ANALYSIS++: Object-Oriented Framework for Generalized Machine Learning Algorithms

Edward S. Blurock
Johannes Kepler University,A-4040 LINZ, Austria
email: blurock@risc.uni-linz.ac.at

July 20, 1999

Keywords:
Object-Oriented, Framework, Generalized Algorithms,
C++ implementation

Abstract

ANALYSIS++ is a system, written in C++, for the manipulation and development of generalized machine learning methods within a multi-strategy approach. The system's flexibility is a result of the object-oriented principles used in the design of its component frameworks: object, algorithm and run. The object framework is for the management of the data used and generated by machine learning methods. The algorithm framework organizes the machine learning methods and their support routines. The run framework manages algorithms in a goal-oriented strategy. This paper outlines the essential classes of each framework with examples on how they are used in the design of generalized machine learning methods.

1 Introduction

Data analysis involves extracting general principles from a history of known samples. Machine learning [8, 9, 5] is a special class of data analysis algorithms in that it tries to mimic humans either in a biological way, such as neural based algorithms [11] or genetic algorithms [4], or in a psychological way, such as algorithms based on logic[7]. Furthermore, machine learning algorithms usually rely on optimization or search, in contrast to, for example, the deterministic matrix operations of multivariate statistical algorithms.

Data analysis is a complex task involving the interaction of transformations, pre-processing operations and machine learning analyses. The use of many algorithms, culminating in a final analysis, is a 'multi-strategy' approach [6] where one does not rely on a single analysis technique, but recognizes that each analysis algorithm contributes information to the problem as a whole. A consequence of this multi-strategic thinking is the emphasis on the data representation rather than on the algorithm, i.e. one uses the combination of available algorithms to search for an optimal representation.

As evidenced by the multitude of machine learning algorithmss [2], how an analyses problem is approached is not unique and involves a great deal of experimentation. It is the task of the researcher to weed through the variety of learning algorithms to find a suitable combination which solves the problem. This is done through a thorough study of the available algorithms and the types of solutions they offer.

The development of machine learning algorithms has been an evolutionary process. The consequence is that there are large sets of algorithms which are only slight variations of each other. For example, a later variation can have an extra procedural step or can substitute other operations or measures for an existing step. A generalized machine learning algorithm tries to encompass in its design a large set of similar algorithms where the procedures within the algorithm are modularized and can be easily exchanged or eliminated. A generalized algorithm can only be achieved after thorough analysis of a set of similar algorithms and their operational similarities.

The purpose of the system described in this paper is, through an object-oriented design, to provide a software framework for the implementation and experimentation of generalized multi-strategy machine learning algorithms. The hierarchical structuring of the data types is a key element in the design of generalized algorithms. The algorithm itself is written using a generic data type and during execution, the actual data type used is any its derived data types. Specific derived data types derived can be specified statically, i.e. hard coded, or dynamically, i.e. through a specification given at runtime. This technique will be illustrated (Section 3) with the logic classes, where a super logical class is defined and, for example, 'crisp' or 'fuzzy'[13] logical classes are derived. Object-oriented design is also used to create a hierarchial structuring of the algorithms where the super classes describe the generalized algorithms and the derived classes describe the specific variations (Section 3). Defining individual machine learning algorithms as complex composite algorithms interacting with the data set is facilitated by a algorithm framework (Section 4). The combination of machine learning methods, promoting the multi-strategy approach, is managed by the run framework (Section 5).

2 Machine Learning Analysis

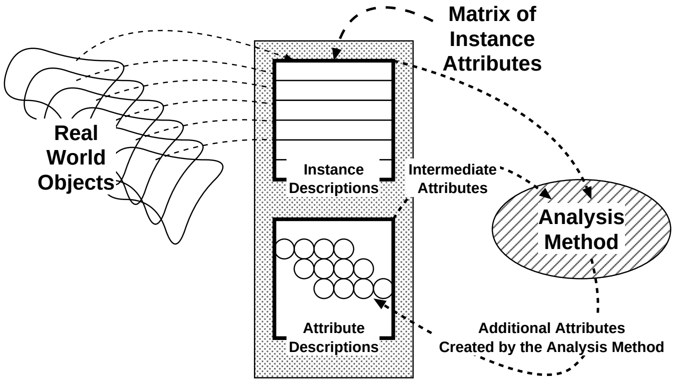

Figure 1: Initially, each 'real world' object is translated into a instance description, i.e. data about the object. Each description is a data type defined in the system. Machine learning analysis algorithms use the instances to find new attributes which are generalized descriptions of the instances.

