# The 'Composite' Design Pattern in Metaheuristics

John Woodward
Computing Science and Mathematics
University of Stirling
FK9 4LA Scotland UK
john.woodward@cs.stir.ac.uk

Jerry Swan
Computing Science and Mathematics
University of Stirling
FK9 4LA Scotland UK
jsw@cs.stir.ac.uk

Simon Martin
Computing Science and Mathematics
University of Stirling
FK9 4LA Scotland UK
spm@cs.stir.ac.uk

## Categories and Subject Descriptors

I.2 [**ARTIFICIAL INTELLIGENCE**]: Problem Solving, Control Methods, and Search|Heuristic methods.

## Keywords

Design Patterns; Evolutionary Computing; Metaheuristics; Hyper-heuristics; Classifiers.

## 1. THE 'COMPOSITE' DESIGN PATTERN

Introduced in [13], the 'Composite' Design Pattern can be observed whenever some 'aggregator' object can be considered to share behavioural commonalities with the entities from which it is composed. It therefore offers a concise means of describing functional [9] or structural recursion [5], which are of course ubiquitous in computer science and software engineering. In computer science, the prototypical example is a recursive data structure such as a tree, defined inductively either as an empty node or else as a node having a list of nodes as its children. In software engineering, an example is a concept such as Displayable, instances of which might include OKButton, (which can directly display itself), or Window which might display itself by asking all the Displayables it contains to display themselves in turn. Figure 1 shows the class diagram for a prototypical example of the composite pattern [13].

It is useful to be able to identify Composite relationships within metaheuristics for several reasons:

- Ease of communicating entity relationships — the same motivation for factoring patterns from software architectures.
- Facilitating automation — a composite implicitly defines a 'grammar template' for the generation of new metaheuristic components.

In general, the pattern is applicable whenever either or both of the following apply:


*GECCO'14,* July 12–16, 2014, Vancouver, BC, Canada.

ACM 978-1-4503-2881-4/14/07 ...$15.00.
http://dx.doi.org/10.1145/2598394.2609848.

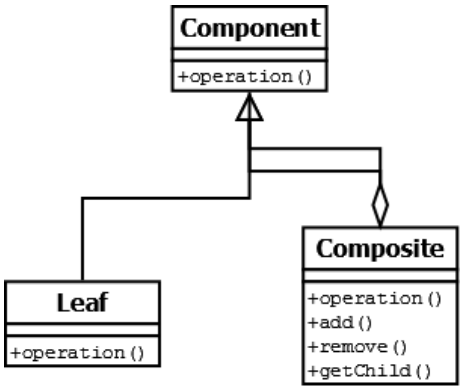


**Figure 1: UML class diagram of composite pattern**

- Decisions can incorporate information obtained by a diverse range of techniques (e.g. ensemble methods, selective hyper-heuristics).
- The granularity at which decisions are made imposes a notion of hierarchy that can be expressed by aggregation (e.g. multi-level search).

In their most general form, the behaviour of instances of the Composite pattern might be expressed by a collection of functions (e.g. isLeaf(), numChildren(),display() etc). In the case where the behaviour to be aggregated consists of a single function $f$ (as is often the case in metaheuristics, as demonstrated by the above example), with signature

$$f(t_1 : T_1, t_2 : T_2, \ldots, t_n : T_n)$$

where $T_1 \ldots T_n$ are the types of the formal arguments, then the value of composite function $f_c$ (with $k$ children with functions $f_1 \ldots f_k$ respectively, each with the same signature as $f_c$) can be expressed in general as:

$$f_c(t_1 : T_1, t_2 : T_2, \ldots, t_n : T_n) \triangleq_{def} g(t_1, t_2, \ldots, t_n, f_1, \ldots, f_k)$$

where the implementation of $g$ may elect to invoke any or all of $f_1, \ldots, f_k$ as some arbitrary function of actual parameter values $t_1, \ldots, t_n$. Such a formulation clearly lends itself to automation, e.g. via genetic programming [20] or decision tree induction [26].

Below, we give two detailed examples of the 'Composite' pattern in metaheuristics (viz. 'Hyper-heuristic as composite metaheuristic' and 'Composition of Classifiers'). These (and the 'composite function' scheme just described) are merely exemplars: if the above criteria apply, then the pattern is of

potential value across the entire spectrum of metaheuristic concerns (e.g. clustering, dimensionality reduction etc).

