

Unsupervised Problem Solving by Optimising through Comparisons

Kieran Greer,
Distributed Computing Systems, Belfast, UK.
Email: kgreer@distributedcomputingsystems.co.uk

Abstract: This paper describes the implementation and functionality of a centralised problem solving system that is included as part of the distributed ‘licas’ service-based network framework. While the framework can include autonomous and distributed behaviour, the problem solving part can perform more complex centralised optimisation operations and then feed the results back into the network. The problem solving system is based on a novel type of evaluation mechanism that prefers comparisons between solution results over maximisation. This paper describes the advantages of that and gives some examples of where it might perform better, including some more cognitive possibilities.

1 Introduction

This paper describes the implementation and functionality of a centralised problem solving system that is included as part of the distributed ‘licas’ service-based network framework [11]. The problem solver uses a hyper-heuristic with a matching process at its core [8]. The algorithm briefly works as follows: The solutions and the problem datasets are placed into a grid and then a game is played to try and optimise the total cost over the whole grid, using a randomising process. This is done by matching values across rows or columns. For the current algorithm, to match any two entities, the algorithm must remove any rows or columns that are in-between the two to be matched. The algorithm therefore also removes solutions as well as trying to keep other ones. While matching does not maximise, the algorithm tries to maximise the overall score and therefore prefers to match higher valued solutions over lower valued ones. It would work particularly well for problems that might require some sort of symbolic evaluation instead of a numerical one. In that case, an exact evaluation of what value is ‘better’ might not be possible and so some sort of matching evaluation would be required instead.

With regard to licas, the purpose of the problem solver is to try to combine distributed sources of information through heuristic search, to generate more meaning over the information sources as a whole. The information to be combined can be partial in nature, or change over time. It is therefore difficult to evaluate accurately what pieces of information would belong together. Additional data could make an evaluation better or worse and so some sort of comparison or matching process might be preferred. The hyper-heuristic is able to control this matching process and feed the results back into the network, so that the distributed information sources that are most likely to contain related information can be combined in an efficient and accurate manner. The problem solver itself can actually solve problems using either a hill-climbing approach or the new matching process, but this paper is concerned with the matching process only. This paper describes the new heuristic and considers the system integration details in particular. It also considers the scenarios where the new heuristic would work particularly well and describes how this differs from the more traditional clustering methods. The conclusions will also try to tie this in with a more cognitive model, which is something that the author is currently working on.

The rest of this paper is organised as follows: Section 2 summarises the main features of hyper-heuristics and why they are useful. Section 3 discusses the problem of variable selection some other heuristic algorithms that are commonly used for categorisation. Section 4 describes the new hyper-heuristic framework in more detail and compares it with a more cognitive model. Section 5 describes some tests results that show the heuristic working in an unsupervised manner, while section 6 gives some conclusions on the work.

2 Hyper-Heuristics

Most of this section has been taken from the literature review on hyper-heuristics [7]. Many real-world problems that require some level of intelligence are difficult to solve. If all of the potential solutions can be realised, then the best one will be available and can be selected. Often, there are too many potential solutions and so heuristic search is required to estimate what the best solution might be. This is where intelligence is required, to help the heuristic

to select what potential solutions should be explored further. The problem solving process is restricted to the information that is available in any potential solution. The search process can then reveal more information that was not originally known, but if the search space is very large, any solution will still only be an estimate or approximation of the true answer. The choice of heuristic that should be used to make this approximation then becomes very important, as different heuristics can evaluate certain concepts better than others. The main goal of hyper-heuristics is to develop algorithms that are more generally applicable. The paper [4] is a recent survey of hyper-heuristics. As noted in [1], a heuristic can be considered as a 'rule of thumb' or 'educated guess' that reduces the search required to find a solution. Allowing for different types of evaluator can give a more complete picture of what the correct evaluation is. While a single heuristic can get stuck in locally optimal solutions, if several heuristics are compared, then a more universal picture can be obtained. This can lead to better solutions somewhere else in the search space. The main drawback is that hyper-heuristics need to be configured, or fine-tuned with the correct parameter settings, for them to work well. This is often a manual trial and error process.

