1. **Background Information and Research**
   1. **About 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. One single heuristic is an algorithm with a candidate solution and some constraints. A heuristic can be described with the following pseudo code:

BEGIN

INITIALIZE Candidate-solution

WHILE (Candidate-solution !satisfied 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 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. But the individual heuristics does not always work well: sometimes it can make serious error.

* 1. **About 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 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.

In this project, we will mainly 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

* 1. **What is VCH?**
* A visualization tool
* Can give a direct and clear expression of hyper-heuristics to the users
* There is increasing interest about hyper-heuristics
* No such a software yet
* market research

1. **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….)

1. **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, algorithms, 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.

1. **Problems**

(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 problems)

E.g. Individual work

When the programme becomes more and more complicated, it become harder that to catch up with others’ thought.

1. **Time Plan**
2. **Appendix**



Time plan

Assign our work and role

Research current solutions

Write Requirement Specification

Write design document

Prototype user interface

Write prototype GUI implement

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tasks | Predecessor | Duration | Exective | Notes |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |