Literature Review for the Multi-Source Intelligence Project Called 'A Stochastic Hyper-Heuristic for Optimising through Comparisons'

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

1 Introduction

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. Different heuristics can evaluate certain concepts better than others. This can also occur simply from where in the search space the heuristic starts. A hyper-heuristic allows a search process to evaluate a wider range of possibilities and therefore provide a more robust answer to the problem. While this can also be performed manually by simply running many different heuristics with different sets of data, the hyper-heuristic will speed up this process through automating it. This is especially important if different heuristics are to be merged or combined as part of the search process.

This report provides a survey of some research that might be related specifically to the hyper-heuristic model of the current project [14]. The context of the work 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 and also symbolic. 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 purpose of the hyper-heuristic is to control this matching process so that the distributed information sources that

are most likely to contain related information can be combined in an efficient and accurate manner. This survey considers the hyper-heuristic framework and also the heuristics that it might use to directly evaluate the information sources. The report does not assume any level of expert knowledge in the domain of heuristics, but it does assume some background knowledge. The rest of this report is organised as follows: Section 2 summarises the main features of hyper-heuristics and why they should be used. Section 3 discusses the problem of selecting the relevant variables for any sort of classification. Section 4 summarises some of the heuristic algorithms that the hyper-heuristic might use for direct evaluation, while section 5 gives some conclusions on the work.

2 Hyper Heuristics

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. The first journal paper to use the term was [5], whilst the first widely available refereed conference paper was [7]. The idea, however, can be traced back to the early 1960s [8][9]. 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 which 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. This is also described in [1] as follows: The first type (heuristics to choose heuristics) is provided with a list of pre-existing heuristics for solving a certain problem. The hyper-heuristic system tries

to discover what the best sequence of applying these heuristics are, for the purposes of finding a solution. The second type (heuristics to generate heuristics) is used to build hyper-heuristic systems that evolve new heuristics, by making use of the 'components' of known heuristics. The process starts simply by selecting a suitable set of heuristics that are known to be useful in solving a certain problem. However, instead of directly feeding these heuristics to the hyper-heuristic system (as is the case for the first type), the heuristics are firstly decomposed into their basic components. Different heuristics may share similar components. However, during the decomposition process, information on how these components are connected with one another can be lost. To avoid this problem, this information can be captured by a grammar. So, in order to provide the hyper-heuristic systems with enough information on how to use components, to create valid new heuristics, one must first construct this grammar. The system then uses a suitable evolutionary algorithm to evolve new heuristics from the grammar.

2.1 Heuristics to Choose Heuristics

Using a hyper-heuristic to choose other heuristics is perturbative in the sense that each evaluation step requires the selection of a new set of heuristics to explore further. Perturbative hyper-heuristics can select potential solutions based on randomness, an exhaustive approach, or a combination of these. A perturbative (improvement) hyper-heuristic operates over a set of perturbative low level heuristics [4]. The search is conducted iteratively, selecting and applying a low-level heuristic (or a subset of low level heuristics) to the current solution(s) until a set of stopping conditions has been met. Most of the recently proposed perturbative hyper-heuristics perform a single point search, processing a single candidate solution at each iteration. However, there are others that use a population of candidate solutions within the search process. Perturbative hyper-heuristics have been applied to a wide variety of combinatorial optimisation problems. Typically, the heuristic selection method would generate an online score for each solution heuristic based on its performance. These values are then processed and/or combined in a systematic manner to select the heuristic to be applied to the candidate solution at each step. All score based heuristic selection techniques require five main components to be implemented: (i) initial scoring, (ii) memory length adjustment, (iii) strategy for heuristic selection based on the scores, (iv and v) score update rules in case of improvement and worsening, respectively.

A hyper-heuristic can also use a learning mechanism to try and improve the selection process. Reinforcement learning can be used to update the strength of each potential solution. If a low-level heuristic improves a solution, then it is rewarded and its score gets updated positively, while a worsening move causes punishment of a heuristic by decreasing its score. Different combinations of operators can be designed for reward and punishment. The acceptance strategy for the next search move can also be different. Deterministic methods make the same decision for acceptance regardless of the decision point during the search. On the other hand, a nondeterministic approach might generate a different decision at a different decision point for the same input. The decision process in almost all non-deterministic move acceptance methods requires additional parameters, such as the time (or current iteration).

2.2 Heuristics to Generate Heuristics

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. This would typically require some sort of genetic algorithm, but any sort of evaluating mechanism can fit into a hyper-heuristic. 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 is directed by optimising the solutions. 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.