# 2. HYPER-HEURISTICS AS COMPOSITE METAHEURISTICS

## 2.1 Problem statement

The uniform treatment of a collection of operators is likely to be useful when:

- It is necessary to escape the combined basins of attraction of a cluster of local minima. The required perturbation strength should therefore ideally be dynamically determined (e.g. as in the 'breakout local search' metaheuristic [**?**]).
- It is desirable (as is the case with multi-level search) to treat perturbation as a heirarchical aggregation of lower-level perturbations (grounding of course in primitive operators that directly affect solution state).
- We wish to clarify the relationship between hyper-heuristics and the operators they invoke or generate. Hyper-heuristics can be described as "heuristics for searching the space of heuristics", and (in the form of *selective hyperheuristics*) can be seen as imposing a homogeneous perspective on the heterogeneous selection of operators.
- A more expansive (yet concrete) definition of hyper-heuristic is required in order to facilitate the creation of solvers that can act with greater autonomy. Although hyper-heuristics have been categorized as selective or generative [4], there is no necessity for solvers to be restricted *a priori* to only one of these categories.

The 'Composite' pattern is applicable precisely when we wish to treat entities and collections of entities in such a uniform manner. Explicit acknowledgement of the presence of the composite pattern allows the relationship between between the levels of operator application to be formally clarified.

## 2.2 The solution

Let $S$ denote solution-state and let $[S]$ be a list of same. We start by formally defining an operator. For generality, it is convenient to operate on traces (i.e. a time-series of solution states) rather than single states — this allows the operator to act as a function of solution history (e.g. in the manner of tabu search). Define an operator to be a function with signature

$$O_S : [S] \rightarrow [S]$$

where the argument represents the trace of solution-states and the result represents the extension of the trace obtained by applying the operator. Following [29], we can then define a hyper-heuristic to be a function with the following signature:

$$\mathcal{H}_S : [S] \times [O_S] \rightarrow [S] \times [O_S]$$

i.e. $\mathcal{H}_S$ takes a list of solutions and a list of operators as input, and produces as output a (potentially updated) list of solutions and operators. It is clear that this suffices to represent both selective and generative hyper-heuristics: the former may update only the solution list, the latter the operator list.

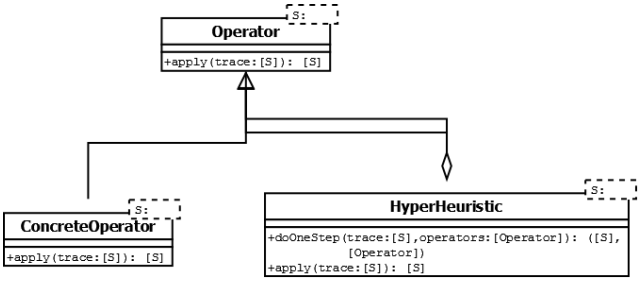


**Figure 2: UML class diagram of composite hyper-heuristic**

We consider $\mathcal{H}_S$ to be *parameterized* by $S$, i.e. it is associated with a specific type of solution-state (e.g. bit-strings, permutations etc). There is of course no necessity for $S$ to be a data-type: it may itself be a function. In particular, $S$ may itself be a meta- or hyper-heuristic: if we take $S$ to be $\mathcal{H}_T$ for some solution-type $T$, then $\mathcal{H}_{\mathcal{H}_T}$ expresses a multi-level search over $T$. This recursive structure is well-known to be characteristic of the composite pattern. Figure 2 gives an instantiation of the Composite pattern for hyper-heuristics. From this diagram, we can see that a hyper-heuristic IS-A Operator and USES-A sequence of operators. Since we can consider metaheuristics (e.g. tabu search, simulated annealing etc) to be synonymous in both behavioural and signature terms with the definition of operator, we can consider hyper-heuristic to be a composite metaheuristic.

## 2.3 Examples

- In their description of local search frameworks, Lourenço et al. [22] note the potential for nesting of operators, thereby implicitly alluding to this pattern.
- The HYPERION framework [28] uses the class decomposition above to allow the same source code to express both metaheuristic and hyper-heuristic, depending on the choice of the type $S$.
- In multi-level search, the appropriate level of nesting cannot usually be determined *a priori*, but the polymorphic relationship between OPERATOR and HYPER-HEURISTIC means that the nesting of levels can be *dynamic*, i.e. determined as a function of search progress at any given level.