The introductory sections in [4] make some interesting points. They note that hyper-heuristics were initially developed as 'heuristics to choose heuristics'. They are not intended to operate on the problem data itself, as a meta-heuristic would do. Instead, they operate on the heuristics that do evaluate the data directly, to select what solutions should be considered at each evaluation stage. With the incorporation of genetic programming, there is also the option of using 'heuristics to generate heuristics'. The hyper-heuristic can select which heuristics are mutated, or changed, for the next evaluation stage as well. The problem solving system currently only uses heuristics to generate new heuristics, inside a genetic programming framework. Using heuristics to generate new heuristics not only involves selecting heuristics for the next evaluation stage, but also the ability to alter them resulting in a new heuristic not previously available. These sorts of problems can be solved by generating random solutions as part of a search process. Each solution is then changed in some way to improve it, until an optimal solution is obtained. The next stage of each search process is then directed by optimising the new solution set. Genetic programming itself is not inherently a hyper-heuristic, as it can also be used to represent the problem solutions.

The hyper-heuristic framework is more likely to use genetic programming principles to mutate existing solutions to generate better ones.

2.1 Hyper-Heuristics Related to the Current Project

This section looks at more specific examples that would be directly related to the current project. The paper [15] describes a hyper-heuristic framework that is self-organising by using reinforcement learning to order potential solutions. In this case, they apply each heuristic to a candidate solution to determine how it changes. If the solution changes positively, then the change is accepted. This is a perturbative approach, but includes both heuristic selection and heuristic creation or mutation. The low-level heuristics are evaluated through reinforcement learning into positive or negative ones. The positive ones are more likely to produce a positive evaluation. They are also then placed into a category of hill-climbing heuristics. The negative ones are placed into a category of mutational heuristics, which can then be changed to produce new ones.

The paper [13] would classify the new hyper-heuristic as an evolutionary mechanism, because it contains a stochastic element and also allows for solution mutations. They note that while this can lead to mistrust in its use, it is also a more natural or bio-inspired way of solving a problem. A key factor with this hyper-heuristic, or for comparisons with other ones, is where in the process the randomness is applied. In most cases, the low-level heuristics can be changed in a random way, to generate new solutions that are then evaluated by the hyper-heuristic for improvements to the current solution. In this case, the randomness applies to the evaluation process of the hyper-heuristic itself. Their own XCS algorithm uses the problem state to determine what heuristic to apply at some stage of the problem solving process. It also however, chooses randomly which problem to solve at each step; and randomness is also used as part of the problem solving process itself. They also write:

‘The key idea in hyper-heuristics is to use members of a set of known and reasonably understood heuristics to transform the state of a problem. The key observation is a simple one: the strength of a heuristic often lies in its ability to make some good

decisions on the route to fabricating an excellent solution. Why not, therefore, try to associate each heuristic with the problem conditions under which it flourishes and hence apply different heuristics to different parts or phases of the solution process? The alert reader will immediately notice an objection to this whole idea. Good decisions are not necessarily easily recognisable in isolation. It is a sequence of decisions that builds a solution, and so there can be considerable epistasis involved - that is, a non-linear interdependence between the parts. However, many general search procedures such as evolutionary algorithms and, in particular, classifier systems, can cope with a considerable degree of epistasis, so the objection is not necessarily fatal.'