Figure 1 shows one learning cycle starting with the set of 'real' objects, translating them into a set of instance descriptions, and then using a machine learning analysis algorithm to find new descriptions, such as a predictive function or classification. The descriptions in the instances are with respect to the single 'real' object and the descriptions in the attributes are with respect to the whole set of instances, i.e. a generalization. The complete 'learning' process consists of many such cycles, each cycle creating a new set of descriptions. In the multistrategy approach, each cycle can involve a different analysis algorithm and the descriptions are reused. At the center of this cycle is a data set with two distinct parts:

Instances The history of samples to analyze are the dataset instances. Each instance corresponds to a real object. The attributes within the instances are a representation of the objects, i.e. an approximation of reality. This set of instance attributes is used by the machine learning algorithms to compare and distinguish between the dataset instances to gain insight into the system.

Attributes The result of machine learning algorithms is a set of attribute descriptions of the data as a whole. Typically, in a learning cycle, a set of attributes are suggested (maybe by a previous analysis algorithm) and it is the job of the learning algorithm to pick out and combine these attributes into a generalization of the whole data set.

In the multi-strategy approach, a description generated by one algorithm can be used in the next algorithm. After the final analysis it contains a characterization of the system and the predictive functions representing the generalities and intuition gained in each learning cycle. The data set description has two basic purposes, to provide more intuitive information about the set of instances (and the real world objects they represent) which can be interpreted by a human, and intermediate data which can be used and interpreted by the next analysis step.

Each analysis algorithm can deliver a different data structure which, in turn, can give additional information or justification of previous results. For example, most rule (logical description) induction algorithms require a list of possible predicates as input. A transformation from, for example, a set of real numbers (the values of a parameter, X) to a set of predicates, $X > 2.0$, $X < 5.3$, $6 < X \leq 11.2$, must be made. This could be accomplished by an initial preprocessing analysis to determine suitable boundaries. This preprocessing could be a simple statistical analysis of the maximum, minimums or means, or the result of more complex clustering algorithm. The goal of multistrategy learning is to be able to use and exchange the algorithms and their corresponding data structures efficiently.

An essential aspect of the multistrategy approach is to examine a algorithm or data structure to see what information can be extracted from it. For example, a clustering[3, 12, 1] algorithm builds a hierarchy of 'similar' objects and as one moves down this hierarchy the objects become more similar. Within this data structure, many types of information can be extracted. For example, the original dataset may be too large (in number) or too diverse (in conditions and situations) and each cluster could be used as a smaller more consistent set for further analysis. The automatic isolation of parameter values to be used as boundaries for the predicates (such as $X < 3.54$) could be automatically accomplished by one-dimensional clustering using the bounds of each cluster as the bounds used in the predicate.

An examination of the descriptions of the clusters themselves could reveal the 'significant' parameters or descriptions. If a cluster is formed where 85% of the objects have $X > 85.9$ (as opposed to a random distribution in the parent cluster), then the significance of the parameter `X` is established and if the description, $Y > 3$, has the same distribution as the parent then a hint is given that the parameter `Y` is less significant. Furthermore, under the principle that similar object behave similarly, following an object down a hierarchy of clusters could give a prediction on about its behavior (This is a generalization of Case-Based reasoning).

2.1 Learning Within the Framework

A framework supporting the development of generalized machine learning algorithms within a mult-strategy approach should support at least the following capabilities:

Reuse of Object Information In most cases, the information within an object created by an analysis technique can be used as input to other algorithms. The framework should support this by providing a means of communication between algorithms, sharing of results and use of results in different contexts.

Algorithm Generalization The natural generalization of the data structures should be supported for the generalization of algorithms. Algorithms can be defined in terms of the high level general classes of the objects, and only during the actual execution of the algorithm are the actual specific data types used.