2.3 Example Optimisation Problems

2.3.1 Generic Problems

There are different application areas for hyper-heuristics, for example: production scheduling, bin-packing, SAT, or the travelling salesman problem. Hyper-heuristics have been widely used with scheduling problems. Genetic programming has rarely been used to solve production scheduling instances directly, because it is unsuitable for encoding solutions. However, it is highly suitable for encoding scheduling heuristics [20]. With bin-packing problems, a number of pieces need to be packed into a number of bins in the most efficient manner. The heuristics generated for this consist of arithmetic operators and properties of the

pieces and bins. For each piece in turn, the framework iterates through the bins, applying the heuristic once to each. The heuristic returns a value for each bin, where the optimal value, as determined by some metric, is then selected each time. With the boolean satisfiability problem, there are a set of conditions made up of boolean variables. This results in an equation that needs to be solved by allocating the correct value to each variable in the equation. The variables of the problem space need to be set the correct boolean values so that they can satisfy each of the equations when used to solve them. The travelling salesman problem is another optimisation problem where the salesman needs to visit a number of locations in the minimum amount of time possible. The survey paper [4] gives more details on most of these methods, while section 4 describes some of the solution heuristics in more detail.

2.3.2 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 [27] 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 proposed framework is also self-organising in nature, but through a more stochastic process. The reinforcement learning could be applied to one of the evaluating heuristics, but not to the framework itself.

The paper [24] 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 are concerned with the problem of creating rule-sets for state/action pairs, where the rule is consulted to find the appropriate heuristic to change the problem state. This can however be applied to general optimisation problems, such as bin-packing. 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 [29]. The heuristic selection mechanism dynamically tunes the priorities of different heuristics during the search. Initially, the heuristic selection mechanism does not know whether any heuristic will perform better than any other. Therefore, all low-level heuristics are treated equally and the heuristic selection decisions are made randomly. While the search is proceeding, the

heuristic selection mechanism starts to apply preferences among different low-level heuristics by learning from, and adapting to, their historical performance. Therefore, the heuristics that have been performing well are more likely to be chosen. To successfully apply a selected heuristic, the simulated annealing acceptance criterion also has to be satisfied, but this will favour the more popular heuristics. Information about the acceptance decisions by the acceptance criterion is then fed back to the heuristic selection mechanism in order to make better decisions in the future.

So this contains a lot of the ideas contained in the current project proposal. 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 current project obtains a score though similarity matching, but the randomness comes from the solution placing in the grid. This is totally random at the start and so equal for any solution. The threshold criteria in effect, is set by the surrounding solutions. As the solutions mutate and merge, the solution groupings will become more similar and therefore will be less likely to change. It will then, more importantly, be the relative scores that determine what solutions are kept or lost.

Structures such as belief networks can be created as part of information theory; to group or cluster distributed information sources. This can even be applied to sensor-based information, as is done in [31]. A hypothesis is generated and the belief network calculates how much each sensor input contributes to the hypothesis. It can create a hierarchical network, with hidden layers representing more complex relationships. This is probably exactly what the hyper-heuristic wants to do, but the proposed framework does not want to do it in that way. The hyper-heuristic framework itself does not want to rely on existing knowledge or beliefs of any sort. It only wants to be able to correctly sort the feedback that it receives from the evaluating heuristics. These evaluating heuristics might eventually be tested with some sort of belief or evidence theory, but this initial test stage is more concerned with the hyper-heuristic framework itself.

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 [17] 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 [21] 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. Relevance as a complexity measure helps to define how complex a function is, with respect to the fewest number of features that produce the minimal error. To be useful to an algorithm, features

need to be allowed to be incrementally added and/or removed as part of a search process. This gives the idea of incremental usefulness.

The paper [3] also describes how heuristic search can be used for feature selection, where each new state in the search space can specify a different subset of features. Induction algorithms, filtering, wrappers, or also weighting, can be used to help to select the features. Induction involves learning the classification through examples. Logical descriptions can also be embedded into the algorithm. They can simply be used to add or remove features based on the prediction of errors in new instances. Filtering can also be used to reduce the number of potential features. Licas [15] has a built-in filtering mechanism, where uncommon features would be less likely to make it to the top level of the linking structure. However, an irrelevant feature could also mean a feature that is always present in every dataset that is presented. Wrapper methods can be added to an induction algorithm to evaluate different feature sets over the same search space. A subroutine is added to evaluate different feature sets using the same induction algorithm and use the resulting different classifiers as its metric, for a more selective search process. The resulting global classifier is then evaluated on a dataset that was not used during the search [21]. This is similar to a filter but can be computationally more expensive. There is also the possibility of feature weighting, as in neural networks.