This appears to state that it is not always obvious or clear when a particular solution should be selected. As with nature, some level of randomness can be used to make an incorrect or imperfect selection process more robust. The paper [2] describes a hyper-heuristic that also uses a simulated annealing approach for selecting which solutions to search further. As written previously, the purpose of simulated annealing is to add a stochastic element, to make the heuristic more generally applicable. Their algorithm adopts a simulated annealing acceptance criterion to alleviate the shortcomings of hill climbing or exhaustive search. Their algorithm also uses stochastic heuristic selection mechanisms instead of deterministic ones, which has been shown to be superior for some evolutionary optimisation problems [17]. They evaluate a heuristic to get a score and then use simulated annealing to generate a probability threshold that the score must then match. The better solution score is more likely to meet the selection criteria. The thesis [18] is also very interesting and tries to develop a hierarchical clustering algorithm that might be more applicable to the aims of the current project. It also describes other types of categorisation and matching functions not listed here.

3 Variable Selection

Before any information source can be analysed, it has to be determined what the most important features of that source are. These features are then used to classify or evaluate the source. This can be a difficult task because a source could be composed of thousands of

different features and so it is important to recognise the most important ones that make it different, or the same, as other sources. The paper [9] describes mechanisms for selecting variables or features from large repositories of unstructured data. These can act as filters, to select what variables or features from a potentially large dataset should be used to actually classify the dataset. It also notes that the most relevant variables are not necessarily the most useful when building a predictor or evaluator and so it is not simply a statistical matter of selecting the most popular variables. It is also possible to select subsets of variables that together have good predictive power. The papers [3] and [10] discuss the difference between relevant and useful variables. In [3] they describe that at a conceptual level, one can divide the task of concept learning into two subtasks: deciding which features to use to describe the concept and deciding how to combine those features. The selection of relevant features, and the elimination of irrelevant ones, is one of the central problems in machine learning. Algorithms can range from something like nearest neighbour, which can calculate attribute distances based on all available information, to weighted feature selection, or even techniques for learning logical descriptions. The definition of relevance can mean [3]:

1. Relevant to the target concept.
2. Strongly or weakly relevant to a sample or distribution.
3. Relevant as a complexity measure.
4. Incremental usefulness.

Relevant to a target concept means that a change in the variable's value can change its classification allocated by the target concept. Relevant to a sample or distribution is the same, except for the fact that the variable is then required to be part of the sample, as well as relevant to the target concept. These notions are more important for an algorithm that is deciding which features to keep or ignore. Relevance to just the target concept can sometimes be used to try and prove the algorithm itself rather than its evaluating results. The new heuristic has potential for feature selection. In particular, for selecting the most appropriate values for certain variables or features from distributed or partial information. If there are several sources sending partial information, then the features or concepts that they describe can be evaluated by the individual sources differently. These evaluations can be better or worse than the true value and might vary around some distribution or mean of the true value. Each group of features or concepts can also be different. Sections 4 and 5

describe how the hyper heuristic can be used to try to select the best value for unknown variables in an unsupervised manner.

3.1 Feature Selection Equations

Existing feature selection usually involves categorising or clustering into distinct groups. This is also often a supervised process, with known clusters being used to train the classifier, so that it can then recognise these clusters in other datasets as well. There are a number of existing equations that can be used to categorise data. Most of these actually belong to clustering algorithms that would try to measure how similar two data objects are. This is not actually what the matching process described in this paper is trying to do and so it already shows a possible difference in the use of the new hyper-heuristic algorithm. Some of these equations are as follows:

3.1.1 Euclidean Distance

This is a linear measurement that is one of the simpler classification metrics. It sums the difference between all attributes of two different input objects to determine how similar they are to each other. The equation can look like [12]:

$$d_{12} = \sum_{j=1}^k (\pi_{1j} - \pi_{2j})^2 \quad \text{where } d \text{ is the distance and } \pi_1 \text{ or } \pi_2 \text{ are the input objects.}$$

3.1.2 Kullback-Leibler Information Divergence

The Kullback-Leibler information divergence is a measure of the difference between two probability distributions. As described in Wikipedia: it can be used as a distance metric and measures the expected number of extra bits required to code samples from P when using a code based on Q , rather than using a code based on P . The equation can look like [12]:

$$D(p || q) = \sum_{j=1}^n p_j \log\left(\frac{p_j}{q_j}\right) \quad \text{for } p = (p_1, p_2, \dots, p_n) \text{ and } q = (q_1, q_2, \dots, q_n)$$

So this measures distances between probability distributions instead of single dataset values and so is probably more useful measuring the distance between the created cluster groups than the individual data objects.

