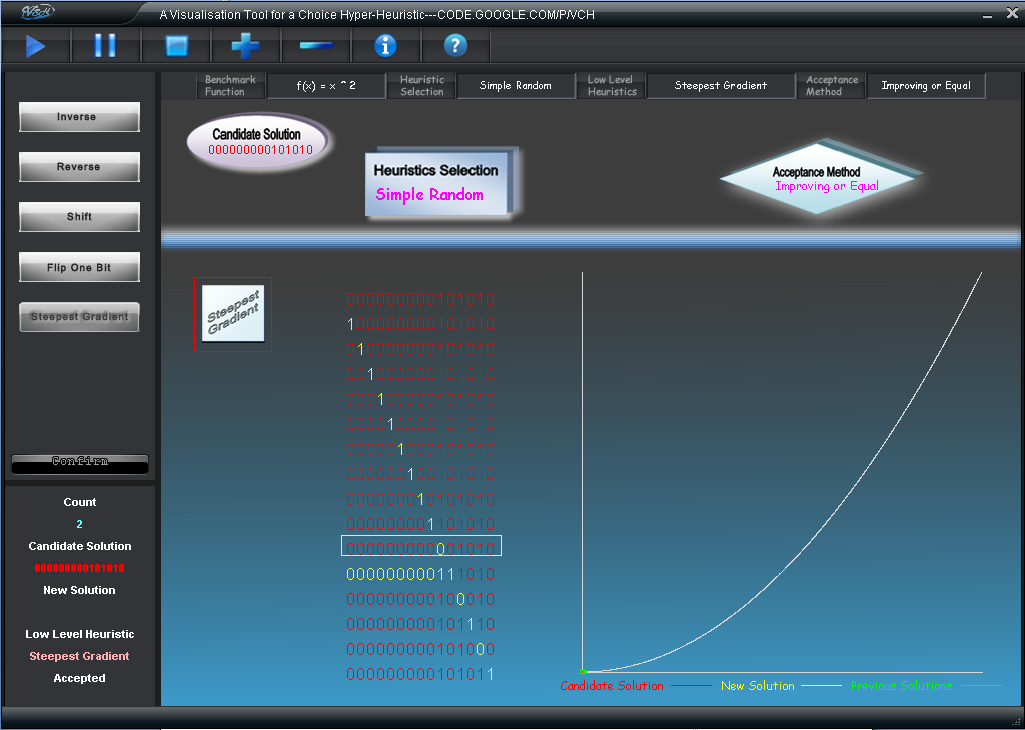
Our final version is not one simple extension of the first, so it would bring a significant difference while using it. As the following picture shows, our initial interface contains all the components mentioned above except the animation panel.



Instead of the animation module, there is a diagram of hyper heuristic process displayed. Thus, user can get some basic ideas and information about the H.H, and then he may begin the dynamic simulation.

As that described before, user can choose all the H.H components they preferred, and submit them individually by click confirm button. There are also some default parameters for a direct start:

* **Benchmark** -
* **Selection** - simple random
* **all the five low-level heuristics**
* **Move Acceptance** - improve and equal



During the processing, user can also control both the performance and behaviors of animation. Normal operations include begin/continue, pause and stop; while there are two additional functions: speed up and slow down. Represented by “+”and “-”, they can control the speed of animation via control its **refreshing rate**; in addition, every click would cause a **1- millisecond** change to the drawing thread.

# Project Mangement

# Final comments

All in all,

# 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***](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.*

* Updated design of the system and its user interface.
* Discussion on the implementation and testing of the system. This must include a list of all major system components, which of these were written by the group and where the others come from, and an overview of the developed [source code hierarchy](http://www.cs.nott.ac.uk/~nhn/G52GRP/G52GRPHandbook.html#software).
* Summary of what was achieved, referring to the stated requirements.
* Reflective comments on the success of the project, both from a technical and a project management perspective, including group working issues etc.
* An appendix giving a description of how the developed system was tested (test cases, example outcomes, etc.)
* Minutes from all formal meetings (appendix).

A user-manual (if appropriate) should be included as an appendix. Excerpts of the developed code can be included in the report for illustrative purposes, but any lengthy excerpts should go into the appendices.

**Two** printed copies of the final group report should be submitted as well as an electronic copy in [PDF](http://webservices.cs.nott.ac.uk/intranet/content/view/51/82/) format (one group member submits on behalf of the group). The front page of the report should include:

**Appendix**

* + - Formal meetings
    - Structure of the code
    - Testing examples
    - User-manual

# Appendix

1. http://www.w3schools.com/browsers/browsers\_display.asp [↑](#footnote-ref-1)