3.1 Feature Selection Equations

There are a number of existing equations that can be used to evaluate if a feature belongs to a particular category or not. The current project is trying something new and is more interested in developing the hyper-heuristic framework. However, the following sections show what sort of evaluations or criteria could be used when deciding what category a feature belongs to. Most of these actually belong to clustering algorithms that would try to measure how similar two data objects are.

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 [22]:

$d_{12} = \sum_{j=1}^{k}(\pi_{1j} - \pi_{2j})^2$ where d is the distance and π_1 or π_2 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 [22]:

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

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 $\left|\frac{T_1 \cap T_2}{T_1 \cup T_2}\right|$

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 [28] is a similarity-based linear classifier that considers both positive and negative input. The equation [13] can be described as: given a training dataset T_r, the

Rocchio classifier directly computes a classifier $\vec{c_i} = < w_{1i}, w_{2i}, ..., w_{ri} >$ 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 Odds Ratio

The paper [26] describes a method called Odds Ratio that also appears to provide a probability based on a positive or negative evaluation. Let *P(t|c)* be the probability of a randomly chosen word being *t*, given that the document it was chosen from belongs to a class *c*. Then *odds(t|c)* is defined as:

$$\frac{P(t|c)}{[1-P(t|c)]} \quad \text{where the Odds Ratio equals to:} \quad OR(t) = \ln[\frac{odds(t|c+)}{odds(t|c-)}]$$

Obviously, this scoring measure favours features that are representative of positive examples. As a result a feature that occurs very few times in positive documents but never in negative documents will get a relatively high score. Thus, many features that are rare among the positive documents will be ranked at the top of the feature list. Odds Ratio is known to work well with the Naïve Bayes learning algorithm.

3.1.6 Information Theoretic Ranking using Probability Densities

The paper [17] 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(xi)* 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. With this simpler case, the probabilities can be estimated from frequency counts. For example, in a three-class problem, if a variable takes 4 values: P(Y = y) represents the class prior probabilities (3 frequency counts), P(X = xi) represents the distribution of the input variable (4 frequency counts), and P(X = xi, Y = y) is the probability of the joint observations (12 frequency counts). It is therefore a measure of how likely the variable with a particular value will belong to a particular class. The estimation obviously becomes harder with larger numbers of classes and variable values. The case of continuous variables (and possibly continuous targets) is the hardest. One can consider discretizing the variables or approximating their densities with a non-parametric method, where no assumptions can be made about the distributions that the values came from. Using the normal distribution to estimate densities would bring us back to estimating the covariance between X_i and Y, thus giving us a criterion similar to a correlation coefficient. When selecting the objective function for variable selection, the function itself should be maximised (goodness-of-fit) and the number of variables to optimise it should be minimised.

3.1.7 Information Gain

The paper [26] 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:

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

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).

3.1.8 Feature Selection based on Linear Classifiers

As described in [26], both SVM and Perceptron, when used as linear classifiers, output predictions of the form:

$prediction(x) = sgn(w^T x + b) = sgn(S_j w_j x_j + b)$.

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.

3.1.9 ROCK Cluster Criterion Function

The paper [16] describes the ROCK clustering algorithm that uses links instead of similarity measures. For data point pairs p_q and p_r belonging to the same cluster, the criterion function would like to maximise the sum of $link(p_q, p_r)$, and also, minimise the sum if they belong to different clusters. This leads to the following criterion function to be maximised for the *k* clusters:

$E_l = \sum_{i=1}^{k} n_i * \sum_{p_q,p_r \in C_i} \frac{link(p_q,p_r)}{n_i^{1+2f(\theta)}}$

Where C_i denotes cluster i of size n_i. Simply summing all links between two points is not ideal because too many points can still be assigned to a single cluster the above equation has therefore refined this by dividing the total number of links between the points ($link(p_q, p_r)$) by the total number of expected links in the cluster C_i ($n_i^{1+2f(\theta)}$) and then weighting this; where the function $f(\theta)$ is also defined, but needs to be assigned the correct value. Dividing by the expected number of links prevents clusters with only a few similar links from being placed in the same cluster.