3.1.3 Jaccard Coefficient

The Jaccard coefficient measures the similarity between datasets. It is a set theoretic measure and can be defined as the intersection of the datasets divided by the union of the datasets. For example:

The Jaccard coefficient between T_1 and T_2 can be defined as $\frac{|T_1 \cap T_2|}{|T_1 \cup T_2|}$

Dividing by the intersection scales the result between 0 and 1. If the two sets are the same, for example, the equation computes to the value 1. If there are no elements the same, then it computes to 0. The Jaccard distance measure is then the opposite of this and measures the dissimilarity between two datasets. One drawback of the Jaccard coefficient is that it does not really consider negative input as well.

3.1.4 Rocchio Classifier

The Rocchio classifier [16] is a similarity-based linear classifier that considers both positive and negative input. The equation [5] can be described as: given a training dataset T_r , the Rocchio classifier directly computes a classifier $\vec{c}_i = \langle w_{1i}, w_{2i}, \dots, w_{ri} \rangle$ for category c_i by means of the formula:

$$w_{ki} = \beta \cdot \sum_{\{d_j \in POS_i\}} \frac{w_{kj}}{|POS_i|} - \gamma \cdot \sum_{\{d_j \in NEG_i\}} \frac{w_{kj}}{|NEG_i|}$$

where w_{kj} is the weight of dataset t_k in document d_j , and *POS* or *NEG* means that document d_j contained in the training dataset, does or does not belong to the classifier category c_i . A classifier built using the Rocchio method rewards the closeness of a test document to the centroid of the positive training examples and its distance from the centroid of the negative training examples.

3.1.5 Information Theoretic Ranking using Probability Densities

The paper [9] gives an example of a ranking equation that can be used with information theoretic criteria. Information theory has to do with data compression and also loss of information through noise. The following is an example of the sort of equation that would be used to do this. The following ranking equation can be used to determine the probability of one variable being associated with another one, or some target concept. This relies on probability densities that can be unknown or hard to estimate. With discrete or nominal variables however, the equation can be written as:

$$I(i) = \sum_{x_i} \sum_y P(X = x_i, Y = y) \log \frac{P(X=x_i, Y=y)}{P(X=x_i)P(Y=y)}$$

Where $P(x_i)$ is the probability density for variable x at time i , $P(y)$ is the probability density for target y and $P(x, y)$ is the probability of them occurring together. The value is therefore a measure of the dependency between the target and the variable in question.

3.1.6 Information Gain

The paper [14] also describes information gain. With this method, both class membership and the presence/absence of a particular term are seen as random variables, and one computes how much information about the class membership is gained by knowing the presence/absence. Indeed, if the class membership is interpreted as a random variable C with two values, positive and negative, and a word is likewise seen as a random variable T with two values, present and absent, then using the information-theoretic definition of mutual information we may define Information Gain as:

$$IG(t) = H(C) - H(C|T) = \sum_{t,c} P(C=c, T=t) \ln \left[\frac{P(C=c, T=t)}{P(C=c)P(T=t)} \right].$$

Here, t ranges over {present, absent} and c ranges over {c+, c-}. As pointed out above, this is the amount of information about C (the class label) gained by knowing T (the presence or absence of a given word).