# 3. COMPOSITION OF CLASSIFIERS

## 3.1 Problem Statement

Supervised machine learning seeks methods to find classifiers which map features to class labels. Since many machine learning algorithms are stochastic [23, 6], they produce different models (classification, prediction, function approximation, etc. herein referred to as classifier)[1] each time they are used. One option is to execute the learning algorithm multiple times and take the single 'best' classifier. However, we might be able to benefit from the composition of

[1]is this not at odds with the simple notion of 'mapping features to class labels'?

multiple classifiers (commonly referred to elsewhere in the literature as an *ensemble*) to produce a better-performing overall classifier. There is also the possibility of selecting a poor classifier. These points raise the following four questions:

- How to combine the classifier outputs to compute an overall classification?
- How to generate multiple diverse classifiers to produce a well-performing ensemble?
- How to set the parameters of machine learning algorithms?
- How can we build high quality classifiers more efficiently in the new era of big data and parallel processing?

While many algorithms used to train classifiers are stochastic, there are a number of exceptions including C4.5 for decision trees, Naive Bayes, and Support Vector Machines [6].

## 3.2 The Solution

One solution to the problem of randomness in classifier training is to adopt well-founded statistical techniques to perform any or all of: combine classifiers; sample the data set; sample the features and generate diverse classifiers. There are a number of methods available which allow us to combine separate classifiers into a single classifier and as machine learning is largely statistical (in respect of both the algorithms to train classifiers and the actual data [16]) it makes sense to turn to a statistical technique to alleviate this issue.

One of the basic tasks of statisticians is to sample a tiny subset of a population in order to estimate the true statistics of the entire population. Likewise, since machine learning algorithms automatically sample a space of classifiers, ensemble techniques allow us to combine them to provide a better reflection of the entire population of classifiers. The law of large numbers dictates that as the sample size increases, the ensemble will converge to better estimates of the true underlying statistics.

Classifiers are functions mapping features to class labels. We can employ one of the following methods to map the outputs of single classifiers to an output for the entire ensemble:

- Majority vote: The entire set of classifiers vote on a class, and the class which receives the most votes is taken.
- Averaging: If the outputs of each classifier are a real number then the outputs can be averaged.
- Weighted average: Each classifier is assigned a weight according to its 'expertise'. When the averaging is done more emphasis is placed on the classifiers with a higher weight.
- Algebraic combiners: real-valued outputs of classifiers are combined through statistical expressions such as sum, mean, product, median, minimum, maximum.

Classifiers in an ensemble should be *diverse*, i.e. they should make different errors and be independent or negatively correlated. There are a number of ways a diverse set of classifiers can be generated from the given dataset. The most obvious is to re-run the machine learning algorithm: however there are more effective methods available, including:

- Bagging (bootstrap aggregation) creates a diverse set of classifiers based on different random samples (usually with replacement) taken from the original dataset [11, 10]. Each classifier is trained on a different subset of the data. As an example, random forests [2] are a combination of classifiers. Each classifier is a tree-model, hence the name random forests. The generalization error of the whole forest tends towards a limit as the number of trees increases, by the law of large numbers. The Kinect motion sensing alogorithm for the Xbox gaming console uses decision tree classifiers [10].
- Boosting adjusts the probability of sampling misclassified data. Thus, misclassified data is more likely to be considered in the training of subsequent classifiers. However, boosting only applies to binary classification but this issue can be addressed by the AdaBoost algorithm [12].
- Stacked Generalization trains multiple levels of classifiers [30]. An ensemble is trained creating level-1 classifiers. The outputs from level-1 are then used to train an ensemble of level-2 classifiers. The outputs from level-1 classifiers can be thought of as features on which level 2 classifiers can make decisions.
- Mixture of Experts generates several classifiers whose outputs are combined through a rule which typically trained using the expectation maximization (EM) algorithm [18]. A hierarchical mixture of experts can be obtained by combining several mixture-of-experts models [19].

Just as bagging and boosting sample subsets of instances, we can also select subsets of features to build diverse set classifiers. Generating different classifiers from randomly-selected subsets of feature is known as the random subspace method [17]. This is particularly useful when unstable machine learning representations are used. Unstable learners are those sensitive to differences in the training data set and include decision trees while stable learners are those less sensitive to differences in the training data set and include nearest neighbour [8].

## 3.3 Consequences

### *3.3.1 Generation of Diverse Classifiers*

As remarked above, classifiers in an ensemble must be diverse. This can be achieved either by altering parameters that determine the architecture of a classifier, or how the space of classifiers is sampled. With an artificial neural network (ANN) [1], this can be achieved by choosing different initial weights, different numbers of hidden nodes or different learning rules. With Evolutionary Computation methods such as genetic programming [20], we choose different numerical parameters such as population size and number of generations, but also different crossover and mutation operators [7]. In fact, when training a diverse set of classifiers intended for an ensemble, our ignorance about appropriate values for parameter settings is turned into an advantage as

we are forced to explore different parameter settings, each combination of parameter settings potentially creating a different classifier.

### 3.3.2 Statistically better behaviour

One major benefit of an ensemble approach is that the combined classifier is the result of all the existing classifiers. Rather than discarding any suboptimal classifiers generated during preliminary parameter setting experiments, they can all contribute to the ensemble. Since the final ensemble results from a larger sample of classifiers, it is more robust: a single execution of a machine learning algorithm may or may not produce an acceptable classifier. However, combining multiple classifiers, each of which came from an independent execution of a machine learning algorithm, is less likely to succumb to statistical fluctuation [8].

An ensemble also exhibits the desirable property of graceful degradation. ssuming a large number of classifiers, should a small number of them fail, the whole ensemble will not fail catastrophically. ANNs also share this property, which was originally modelled on the gradual reduction of brain function (and the lack of a 'grandmother' neuron) [27].

### 3.3.3 Integration of different types of classifier

Part of the broad appeal of ensembles is the integration of different types of classifier. For example a decision tree might be trained by C4.5, and an ANN might be trained by back-propagation [24]. Combining these different types of classifier into an ensemble may have more benefit than combining classifiers of the same type, as each approach will have a different bias. Separate researchers with expertise in different sub-fields of machine learning can collaborate by contributing classifiers to an ensemble. Different machine learning researchers do not need to understand the training method of each other's technique as the classifiers can be inserted directly into the ensemble.

### 3.3.4 Learning Classifier Systems

Learning Classifier Systems offer an interesting solution to the question of how to build a system of classifiers. A set of classifiers is evolved, but each one is 'guarded' by a rule stating which part of the input space it applies to. These condition-action pairs, as they are called in the Learning Classifier Systems literature, can be guarded by relatively simple conditions such as ternary-schema matching or by more complex S-expressions. Thus a single classifier can be considered to be an expert in classifying parts of the feature space. The relationship between Learning Classifier Systems and ensembles has already been investigated [3].

### 3.3.5 Confidence

With a single classifier it is difficult to say what confidence we have in the classification. The outcome of a single classification task is Boolean and we have no further information to make any judgement about the confidence of the classification. However if we have a set of classifiers, we can also give some indication of our certainty in our prediction. For example, imagine we have one hundred binary classifiers. In one case, if we have 99 votes for true and 1 vote for false we can be more confident than if we only received 51 votes for true and 49 votes for false. An ensemble is a probabilistic model which allows us to perform classification and report our confidence in this classification. Thus, we can estimate the posterior probabilities of the classification decisions [25]. It should also be emphasized that combining classifiers does not guarantee an improvement in performance when compared to the best classifier in the ensemble[2] and of course high confidence does not imply that the classification is correct.

### 3.3.6 Levels of Measurement

Using statistics as a fundamental foundation, an ensemble should be able to deal with different levels of measurement (Discrete, Categorical, Nominal, and Ordinal data). [31] defines three types of classifier outputs[3].

- Abstract-level output: output is a unique class label for each input.
- Rank-level output: output is a list of ranked class labels for each input.
- Measurement-level output: output is a vector of real-valued measures representing estimates of class posterior probabilities.

### 3.3.7 Statistics and Machine Learning

There is a tension between statistics and machine learning. Statisticians follow at least a two-stage process when fitting a model to the data (i.e. constructing a classifier):

- Examine data manually and choose a model (e.g. a polynomial of a given degree).
- Choose the best parameters for the model (e.g. by least squares).