3.1.10 K-ANMI Similarity Measure

The paper [18] describes a metric for measuring similarity between categorical or textual sets of attribute values. This is not the *k*-ANMI algorithm itself, but their similarity measure. Their equation goes as follows: let *X* and *Y* be two different categorical objects described by *r* categorical attributes. The dissimilarity measure between the two objects can be defined as the total number of mismatches between the corresponding attribute values, where the smaller number of mismatches defines greater similarity. The equation can be written as:

$$d_1(X, Y) = \sum_{j=1}^{r} \delta(x_j, y_j) \qquad \text{where } \delta(x_j, y_j) = \begin{cases} 0 & (x_j = y_j) \\ 1 & (x_j \neq y_j) \end{cases}$$

Then, given a dataset *D* with objects $(X_1, X_2, X_3, \ldots, X_n)$ and related object *Y*, the dissimilarity measure between *X* and *Y* can be defined as the sum of the similarity distances between all X_i and *Y* normalised. They also optimise the equation for computing efficiency, but this equation essentially sums how many attribute values are the same and how many are different, allocating a boolean result of 1 or 0 to each value evaluation.

4 Algorithms

This section summarises the details of some algorithms that a hyper-heuristic might use as potential solutions to problems. These algorithms would be used to evaluate the problems directly, or to drive the heuristic search.

4.1 Hill Climbing and Greedy Algorithms

These algorithms are relatively similar to each other. With these algorithms, the best solution is selected each time, which means that the search process moves quickly towards a local optimum. This could also be a global optimum, but as there is less variability in the search selection process, there is no guarantee that all of the potential possibilities have been robustly evaluated. A local optimum represents an optimal set of values for the parameter set that has been evaluated. A different set of values somewhere else however could lead to a much better solution that the current set of values cannot know about. Greedy algorithms are short-sighted and do not consider future moves at all. They base their next move only on the current evaluations. These algorithms are therefore very quick, but not particularly accurate, unless the heuristic is well suited to the problem in question. The loss of accuracy can be compensated to some degree by trying several evaluation searches, starting from different positions. A hyper-heuristic is therefore particularly useful with these algorithms, where it can automate and randomise the execution of the algorithms at different places in the search space.

4.2 Linear Classifiers

Linear classifiers use linear methods to separate or classify data. They would be a simpler form of classifier, compared to the non-linear ones. This can be used with something like a nearest neighbour algorithm or neural networks, for example. For a linear classifier to be useful however requires that the data can be separated by linear hyperplanes, which in turn means that the classification function is linear in nature. A linear classification means that the categories can be realised through linear or straight classification planes that result from linear equations. For example:

$Y = f(w.x + z)$ is a linear equation, whereas: $Y = f((w.x) + z)^2$ is quadratic and therefore non-linear.

If drawing on a graph, the first equation results in a straight line, whereas the second would result in a curved line. If considering neural networks, a stepwise function would be linear, whereas a sigmoid one would be non-linear.

The paper [26] looks at different weighting methods that can be used for feature selection. They note that in text classification, feature selection is typically used to achieve two objectives: reduce the size of the feature set in the data representation and remove noise from the data in order to optimize the classification performance. Stop word removal or linguistic normalization using stemming can be used to combine text or words, to reduce the feature space further. They suggest that the sophistication of the weighting method is more important than its apparent compatibility with the learning algorithm. They tested three learning algorithms: naïve bayes, perceptron and support vector machine; with three feature selection methods: odds ratio, information gain and feature selection based on linear classifiers. The two linear classifiers tried were perceptron and linear support vector machine (SVM). These feature selection methods are described in section 3. They note the success of SVM and also the fact that other classifiers can be improved by using the feature selection results derived from SVM. They also note that both linear classifiers perform better than Odds Ratio when the sparsity of the feature vectors falls below 40 features per vector. Sparsity refers to the number of elements in the matrix or vector that are occupied. They note that the algorithms are more sensitive to sparsity than the total number of features.

4.3 Clustering or Nearest Neighbour Algorithms

Clustering can be used for feature construction, for example, *k*-means. As described in Wikipedia: *k*-means is a method of cluster analysis which aims to partition *n* observations into *k* clusters in which each observation belongs to the cluster with the nearest mean. It is similar to the expectation-maximization algorithm for mixtures of Gaussians in that they both attempt to find the centers of natural clusters in the data as well as in the iterative refinement approach employed by both algorithms.

Clustering is usually associated with unsupervised learning and can be used to categorise textual documents, for example. Nearest neighbour algorithms work by trying to classify a new input with the group of inputs that it is most similar to. The main difficulty is to determine what features in the input data set should be compared for similarity. The paper [22] describes algorithms for clustering heterogeneous databases and also introduces the similarity metrics of Euclidean and Kullback-Leibler distance metrics. These are described further in section 3. The paper then describes a hybrid approach based on these two equations

for dealing with heterogeneous data, even when some of the information is missing; but dealing with heterogeneous information lies outside the scope of this survey.

One example of a nearest neighbour algorithm applied to text categorisation can be found in [13]. This uses a *k*-NN algorithm, which tries to match the current text input with a total number of *k* neighbours. This is in fact more efficient than a traditional nearest neighbour algorithm that would consider every existing cluster when trying to match with a new input [30]. The use of *k*-NN in [13] is combined with the Rocchio method [28]. The Rocchio method is a similarity-based algorithm that tries to generalise by summarising the contribution of each training instance to the category in question. It only then needs to compute the inner product between any new instance and the generalised instances. Through summarising, it can also deal with some level of noise and distil out certain relevant features. The drawback is that the classification is restricted to a certain set of linear hyperplanes that are less expressive than the *k*-NN algorithms. The Rocchio algorithm is described in more detail in section 3.1.4. The new algorithm is called *k*-NN model-based classifier (*k*NNModel) and tries to use the generalisation of categories that the Rocchio method creates to make the clustering process more efficient. The *k*-NN algorithm can be inefficient because it keeps all of the training data for classification. To speed this up, generalisations of this data can be calculated using the Rocchio method. To maintain accuracy, the algorithm then also stores a number of centroids for each cluster category instead of just one. Where the Rocchio method by itself would miss-classify through linear hyperplanes, the new *k*NNModel can be more accurate by using more than one centroid for each generalised classification, also making the classification non-linear. Each new data point to be classified can then be compared to each generalised instance and also the distances from each centroid for that instance. Tests showed that the *k*NNModel algorithm outperformed either of the other two methods.

While there are not many examples related to the current project, the papers [16] and [18] look to be directly relevant. The clustering problems in the current project are also written about in [16]. They describe two different types of clustering algorithm: partitional and hierarchical. Partitional clustering algorithms could use a Euclidean distance metric to measure how close a data object is to a cluster and use hill climbing to try to then maximise the total objective value. While this could be suitable for numerical data, it might not be for categorical data. For example, if clustering shopping baskets, there are potentially thousands of different products and each customer only buys a few of those. People may be clustered

into groups that overlap only slightly, that is, they buy only a few of the items that define a much larger cluster. Furthermore, they are related to other people in the cluster through only a few items. So through a sequence of links all belonging to the same cluster, one person can be clustered with another person that he/she has relatively little in association with. An option can then be to split the larger clusters into smaller ones, but this is not desirable because the larger cluster can be split through planes that would also separate well connected data objects. Hierarchical clustering algorithms also have problems. They would typically take two existing clusters and merge them into another one at the next level in the hierarchy. For the categorical attributes however, the distances between the centroids of clusters is still a poor measure of similarity. As the cluster size grows, the number of attributes appearing in the mean group goes up and the importance of each attribute goes down. This then makes it more difficult to distinguish between two objects that differ by only a few attributes.

An alternative distance metric such as the Jaccard coefficient [19] can be used. This is described in section 3.1.3 and measures the distance between two objects directly. It does not measure cluster means or centroids. The centroid-based hierarchical approach cannot therefore be used with it, but a ‘minimum spanning tree’ or ‘group average’ approach [19] can. The minimum spanning tree tries to connect nodes in a graph, while minimising the function, where the nodes would represent the clustered data points. For hierarchical clustering, the MST algorithm merges, at each step, the pair of clusters containing the most ‘similar pair’ of points, while the group average algorithm merges the ones for which the ‘average similarity’ between pairs of points in the clusters is the highest. The MST algorithm is known to be very sensitive to outliers while the group average algorithm has a tendency to split large clusters. The MST algorithm is also known to be fragile when clusters are not well-separated. The Jaccard coefficient does not reflect the properties of the neighbourhood and consequently, it fails to capture the natural clustering of ‘not so well-separated’ data sets with categorical attributes.