3.1.7 Feature Selection based on Linear Classifiers

As described in [14], both SVM and Perceptron, when used as linear classifiers, output predictions of the form:

$$\text{prediction}(x) = \text{sgn}(w^T x + b) = \text{sgn}(\sum_j w_j x_j + b).$$

Thus, a feature j with the weight w_j close to 0; has a smaller effect on the prediction than features with large absolute values of w_j . The weight vector w can also be seen as the normal to the hyperplane determined by the classifier, to separate positive from negative instances. Thus we often refer to the procedure as 'normal based feature selection'. One speculates that since features with small $|w_j|$ are not important for categorization they may also not be important for learning and, therefore, are good candidates for removal. A theoretical justification for retaining the highest weighted features is to consider the feature important if it significantly influences the width of the margin of the resulting hyperplane. This is described further in the paper.

4 New Stochastic Hyper-Heuristic Framework

The new hyper-heuristic framework was first introduced in [8]. Essentially, it uses a matching evaluation over a maximising one. However, it also tries to maximise the matching score and so will favour higher scoring matches over lower scoring ones. It also uses a randomising procedure to place all potential solutions in a grid, where any solution can be placed in any position. The matching process then matches solutions by removing any solutions that are between them in the grid and grouping the matched ones together for evolving into a new solution. This process therefore also removes solutions as well as evolving the potentially better ones and in that sense is self-regulating. There are however other parameters that need to be set during configuration.

This sort of process might be preferable for the feature selection problem that has been described, or the heuristic might, in general, be more suitable for a different class or type of problem. The clustering heuristics that have been described are intended to categorise similar datasets through a matching process, where the datasets with the most similar characteristics are matched together. This therefore requires several category types and then several datasets belonging to each category type. The aim of the new heuristic is to try to 'optimise some evaluation' through a similar matching process. This evaluation applies to the data or problem set more as a whole than its parts. It is more similar to a neural network trying to realise a single evaluation function that maps its input to its output than several evaluation functions that each map a different thing. It is not a case of categorising the different datasets into similar groups, but rather, trying to evolve the solutions in the most robust way, in order to arrive at the optimal solution value for a problem. The whole search space belongs to the same single problem. Some of the best potential solutions can be removed, if it means that other ones match better as a result. Each matching phase is probably also associated with an evolution of the related solutions, to produce offspring that would then more closely match or solve the problem. The correct evolutions are not known beforehand and so it is consistency through matching that is used to decide which solutions to evolve.

4.1 Implementation Details

The problem solving framework has now been implemented as part of the licas system [11]. Licas provides a framework for building distributed service-based networks of information sources, for example. The individual services can self-organise through a novel mechanism and can also display autonomous behaviours. The self-organising mechanism is relatively lightweight and essentially stores links from one service or node to another represented by a weight strength. The links are made more accurate through a path description made up of metadata or concepts that relate to the association between the linked services or nodes. So there is not a great deal of computation that takes place to create these links. It is also a highly distributed solution, where one link or association does not have to relate to any

other one. It is built up purely through the feedback of the system use and not any centralised or knowledgeable algorithm.

The problem solver is then more of a centralised solution. It can be sent the information from the network sources, use heuristic search and evaluations to perform a more complex problem solving operation and then feed these results back into the network, to allow the sources to update themselves through the more complex or sophisticated procedure. The problem solver can also be used by itself without the network, to solve any sort of problem using genetic algorithms, where the framework is very extendible with the user's own classes. Figure 1 is a schematic of the general problem solving framework.