Organization of Tasks A complex algorithm may consist of many subalgorithms or tasks. When implementing an algorithm, one should be able to concentrate on the algorithm itself. Furthermore, the exchange of algorithms within a complex task and the reuse of algorithms in different contexts should facilitated. This also facilitates experimentation of new combinations of algorithms.

One of the aims of the current system is to provide an efficient experimentation framework for the use of algorithms and their combination and the objects they produce. The complete ANALYSIS++ framework is made up of a hierarchy of the following subframeworks:

Object Framework The object framework forms the basis of objects available for attribute (and result) descriptions of the instances under analysis. A hierarchy of objects is provided to promote the exchange of objects between algorithms and their use in generalized machine learning algorithms.

Algorithm Framework The algorithm and goal frameworks represent the characterization and modularization of the tasks within machine learning algorithms and the supplementary tasks needed for their setup and use.

Run Algorithm The run framework manages the execution of algorithms and the data objects produced. Modularization of tasks is promoted through the separation of task prerequisites and the actual job that the task has to perform.

3 Object Framework

A significant part of developing generalized algorithms is the definition of more general or generic types. This is accomplished in the object framework. Data classes can be combined and instantiated into new data types both at compile time, where more specific classes are derived from super data class types, and at runtime, where composite classes can be built up through a runtime specification.

The representation of the instances and descriptions is shown in the class structure of Figure 2. The base class of the framework is the `Object` class. The `SetOfObjects` class holds lists of classes derived from `Object`. `Instance` is a `SetOfObjects` with the attributes descriptions of the instance. The `SetOfInstances` class holds the matrix of instances and attributes (i.e. the set of instances) and is itself a `SetOfObjects` holding the descriptions over the entire data set. `SetOfInstances` corresponds to the data of Figure 1.

3.1 Expandable Class Structure

Associated with each value class, where the value pertaining to a description is instantiated, is an type class, describing its fundamental characteristics. The hierarchy of the set of data value classes is exactly the same, i.e. dual to, the set of data type classes. The set of possible data types are collected together in one place in a data type list. The values, as used in descriptions, have many instantiations in, for example, the instances and attributes.

The foundational data type classes provide the 'hard-coded' classes from which data types can be derived. Some foundational data types have no ad-

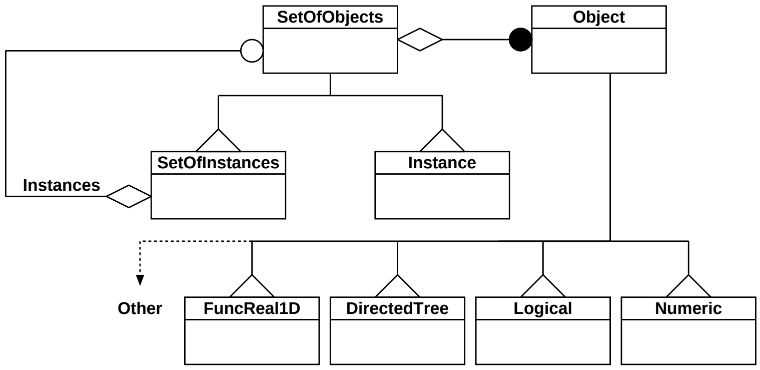

Figure 2: All objects stem from the basic object type, `Object`. The `SetOfInstances` is the data set described in figure 1, where the attributes are stored in the `SetOfObjects` class from which it is derived and the **Instances** is a member of type `SetOfObjects`. Each object in the **Instances** is a `Instance` class, which, in turn, is also a `SetOfObjects`. Each class derived from `Object` is a description, either an instance description or an attribute description.

ditional information and serve as place holders in the list of data types. However, other foundational data types have addition information, i.e. members, which need to be specified at runtime to complete the class definition. There are basically three levels to the data specification:

Foundational Data Type This is the hard coded (compiled) data type class available. It provides the pattern for the instantiated data type.

Instantiated Data Type When the foundational data type class is instantiated, a specific attribute data type is defined. Some of these data types require additional information to be instantiated to a complete data type. Others require no additional information and the foundational data type class is itself complete and no new runtime class would be defined from it. When a data type is instantiated, it is put in the list of data types.

Instantiated Data Value This is the instantiated value belonging to the corresponding instantiated data type and make up the description data objects.

Some data types, such as **Real**, have no additional information to be specified and thus no new class can be derived from them. They would instantiated once and put in the list of data types. However, many instantiated values of the type, for example **Real** values, can be found in the instance and attribute descriptions.

Some data types require additional information to be a complete data type. A simple example is a nominal attribute representing a set of colors. The foundational data type class for any nominal attribute is the class **NValuedLogic** (which is a class derived from **Logical**). The instantiation of this class to the new data type of **Colors** is done by adding the three labels, **red**, **green** and **blue** to an instantiation of the **NValuedLogic** data type class and given the name **Colors**. The class would then be added to the list of data types. A specific data value (as used in an instance) of this data type could be, for example, **green**.

Some foundational data type classes are composite classes of several other foundational data type classes. The instantiation to a specific data type is the dynamic instantiation of the data types making up the compound class. For example, within Figure 3 the foundational data type class **NumericPredicate** is a predicate function in which a real value is mapped to a logical value. It main use is to form fuzzy logic predicates. Its definition needs a logic class (**Logical**) and a numeric function class (**FuncRead1D**). The logic class specifies the form of the logic that the predicate will return and the numeric function specifies the functional dependence, which, for fuzzy logic, maps the real numbers to a number between zero and one. To instantiate, for example, the **FuzzyPredicate** data type derived from the **NumericPredicate** data type, the **FuncReal1D** and **Logical** data types within **NumericPredicate** are instantiated with **PlateauFunction** and **Continuous**, respectively. The **FuzzyPredicate** data type represents a typical fuzzy predicate type, i.e. one made of up line segments and is stored in the list of data types. The instantiation of a specific data value, in this case a specific fuzzy predicate, the **NumericPredicate** data value class is instantiated with a **PlateauFunction** data value class with the parameters specified.

A common description used in machine learning rule induction methods asks a question about a specific parameter; for example, is the parameter **X** (of an instance in the dataset) more or less greater than 2 (a fuzzy logic predicate). The **Predicate** foundation data type class is used to define such questions and is composed of a logical **Operation**. In the instantiation of the data type, **Operation** must be specified. Since, in the example above, **FuzzyPredicate** is derived from **NumericPredicate** which, in turn, comes

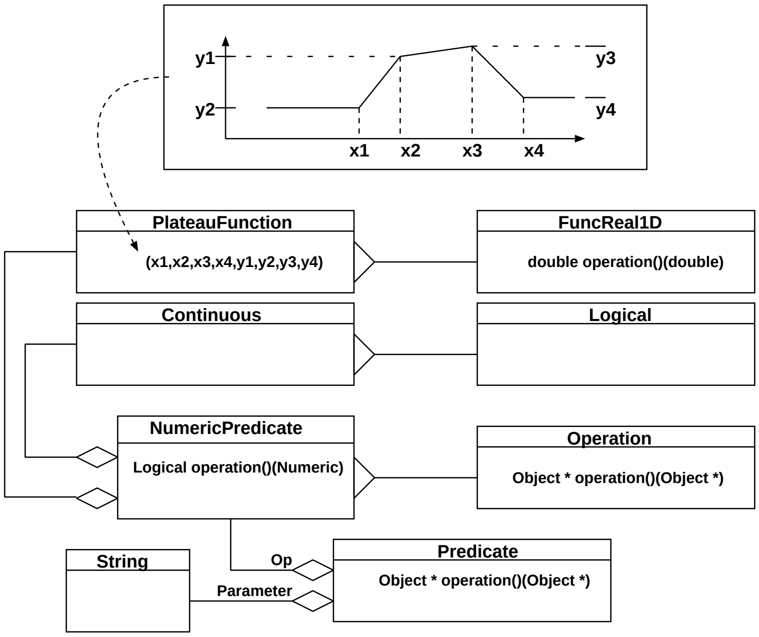

Figure 3: The interaction and building of a complex data structure from the frame given is shown. The 'Predicate' object has two strings for the parameter name (to be found for each example instance) and the logic type of the predicate. The predicate itself is made up of a numeric predicate (a function mapping a number to a logic value). The numeric predicate, in turn, has two slots for the logic type and the numeric function type. The text shown is the 'dynamic' definition of a particular predicate class.

from `Operation`, the newly defined `FuzzyPredicate` can be specified as the `Operation` in a new `Predicate` class. This is an example where a new composite data type can be dynamically defined in terms of a data type that itself was dynamically defined.

3.2 Hierarchy of Data Structures

One way to facilitate using the information in a data structure is to define the data structures in a hierarchy of increasingly detailed classes. At each class level, a given aspect of the data structure is available. Progressing up the hierarchy, more detailed and hence other aspects of the data are made available.

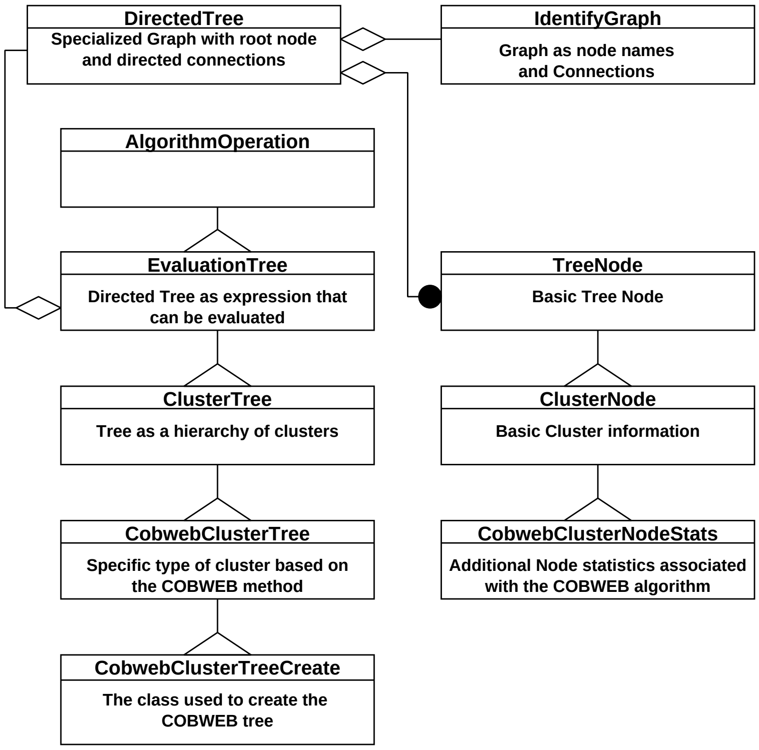

Figure 4: Part of the organization of the algorithms is the hierarchy of functionalities. The algorithm is interpreted at the different levels in different ways. Going down the hierarchy specifies addition functionality and interpretations.

One way to use the hierarchial structuring of data objects is in the definition of generalized algorithms. In the code of the algorithm, only the super class would be specified. During execution, then any derived data type could be used. To design an algorithm in this way, a thorough analysis of the commonalities of how the data structure is used within the algorithm is needed. The common aspects would then be represented as virtual operations in the super class.

For example, in a generalized algorithm which involves logical objects, only the super class `Logical` would appear in the implementation. But during execution, the logical class `Colors`, for example, could be dynamically defined and since it is ultimately a derived class of `Logical` (through `NValuedLogic`) it can be used within the algorithm. Furthermore, since, for example, the fuzzy logic class, `Continuous`, also stems from `Logical`, the same generalized definition describes both a fuzzy logic implementation and a 'crisp' logic (through the `NValuedLogic` class) implementation.

Sometimes a set of similar algorithms can be defined in terms of a hierarchy of algorithms where, as one progresses through the derived classes, more specific variations of the algorithm are defined. An example of one branch of such a hierarchy leading to the COBWEB[3] clustering method is shown in Figure 4. The resulting data structure is, at one level, an `EvaluationTree`. Here, the tree structure is used to make predictions about other 'new' objects and how they fit in the tree. Also branching from this class is are the generalized decision tree method (derived from the original Quinlan[10] formulation) where the tree is not a cluster, but a decision tree. But the decision tree and the cluster tree have aspects in common and these are found in the `EvaluationTree` class.