The authors of [16] then suggest an improvement to the clustering algorithms written about above, with their own algorithm called ROCK. Their algorithm is based on ‘links’ between data points, instead of similarity distances or the Jaccard coefficient. They have already described the unsuitability of these for domains with discrete non-numeric attributes. The situation with these distance metrics further worsens as the number of attributes/dimensions increase. The notion of links between data points helps to overcome the problems as follows:

Let a pair of points be neighbours if their similarity exceeds a certain threshold. The similarity value for pairs of points can however be based on any of the distance metrics. The number of links between a pair of points is then the number of common neighbours for the points. Points belonging to a single cluster will in general have a large number of common neighbours, and consequently more links. Thus, during clustering, merging clusters/points with the most number of links first will result in better and more meaningful clusters. They argue that clusters can have certain points that are close to each other. If two objects are then compared based only on the attributes that they have in common, this can cause these clusters to merge together. Two objects can be compared on certain attribute values, but attribute values that are missing are not considered and so this is a local comparison only. If two objects in different clusters are neighbours, it does not mean that they also have a large number of other common neighbours, that is, other objects that are also neighbours to both. Therefore it is incorrect to classify them as the same. The link-based approach that they suggest is also able to capture global knowledge of neighbouring data objects and use this also for the clustering evaluation. Another option here might be to put more emphasis on negative comparisons as well. This mechanism therefore uses links and thresholds, as does the licas linking mechanism. In this case, the links are built up purely from the existing knowledge or information. It does not have to rely on any time-based element to suggest likely clusters. But of course, time can add information that is not otherwise present. The criterion that they use to recognise clusters is given in section 3.1.9. The ROCK algorithm itself is then hierarchical in nature and uses a goodness measure to determine the best clusters to merge at each new hierarchy level.

The paper [18] provides a new *k*-means link clustering algorithm based on similarity measures between attribute values for different sets of attributes. The data that they consider is also categorical, or textual and so this is similar to the current problem. The metric itself is described in section 3.1.10. They also use a ‘cluster ensemble’, which combines the runs of different clustering algorithms to get a common partition view over all of the suggested results. They consider the categorical classification problem to be a cluster ensemble, because each clustering algorithm can be used to cluster different attribute values, to give a ‘best clustering’ for that attribute; without the requirement to consider the other attributes as well. The different ensemble results can then be combined using the equations of section 3.1.10 to give a total score for each object comparison. The *k*-ANMI algorithm itself takes the number of desired clusters *k* and iteratively changes the class label of each data object to try and

improve the object's function value. If the value is improved, then the label is changed. The algorithm terminates when there are no more changes. Their example in table 1 shows the different clustering that can occur when new information is combined:

Record Number	Attribute 1	Attribute 2
1	M	A
2	M	B
3	F	B
4	F	A
5	M	C
6	F	C
7	M	C
8	F	C
9	F	A
10	M	B

Table 1. Example of categorical attribute values, from table 1 in [18].

- If clustering on just attribute 1, then the clusters are {(1, 2, 5, 7, 10), (3, 5, 6, 8, 9)}.
- If clustering on just attribute 2, then the clusters are {(1, 4, 9), (2, 3, 10), (5, 6, 7, 8)}.
- If clustering on both attributes, then the clusters are {(1, 2, 3, 4, 5), (6, 7, 8, 9, 10)}.

4.4 Genetic Algorithms

As written in [6], computers do not program themselves; they need a qualified and experienced programmer who understands the complexity of the task. An alternative to paying a human programmer to design and debug a program is to build a computer system to evolve a program. This may not only be cheaper, but has the advantage that progress can be made on problem domains where a human programmer may not even have a clear idea of what the programming task is, as there is no formal program specification. Instead, a partial description of the desired program's behaviour could be supplied in terms of its input-output behaviour. Genetic Programming, a branch of program synthesis, borrows ideas from the theory of natural evolution to produce programs. The main components of evolutionary

computation are inheritance (crossover), selection and variation (mutation). Inheritance implies that the offspring have some resemblance to their parents as almost all of the offspring's genetic material comes from them. Selection means that some offspring are preferable to others, and it is this selection pressure which defines what individuals are fitter then others. Variation supplies fresh genetic material, so individuals containing genetic material that was not present in either of the parents (or the wider gene pool) can be created. Evolutionary computation can be thought of as the interaction of these three components. Informally, a population of computer programs is generated, and the genetically inspired operations of mutation and crossover are applied repeatedly in order to produce new computer programs. These programs are tested against a fitness function that determines which ones are more likely to survive to future generations.

