**G52GRP** Interim Group Report

29th November 2009

***A Visualisation Tool for a Choice Hyper-Heuristic***

*Group***gp09-exo**

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

*Supervisor*  
**Dr. Ender Ozcan** exo

Contents

[Problem Description 3](#_Toc247624231)

[Background Information & Research 3](#_Toc247624232)

[Heuristics 3](#_Toc247624233)

[Hyper-Heuristics 4](#_Toc247624234)

[What is VCH? 4](#_Toc247624235)

[Requirements Specification 5](#_Toc247624236)

[Functional Requirements: 5](#_Toc247624237)

[Non-functional Requirements: 5](#_Toc247624238)

[Initial Design 6](#_Toc247624239)

[Key implementation 6](#_Toc247624240)

[Discussion of Problems 7](#_Toc247624241)

[Technical Issues 8](#_Toc247624242)

[Non-Technical Issues 8](#_Toc247624243)

[Timeplan 10](#_Toc247624244)

[Appendix 10](#_Toc247624245)

# Problem Description

# Background Information & Research

## Heuristics

Finding a general solving algorithm is not always efficient: for example, some problems have a very completed methodology which would take a lot of time to implement but not widely-used. On the other hand, although there are many problem-solving computer algorithms, they are too knowledge-intensive to implement for most potential users. In such a case, people use heuristics as a substitution.

A single heuristic is an algorithm with a candidate solution and some constraints. And one example of a heuristic method is described with the following pseudo code:

|  |
| --- |
| BEGIN  INITIALIZE Candidate-solution  WHILE (Candidate-solution !satisfy Constraints)  Temp-solution 🡨 Heuristic-Algorithm(Candidate-solution)  IF (Temp-solution isBetterThan Candidate-solution)  Candidate-solution 🡨 Temp-solution  RETURN Candidate-solution  END |

The idea of heuristic is, there was randomly generated a candidate solution which is represented by a array of *binary value* called domain bearer, then the heuristic algorithm would operate the domain bearer to generate a new candidate solution, then compare them through the requirements(or constraints) and choose the better one. Finally, if the programme meet the constraints and stopped then the current candidate solution would be returned.

Furthermore, some basic ideas of those heuristic algorithms can be represented as:

* **Flip**: since every bit in the domain bearer is a binary value, it is efficient to flip one or several of them to opposite values. And also, the number of flipped bits is flexible.
* **Swap**: this means to choose two bits in the domain bearer randomly and swap them as a pair; and there also can be one or several these pairs.
* **Combination:** that means to use both the **flip** and **swap** methods in one heuristic.

Usually, heuristics is easy to implement and maintain, so it can be used to solve the problems such as data-mining, cutting, packing, and scheduling; and it also has been used in the areas such as anti-virus technology. However, the individual heuristics does not always work well: for example, sometimes it can make serious errors for some special data structures; and on the other hand, one single heuristic cannot always be efficient.

## Hyper-Heuristics

Since those existing problems of heuristics influenced its performance, hyper-heuristics was developed as an improvement. In general, a hyper-heuristic method is always described as “a heuristics to choose heuristics”, who operate on a specified domain of existing heuristics instead of the candidate solutions. Moreover, hyper-heuristics algorithms also have candidate solutions and constraints; but different from the low-level heuristics, hyper-heuristics would decide how to use low-level heuristics to generate new potential solutions rather than generate those new candidate solutions (or domain bearer) itself.

There are several heuristics-selection methods, such as:

* **Simple Random (SR)**, which select the low level heuristics randomly.
* **Greedy Random (GR)**, which apply all the low level heuristics to the candidate solution and choose the one who generated the best solution.
* **Choice Function (CF)**, which records the performance of each low level heuristics and compare them with several criteria.

Through these methods, one hyper-heuristic process would finally decide which low-level heuristics are more appropriate under the current situation. 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. The pseudo code of hyper-heuristic can be probably described as:

|  |
| --- |
| BEGIN  INITIALIZE Candidate-solution  INITIALIZE Low-level-Heuristics[n]  WHILE (Candidate-solution !satisfied Constraints)  random 🡨 RANDOM()  a 🡨 random%n  Temp-solution 🡨 Low-level-Heuristics[a](Candidate-solution)  IF (Temp-solution isBetterThan Candidate-solution)  Candidate-solution 🡨 Temp-solution  RETURN Candidate-solution  END |

## What is VCH?

VCH is to be the name of our application. VCH is literally defined as “Visualization for Choice Hyper-heuristic”. As its name suggested, this project is a visualization tool to facilitate people’s understanding about hyper-heuristics. VCH 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 VCH 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, VCH will be an open source project to response the increasing interest for hyper-heuristics at present. In general, VCH 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 VCH would bring optimistic benefit either as free software or commercial software.

# Requirements Specification

Requirements specification for the system to be built (agreed between the group and supervisor).

## Functional Requirements:

1. Implement the heuristics as well as the hyper-heuristics.
2. Ability for the user to select specific function for their requirements.
3. A number of standard heuristic choices to be available, at least three to begin with, with the option of adding additional ones if need be.

## Non-functional Requirements:

Programming language: JAVA

OS environment: mainly Windows

Project website: <http://code.google.com/p/vch/> subversion

\*use some diagram (use-case, dataflow, sequence….)

# Initial Design

* Software and hardware to be used, along with reasons for those decisions.
* Initial design of the proposed system and its user interface.
* Results of any initial implementation steps/prototyping.
* Results of technical research into suitable platforms, tools, technologies, algoritims, data structures, etc.

## Key implementation

* Implementation for a specific hyper-heuristics
* A base-user interface
* A process-display interface
* Attached functions such as show the history of the chooses each time, set low-level heuristics, set objective function, set the range of domain bearer, etc.

# 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 number 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 visualistic 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.

## Technical Issues

* Draw the dynamic diagram of the function
* Problems with the extend functions such as set the low-level heuristics by the user, read objective functions, set domain bearer, extended hyper-heuristics algorithms, etc.
* To organise and optimize the GUI

## Non-Technical Issues

* E.g. Individual work

# Timeplan

# Appendix