At the next level, `ClusterTree`, the data object is considered a tree of clustered objects. Here, information about the clustering, for example, which objects are grouped together and at what level, can be accessed. Branching from this level are other versions of clustering methods, such as standard distance clustering. Though the derivation of the cluster is different in all the derived methods and also the descriptions of each cluster may vary, `ClusterTree` holds the fundamental properties of a cluster of objects that are common to all clustering methods.

The next level is the produced directly by the method itself, `CobwebClusterTree`. At this level a specific description of the individual clusters is available. The last level, `CobwebClusterTreeCreate` is the class which is used to create the tree from the dataset[3]. A similar hierarchy is shown for the nodes within the trees.

The natural structure of the relationships (and commonalities) between the different structures that different machine learning algorithms are mirrored. It is, however, left to the developer to find, through extensive analysis, the new method's place in the hierarchy. If a concept does not pass into the well defined hierarchy, then coercion would be used to create a data structure that is in the hierarchy. It is especially important in the implementation of the initial set of algorithms of a given type that an initial efficient and useful structure is given on which others can be built. It is a major goal of the framework to provide a means to experiment with implementation structures.

4 Algorithm Framework

An essential facility towards the development of and experimentation with machine learning methods, especially in a multi-strategy approach, is the reuse, exchange and combination of algorithms. The algorithm subframework accomplishes this through the characterization and modularization of tasks. It structures the selection of the set of available 'hard-coded' operator and algorithm classes through characterization within a selection tree mechanism. The philosophy is to isolate tasks and to provide a mechanism to form complex algorithm combinations.

In the last section, the `Operation` class was used in the definition of the `NumericPredicate` class to specify a particular functional relationship defining, as in the example, a fuzzy predicate. This is a typical use where the `Operation` class represents a place where a particular method can be 'plugged in'. The `Algorithm` class is the next level modularization of method definition. The main extension is that the `Algorithm` interacts directly with the instances and attributes of the data set (as opposed to `Operation` which just has a fixed operator input and output structure). Furthermore, `Algorithm` is designed to facilitate the dynamic use of other plug-in operations and algorithms. The generalized machine learning methods (and their major components) are defined as algorithms.

An example of a composite algorithm is the generalized decision tree method. It generates a decision tree consisting of the generalized logic and generalized predicate data structures explained in the previous section. Furthermore, the decision tree algorithm is dependent on three additional algorithms which are dynamically specified at runtime:

Measure The measure for ranking how good a given predicate is relative to the goal description of the decision tree.

Selection A procedure that uses the measure value to select out the predicate to be used on the node of the decision tree.

Filter This filters out unwanted nodes in the decision tree.

To promote modularization and ease of implementation (and to reduce errors) an algorithm is structured into six virtual functions, each of which has a well defined and isolated task:

SetUpAlgorithm This selects the subalgorithms to use from the algorithm selection tree (See section 4.1). This is where, for example, the measure, selection and filter algorithms of the decision tree algorithm are set up.

CheckInput This checks whether the required input parameters from the attribute list are available.

SetUpInput This instantiates the input parameters from the available attribute values in the data set. This routine could also do some preprocessing on these attributes. A Typical attribute value needed by machine learning methods is, for example, the list of instances to be used to build the description.

Calculate This is the heart of the algorithm and assumes that all the necessary information has been set up. This could also set up the general procedural structure of the algorithm in terms of virtual functions found with the current class and substituted in variations defined by the derived classes.

WriteOutputValues This writes out the results as intermediate values (See section 5).

ConcludeRun This concludes the run by cleaning up objects used by the algorithm.

This breakdown structures (and isolates) the essential tasks associated with algorithm execution.