The papers [10] and [25] try to make a direct comparison between hill climbing and genetic algorithms and try to conclude why one works better than the other. They note that while genetic algorithms would be expected to outperform hill climbing on a set of tests, the results showed that this was not the case; although, this did depend on the type of hill climbing algorithm that was used. Genetic algorithms would be expected to work better because the mutation process keeps previous successes in the new better solutions. The more random nature of the hill climbing algorithms would not necessarily do this. However, while hill climbing needs to optimise certain bits of a problem simultaneously; requiring some form of coordination, the genetic algorithm can suffer from a feature called 'hitchhiking'. This can mean that a highly-fit or better solution can overpopulate a solution space and bring incorrect elements of other solutions along with it (they hitchhike). In [10] they found that the incorrect hitchhiking elements were not always beside the fittest parts of the mutated solution, where they could possibly be missed by crossover. This problem slows down the discovery of solution parts in the other positions from those defined in the highly-fit solution. The genetic algorithm can, in effect, fix certain solution regions sub-optimally. In [25] they note that a hill climbing approach that is random in nature can outperform a genetic algorithm as well. By random it is meant that the next solution to evaluate is a random change to the current one. They also show however that an idealised genetic algorithm will perform better. The idealised version can mutate a new solution completely independently of previous ones, allowing any solution region to be changed, but to do this requires knowledge of the final correct solution.

4.5 Reinforcement Learning

Reinforcement learning, as its name suggests, reinforces certain criteria through feedback from its environment to make certain conditions true, while other conditions are made false. This can be done through something such as a weight increment or decrement mechanism and is essentially what the licas linking mechanism uses. As written in [23], reinforcement learning is a synonym of learning by interaction. During learning, the adaptive system tries to perform certain actions on its environment that are then reinforced through a scalar evaluation of those actions. The reinforcement learning algorithms selectively retain the outputs that maximize the received reward over time. They also note that the problem of reinforcement learning is in knowing what to reinforce. Motivation cannot rely on a blind mechanism that strengthens or weakens connections based only on their temporal proximity to certain stimuli. While temporal difference reinforcement may work well enough in small systems, it becomes prohibitive in large systems. As the number of entities grow; and each entity needs to be measured for each event, then this would become prohibitive. The work in this project will actually rely heavily on time for defining the clusters. The authors of [23] are possibly thinking of a system where events occur in parallel and are interconnected. Stigmergy then uses a time element to define the strength of the stimulus that it leaves behind. This can affect any other event that happens by any other entity afterwards. So this is a more autonomous interaction, with the entity itself being influenced by the stimulus. This is different to a more centralised clustering algorithm being presented different groups of data, which is in turn different to a completely centralised algorithm being presented ‘all’ of the data in one go. The author would then argue that time is a key advantage that this sort of system would have over one that needs to sort the whole environment in one go. Time will help to define what pieces of data could be placed into plausible clusters.

4.6 SAT Algorithms

SAT problems are described in [1], [11] and [12], for example. As written in [1], the target in the satisfiability problem (SAT) is to determine whether it is possible to set the variables of a given boolean expression in such a way as to make the expression true. SAT problems have been shown to be NP-hard, which means that they are the more difficult problem types that can be solved in polynomial time, that is, in time n^k. They also note in [12] that sideways or poorer moves can be good for the SAT problem, when dealing with hill climbing and local optima. They also note that a totally random or greedy process of ‘flipping’ states is not as

good as one that includes some form of hill climbing. Hill climbing will give some sort of direction to the search. The SAT algorithm looks relevant to the current problem, because when dealing with arbitrary concepts, they can either belong to a particular solution (global concept) or not. However, if you consider this problem more closely, it does not help very much to try and specify that certain conditions should be true. This will not actually evaluate them to true. If specific values or conditions are involved however, then this sort of equation might become more important, but the general form of an inclusion equation would probably be more like a Horn clause. This can be rewritten to state that to show the solution, you need to show all of the concepts in it. This again might not help to prove where a concept belongs, but if the solutions were mutually exclusive, that is, a concept is allowed in only one solution, then this sort of logic could be a help. Also, if the truth of the concept was not completely independent of other concepts, then this sort of logic could again help.

4.7 Induction or Learning Algorithms

Induction algorithms can be used to learn classifications from examples. They involve a search process through potential solutions to create a classification tree or evaluating metric that can be used to correctly classify unknown examples as well as the training dataset. The hyper-heuristic itself would probably replace an induction algorithm and so these do not need to be listed here. The main concern is how the hyper-heuristic creates its own evaluating metric, which is the same goal as for the induction search process. While induction algorithms will not be considered, the similarity between the licas linking mechanism path descriptions and an induction tree should be noted.

5 Conclusions

This survey has summarised a number of learning algorithms that might be more relevant to the classification problem of the current project. The aims of the project are to develop a hyper-heuristic framework for evaluating information sources, to try to combine sources that might be related. This review however is not meant to be exhaustive, but to give a flavour of the sort of problem-solving domain that the current project lies in and how it might relate to these current mainstream solutions.

Acknowledgements

This work was funded by the UK MoD research call on Multi-source Intelligence environments (CDE 19857).