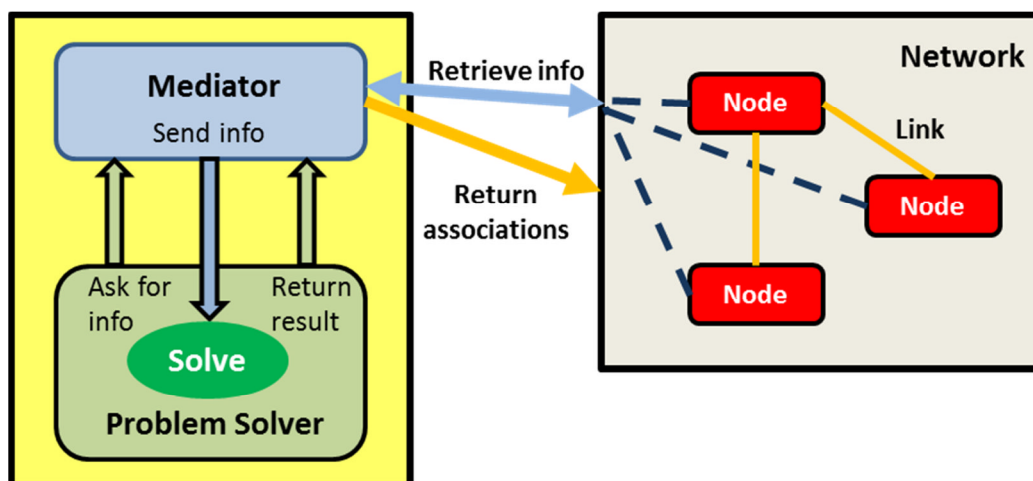


Figure 1. General Problem Solving framework.

The problem solving is performed locally and not distributed throughout all of the network services or nodes. An information mediator can be used to retrieve the information from the distributed sources. This is then sent to the problem solver, which creates solutions of the specified type and clusters the sources or solves the problem as best it can. The resulting clusters can then be turned into dynamic links and used to update the network structure. The information mediator can communicate directly with the services running on

a network and the results viewed in a GUI, or the problem solver can be used by itself without any network or GUI.

4.2 As Part of a Cognitive Model

The author also has an interest in developing a more cognitive model. While the distributed part of this model has been written about recently [6], there is also possibly a centralised part to the brain function as well. This has been noted and could even be thought of as the consciousness part. If the neurons in the brain form in a purely mechanical way, without any real intelligence, then a more centralised and intelligent part that can interpret the firing neurons might be required. It is also known that the human brain does not store information only once, but duplicates it in different areas. When the neurons fire, either a stronger signal from one set of neurons, or possibly from duplication of the firing pattern in different areas, could determine what the correct thought or decision is. If several areas fire at the same time with parts related to the same problem, this would produce a stronger overall signal and could help to suggest what the correct interpretation is. This is especially useful if the search process is not particularly structured or constructive, that is without a clear search path, for example; when the duplication can help to confirm what the correct interpretation is. This also means that a matching process is more attractive as the decision maker. If the decision is based on a signal strength from one place only, then the actual measurement of this has to be interpreted slightly more accurately and the neuron might still require some level of intelligence to change the strength. Although, more inputs to the neuron would also simply change this. If the signal is stronger through duplication however, then a more simple matching process can possibly derive the same conclusion. The author therefore finds the hyper-heuristic that uses comparisons more attractive as the centralised component for his current cognitive model. A decision would be formed through more neuronal areas firing relevant signals when the input is received. This would not require a highly structured search process, but instead, would be interpreted by a more complex or intelligent component that could recognise the same firing pattern in different places.

5 Testing

A set of tests have been created to test the problem solving framework for usefulness as a feature selector. The tests are designed mainly to determine if the problem solving process is constructive – that is – it is doing something in an intelligent manner and not simply trying to match things in a random way. The test is as follows: datasets of concepts or features are created. Each dataset has a specified number of features in it, selected from a larger number of features. Each dataset therefore has a randomly selected subset of all of the features. Each feature is assigned a score. The scores are also randomly assigned, but they are distributed around some mean value. Therefore, there is a preferred or more common mean value, with a distribution of values either side of it. Each feature therefore has a random score, but more commonly, this score will be closer to the mean value of the distribution. The matching process therefore should match the more common scores and therefore choose those related solutions for evolving further. It would also naturally select certain groups of features for evolving as part of the process.