Some machine learning techniques attempt to automate the model selection process itself, selecting the most appropriate parameters. For example ANN can represent any continuous, differentiable function, and is not constrained to a highly restricted model [1]. Similarly Genetic Programming can construct any computable function if the function set is expressive enough, while simultaneously choosing any numerical parameters the model requires [24]. In machine learning a model (i.e. a classifier mapping features to class labels) is ratified by A/B testing where the dataset is partitioned into two disjoint sets:

- The training set is used to train the classifier.
- The testing set confirms the validity of the classifier.

Classifiers chosen manually by statisticians are clearly human-understandable. Machine learning techniques may generate more accurate classifiers but be more difficult to interpret. For example, a linear model is easy to understand while ANNs have no such simple interpretation [27]. An ensemble approach offers a seamless marriage between traditional statistical approaches and contemporary machine learning techniques, potentially reconciling some of these tensions. For example, an ensemble could consist of a number of human-selected classifiers and a number of classifiers trained with machine learning techniques. Depending on our perspective, we could increase the weight of the human selected

[2]Is this not at odds with some of the above statements

[3]How do these relate to the above 'levels of measurement'?

classifiers if we demand explanations of the data. If we alternatively demand accuracy, we could increase the weight of the machine learning trained classifiers. In other words, the weights of the different kinds of classifiers provide a sliding scale between the manual and automatic approaches to classification.

## 3.4 Classifier Outputs as Features

In machine learning, features are aggregate information extracted from the raw data of the application domain, the motive being that features are easier for machine learning algorithms to handle than raw data. For example: eye colour of a person could consist of the set {blue, brown, hazel, green} could be extracted from the RGB colour pixels in a photograph which is subject to noise; or a person's height could be discretized into the set {short, medium, tall}. These features are units of information that the machine learning algorithms and the classifiers they produce receive as input. Features form the basis space, from which subsets of features can be selected to reduce the dimensionality of the problem [15] or more aggregate features can be constructed [21].

Classifiers can be considered as a feature-space mapping from the original features to the space of class labels. Similarly, data points in the input space can be used to classify unseen data based on its distance to existing data points, e.g. by using the $k$th nearest neighbour classifier. However instead of using the entire dataset, a subset of the initial set of data points can be used, and therefore mapping the original data to a subset can be considered as reducing the number of features [14].

## 3.5 Ensembles of Linear Classifiers

Ensembles of linear classifiers can learn non-linear decision boundaries. Linear classifiers place a linear decision boundary across the feature space. For example, perceptrons can only express linearly separable decision boundaries and are the 'computational unit' in ANNs. An ANN is similar to an ensemble of perceptrons, with a layer of classifiers feeding into a threshold function (typically a sigmoid function). Assuming three-layer feed-forward architecture, we can think of the first layer as an ensemble. Similarly, the outputs from the second layer can be considered to be a set of classifications which are combined in the third layer. However, it must be remembered that in the case of ensembles each classifier is produced independently, whereas in the case of ANN, perceptrons may not be learned independently.

There is thus a parallel between ANNs and ensembles. A linear classifier places a linear decision boundary across the feature space, and cannot express non-linear decision boundaries. However, and ensemble of linear classifiers can express a convex decision boundary. A single perceptron can only learn and express a linear decision boundary and can never learn to express the logical function XOR (or the general parity problem). It was later discovered that a second layer of perceptrons could express convex decision boundaries, and a third layer could express concave decision boundaries [1]. Perhaps if ensemble approaches had been more prevalent in the 1960s, it would not have taken 20 years between the development of perceptrons and multilayer ANN [27]

## 3.6 Big Data and Parallelism

Ensemble techniques are increasingly important for big data analysis and parallel processing. Techniques for dealing with classification should scale with data volume. Large amounts of data can be sampled and classifier training algorithms can be run on physically separate processors, before being combined into an ensemble. Bagging is particularly disposed to dealing with large amounts of data as it is a divide-and-conquer approach to training a classifier that can be parallelized.

Once a set of classifiers is learned, if more data instances arrive, we do not need to train an entirely new ensemble from scratch. Classifiers which are typically learned off-line on a batch of data, can thus be learned on new sets of data instances upon arrival, and be integrated into the ensemble. We can simply add new classifiers trained on the new data, and include them in the ensemble without altering the existing classifiers. This is of increased importance with larger and incrementally-growing datasets. In summary, both the training of classifiers and the classification of data can be parallelized.