6 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] Burke, E.K., Kendall, G., Soubeiga, E. (2003). A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics*, Vol. 9, No. 6, pp. 451–470.

[6] Burke, E.K., Hyde, M.R., Kendall, G., Ochoa, G., Ozcan, E. and Woodward, J.R. (2009). Exploring hyper-heuristic methodologies with genetic programming. In C. Mumford and L. Jain, editors, *Collaborative Computational Intelligence*. Springer.

[7] Cowling, P., Kendall, G., Soubeiga, E. (2000). A hyperheuristic approach for scheduling a sales summit. In: *Selected Papers of the Third International Conference on the Practice And Theory of Automated Timetabling*, PATAT 2000, Springer, Konstanz, Germany, Lecture Notes in Computer Science, pp. 176–190.

[8] Fisher, H. and Thompson, G.L. (1961). Probabilistic learning combinations of local job-shop scheduling rules. In: *Factory Scheduling Conference*, Carnegie Institue of Technology.

[9] Fisher, H. and Thompson, G.L. (1963). Probabilistic learning combinations of local job-shop scheduling rules. In: *Muth JF, Thompson GL (eds) Industrial Scheduling*, Prentice-Hall, Inc, New Jersey, pp. 225–251.

[10] Forrest, S. and Mitchell, M. (1993). Relative Building-Block Fitness and the Building-Block Hypothesis, in Whitley, D. (ed.) *Foundations of Genetic Algorithms 2*, Morgan Kaufmann, San Mateo, CA.

[11] Fukunaga, A.S. (2008). Automated Discovery of Local Search Heuristics for Satisfiability Testing, *Evolutionary Computation*, Vol. 16, No. 1, pp. 31-61.

[12] Gent, I. and Walsh, T. (1993). Towards an understanding of hill-climbing procedures for SAT. *In Proceedings of National Conf. on Artificial Intelligence (AAAI)*, pp. 28–33.

[13] 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*, Cicling 2004, Seoul, Korea, Author(s): Gelbukh, Alexander, ISBN 3540210067, Springer-Verlag New York Inc, pp. 559 - 570.

[14] 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.

[15] Greer, K. (2009). A Cognitive Model for Learning and Reasoning over Arbitrary Concepts, *The 2nd International Symposium on Knowledge Acquisition and Modeling (KAM 2009)*, Nov 30 – Dec 1, Wuhan, China, pp. 253 - 256. Online version on IEEE Xplore.

[16] Guha, S., Rastogi, R. and Shim, K. (2000). ROCK: a robust clustering algorithm for categorical attributes, *Information Systems*, Vol. 25, No. 5, pp. 345 – 366.

[17] Guyon, I. and Elisseeff, A. (1993). An Introduction to Variable and Feature Selection, *Journal of Machine Learning Research*, Vol. 3, pp. 1157-1182.

[18] He, Z., Xu, X. and Deng, S. (2008). K-ANMI: A Mutual Information Based Clustering Algorithm for Categorical Data, *Information Fusion*.

[19] Jain, A.K. and Dubes, R.C. (1988). Algorithms for Clustering Data. Prentice Hall, Englewood Cliffs, New Jersey.

[20] Jakobovic, D., Jelenkovic, L. and Budin, L. (2007). Genetic programming heuristics for multiple machine scheduling. In: *LNCS 4445. Proceedings of the European Conference on Genetic Programming (EUROGP'07)*, Valencia, Spain, pp. 321–330.

[21] Kohavi, R. and John, G. (1997). Wrappers for feature selection. *Artificial Intelligence*, Vol. 97, No. 1-2, pp. 273–324.

[22] 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.

[23] Mano, J.-P., Bourjot, C., Lopardo, G. and Glize, P. (2006). Bio-inspired Mechanisms for Artificial Self-organised Systems, *Informatica*, Vol. 30, pp. 55 – 62.

[24] 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.

[25] Mitchell, M., Holland, J.H. and Forrest, S. (1994). When Will a Genetic Algorithm Outperform Hill Climbing? in Cowan, J.D., Tesauro, G. and Alspector, J. (eds.), *Advances in Neural Information Processing Systems 6*, Morgan Kaufmann.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] Skalak, D.B. (1994). Prototype and Feature Selection by Sampling and Random Mutation Hill Climbing Algorithms, *Proceedings of the Eleventh International Conference on Machine Learning*, pp. 293-301.

[31] Zhang, Y. Ji, Q. (2009). Efficient Sensor Selection for Active Information Fusion, *IEEE Transaction on Systems, Man, and Cybernetics* - Part B: Cybernetics.