The evaluation and evolution process is simply an intersection of the dataset features. During the comparison, if both datasets have certain features in common, their distribution from the mean for those features only is calculated and this is taken to be the similarity score. It is also only those features that are then included in an evolved solution. The evolution algorithm evolves using an intersection and so the sums for single datasets will generally be larger than for evolved sets. A hill-climbing approach did not work as well for this type of problem, because maximising would prefer the original datasets with all of the features over an evolved one with only subsets of those features. The hyper-heuristic using comparisons did not have this problem. Other evaluation equations were also tried, such as Euclidean, Jaccard or similarity, see section 3.1, but the intersection of features proved to be a more suitable approach.

5.1 Example test data

One example test was as follows: 30 datasets were created randomly. Each dataset contained 20 random features selected from a possible 40 features. Each feature was assigned a random score taken from the following distribution in the range 1 to 19:

0.05:3 - 0.075:3 - 0.175:3 - 0.4:1 - 0.175:3 - 0.075:3 - 0.05:3

This means that there was a mean value of 10 that would be selected 40% of the time. There was then a distribution either of values 11 – 13 or 7 - 9 that would be selected 17.5% of the time. The next distribution was 14 – 16 or 4 – 6, selected 7.5% of the time. Finally, the distribution of either the values 17 – 19 or 1 – 3 would be selected 5% of the time. A random process might be expected to produce an average or 50% deviation from the true mean value when selecting variable or feature values. If considering just the difference from the mean value of 10, if 10 values are selected then this could be calculated on average as: the value 10 is selected 4 times, 8 is selected 3.5 times, 5 is selected 1.5 times and 2 is selected 1 time. This gives a total average of:

$$40 + 28 + 7.5 + 2 = 77.5 / 10 = 7.75$$

This is a difference of 2.25 from the true mean value of 10. The success of the test was calculated as follows: Some of the final clusters contained the original solution datasets, while others contained the evolved solution datasets. If the process is constructive or intelligent, then the evolved solutions should be closer to the mean value than the original ones. Therefore the stats calculated the difference to the mean for the original solutions that were part of the final solution set and also the evolved solutions that were part of the final solution set. The original solutions that were included actually had an average difference of possibly around 2.27, which is close to the random average value. The evolved solutions had an average difference of possibly 1.57, which is a 30% improvement on the random or single datasets value. While this might not be able to prove something, it shows that in this set of tests the process is constructive and is doing something intelligent. It is unsupervised and therefore all of the evaluations are taken from the information that is presented at the time. Using the intersection method as the evaluator and evolver is not necessarily the best, so possibly improvements could be made there as well.

6 Conclusions

This paper has described a new type of problem solving framework based more on corroborative or matching evidence than on purely maximising some function. The aims of the current project are to develop a hyper-heuristic framework for evaluating information sources, to try to combine sources that might be related. The problem itself however might not be a typical categorisation one, but one that tries to realise some global function over the whole set of problem data. It might be more similar to a neural network trying to realise a single evaluation function that maps its input to its output than several evaluation functions that each map a different thing. Tests have shown that it could be useful, for example, for selecting feature groups or best values out of partial information sets. How the hyper-heuristic would fit into a more cognitive model, as the centralised decision-making component, has also been considered.