# 4. REFERENCES


[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[2] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[3] Gavin Brown, Tim Kovacs, and James AR Marshall. Ucspv: principled voting in ucs rule populations. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1774–1781. ACM, 2007.

[4] E.K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. R. Woodward. *A Classification of Hyper-heuristic Approaches*, chapter Handbook of Meta-Heuristics, pages 449–468. Kluwer, 2010.

[5] R. M. Burstall. Proving Properties of Programs by Structural Induction. *The Computer Journal*, 12(1):41–48, February 1969.

[6] David Corne, Marco Dorigo, Fred Glover, Dipankar Dasgupta, Pablo Moscato, Riccardo Poli, and Kenneth V. Price, editors. *New Ideas in Optimization*. McGraw-Hill Ltd., UK, Maidenhead, UK, England, 1999.

[7] Tianxiang Cui, Jingpeng Li, John R. Woodward, and Andrew J. Parkes. An ensemble based genetic programming system to predict english football premier league games. In P. N. Suganthan, editor, *2013 IEEE Symposium Series on Computational Intelligence*, pages 138–143, Singapore, 16-19 April 2013. IEEE.

[8] Thomas G. Dietterich. Ensemble methods in machine learning. In *MULTIPLE CLASSIFIER SYSTEMS, LBCS-1857*, pages 1–15. Springer, 2000.

[9] E.W. Dijkstra. Recursive programming. *Numerische Mathematik*, 2(1):312–318, 1960.

[10] Peter Flach. *Machine Learning: The Art and Science of Algorithms That Make Sense of Data*. Cambridge University Press, New York, NY, USA, 2012.

[11] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[12] Yoav Freund, Robert E Schapire, et al. Experiments with a new boosting algorithm. In *ICML*, volume 96, pages 148–156, 1996.

[13] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[14] Roberto Gil-Pita and Xin Yao. Evolving edited k-nearest neighbor classifiers. *Int. J. Neural Syst.*, 18(6):459–467, 2008.

[15] Isabelle Guyon. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

[16] Trevor Hastie, Robert Tibshirani, Jerome Friedman, T Hastie, J Friedman, and R Tibshirani. *The elements of statistical learning*, volume 2. Springer, 2009.

[17] Tin Kam Ho. The random subspace method for constructing decision forests. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(8):832–844, 1998.

[18] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

[19] Michael I. Jordan. Hierarchical mixtures of experts and the em algorithm. *Neural Computation*, 6:181–214, 1994.

[20] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[21] Huan Liu and Hiroshi Motoda. *Feature Extraction, Construction and Selection: A Data Mining Perspective*. Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[22] Helena R. Lourenço, Olivier C. Martin, and Thomas Stützle. Iterated local search. In *Handbook of Metaheuristics, volume 57 of International Series in Operations Research and Management Science*, pages 321–353. Kluwer Academic Publishers, 2002.

[23] Sean Luke. *Essentials of Metaheuristics*. Lulu, second edition, 2013. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[24] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[25] Michael Muhlbaier, Apostolos Topalis, and Robi Polikar. Ensemble confidence estimates posterior probability. In Nikunj C. Oza, Robi Polikar, Josef Kittler, and Fabio Roli, editors, *Multiple Classifier Systems*, volume 3541 of *Lecture Notes in Computer Science*, pages 326–335. Springer, 2005.

[26] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[27] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. MIT Press, 1986.

[28] Jerry Swan, Ender Özcan, and Graham Kendall. Hyperion - a recursive hyper-heuristic framework. In Carlos A. Coello, editor, *Learning and Intelligent Optimization*, volume 6683 of *Lecture Notes in Computer Science*, pages 616–630. Springer Berlin Heidelberg, 2011.

[29] Jerry Swan, John Woodward, Ender Özcan, Graham Kendall, and Edmund Burke. Searching the hyper-heuristic design space. *Cognitive Computation*, 2013.

[30] David H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.

[31] Lei Xu, Adam Krzyzak, and Ching Y Suen. Methods of combining multiple classifiers and their applications to handwriting recognition. *Systems, Man and Cybernetics, IEEE Transactions on*, 22(3):418–435, 1992.