7 References

- [1] Bader-El-Den, M. and Poli, R. (2007). Generating SAT Local-Search Heuristics using a GP Hyper-Heuristic Framework, *In Proceedings of Artificial Evolution (EA'07)*.
- [2] Bai, R., Blazewicz, J., Burke, E.K., Kendall, G. and McCollum, B. (2007). A Simulated Annealing Hyper-heuristic Methodology for Flexible Decision Support, *Tech. Rep. NOTTCS-TR-2007-8*, School of CSIT, University of Nottingham.
- [3] Blum, A. and Langley, P. (1997). Selection of relevant features and examples in machine learning. *Artificial Intelligence*, Vol. 97, No. 1-2, pp. 245–271.
- [4] Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E. and Qu, R. (2009). A Survey of Hyper-heuristics, *Computer Science Technical Report No. NOTTCS-TR-SUB-0906241418-2747*, University of Nottingham. (hhSurvey09)
- [5] Guo, G., Wang, H., Bell, D., Bi, Y. and Greer, K. (2004). An kNN Model-Based Approach and Its Application in Text Categorization, in *Computational Linguistics and Intelligent Text Processing, 5th International Conference*, Cicing 2004, Seoul, Korea, Author(s): Gelbukh, Alexander, ISBN 3540210067, Springer-Verlag New York Inc, pp. 559 - 570.
- [6] Greer, K. (2011). Symbolic Neural Networks for Clustering Higher-Level Concepts, *NAUN International Journal of Computers*, Issue 3, Vol. 5, pp. 378 – 386, extended version of the WSEAS/EUROPMENT International Conference on Computers and Computing (ICCC'11) paper

- [7] Greer, K. (2011). Literature Review for the Multi-Source Intelligence Project Called 'A Stochastic Hyper-Heuristic for Optimising through Comparisons' *Distributed Computing Systems Research Report*, published on Scribd [<http://www.scribd.com/doc/58227009/Hyper-Heuristic-Literature-Review>].
- [8] Greer, K. (2010). A Stochastic Hyper-Heuristic for Optimising through Comparisons, *The 3rd International Symposium on Knowledge Acquisition and Modeling (KAM 2010)*, Oct 16 - 18, Wuhan, China, pp. 325 – 328. Online version on IEEE Xplore.
- [9] Guyon, I. and Elisseeff, A. (1993). An Introduction to Variable and Feature Selection, *Journal of Machine Learning Research*, Vol. 3, pp. 1157-1182.
- [10] Kohavi, R. and John, G. (1997). Wrappers for feature selection. *Artificial Intelligence*, Vol. 97, No. 1-2, pp. 273–324.
- [11] Licas, <http://licas.sourceforge.net>.
- [12] McClean, S., Scotney, B., Greer, K. and Pairceir, R. (2001). Conceptual Clustering of Heterogeneous Distributed Databases, *Joint 12th European Conference on Machine Learning (ECML '01) and 5th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD '01)*. Workshop on Ubiquitous Data Mining for Mobile and Distributed Environments, September, pp. 46- 55.
- [13] Marin-Blazquez, J.G. and Schulenburg, S. (2006). Multi-Step Environment Learning Classifier Systems applied to Hyper-Heuristics, GECCO'06, July 8–12, Seattle, Washington, USA, pp. 1521 - 1528.
- [14] Mladenic, D., Brank, J., Grobelnik, M. and Milic-Frayling, N. (2004). Feature Selection using Linear Classifier Weights: Interaction with Classification Models, *In SIGIR*, Sheffield, U.K, pp. 234–241.
- [15] Ozcan, E., Misir, M. and Burke, E.K. (2009). A self-organising hyper-heuristic framework, in *Proceedings of the 4th Multidisciplinary International Scheduling Conference: Theory & Applications (MISTA'09)*, Dublin, Ireland, August 10–12, pp. 784–787.
- [16] Rocchio, J.J. (1971). Relevance Feedback in Information Retrieval. In *The SMART Retrieval System: Experiments in Automatic Document Processing*, pp. 313 – 323, ed. Gerard Salton, Prentice Hall, Englewood Cliffs, New Jersey.
- [17] Runarsson, T.P. and Yao, X. (2000). Stochastic ranking for constrained evolutionary optimization, *IEEE Transactions on Evolutionary Computation*, Vol. 4, No. 3, pp. 344-354.
- [18] Sahoo, N., Callan, J., Krishnan, R., Duncan, G. and Padman, R. (2006). Incremental Hierarchical Clustering of Text Documents, Published in: CIKM '06 Proceedings of the 15th ACM international conference on Information and knowledge management, ACM New York, NY, USA.