4.1 Algorithm Selection Tree

The goal of the algorithm selection framework is to provide an organizational structure to the classes of algorithms. At compile time, the entire set of algorithms is an unstructured list with no interrelationships or characterizations. The selection tree is built at runtime to dynamically structure the

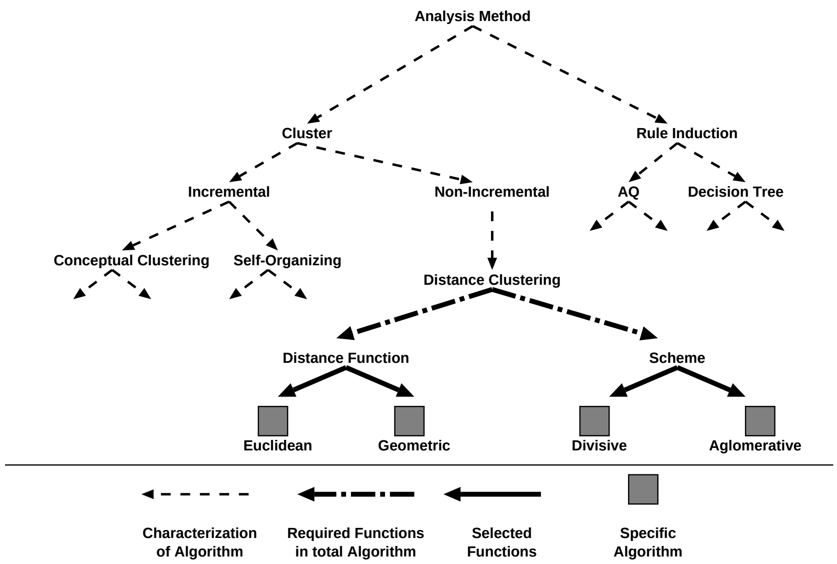

Figure 5: The tree structure provides an organizational level on top of the set of individual algorithms. The top level is the characterization levels, which are keywords isolating the type and general function of the algorithm. At the next level, a particular algorithm may be composed of a set of subalgorithms which can be selected.

algorithms to suit the particular application. The organization is a three level hierarchy:

Characterization Through external analysis of the algorithm, the purpose, resulting data structure and type of the algorithm is determined. This information is structured in a hierarchy of keywords and gives a 'semantic' characterization and organization of the algorithm.

Selection Given a category or type, the selection of the specific algorithm begins. This level refers to a specific implemented algorithm.

Configuration A specific algorithm may be a composite of several subalgorithms; Which set is needed is given at the level directly below the 'selection' level. The configuration pointing to the specific algorithm can have many levels.

The key to the organizational structure of the algorithms is the 'algorithm selection tree'. Figure 5 shows a portion of a selection tree specifying a cluster algorithm. All the non-leaf nodes in the tree are keywords and can be specified by an ASCII or GUI interface. Depending on the position in the tree the keywords are either characterizations, selections or configurations. The leaves of the tree are the selected algorithms. The characterization (as shown in the example) could refer to the type of algorithm, like 'clustering' or 'rule induction'. However, the purpose of the analyses could also be characterized, like 'attribute reduction', specifying methods that can be used to reduce the number of attributes, or 'dataset simplification', to cluster a disperse set of instances into more similar groups for more exact analysis; in future uses, these could be the basis of a more sophisticated use characterization involving 'reasoning'. It should be noted that the leaf nodes of the selection tree are not limited to `Algorithm` classes and could be used as a general selection mechanism for any data structure.

The selection level refers to a specific algorithm. If the algorithm is made up of or needs several subalgorithms, then the configuration of the algorithms guides the choice. The nodes directly below the 'selection' algorithm are the categories of algorithms needed. Under these nodes is a hierarchy leading to specific algorithms. For example, within a given clustering algorithm, there are two choices of operations making up the algorithm, the distance function, for example, geometric or euclidian, and how this distance function should be used, for example, agglomerative or divisive.

5 Run Framework

The `Run` framework manages the execution of algorithms and the access to data objects. Figure 6 shows the class structure. The `AlgorithmRun` framework specifies and runs an algorithm and the `GoalRun` framework, inheriting the `AlgorithmRun` framework, gives an added semantic to the algorithm by defining it as a goal; each of these has essentially two levels:

Summary The `AlgorithmSummary` and `GoalSummary` classes use string name specifications (`KeyWords` and `ClassNamePairs`) allowing for a straightforward interface specification (ASCII or GUI).

Run The `AlgorithmRun` and `GoalRun` frameworks hold the instantiated input, intermediate and output objects used during the execution of a goal or run.

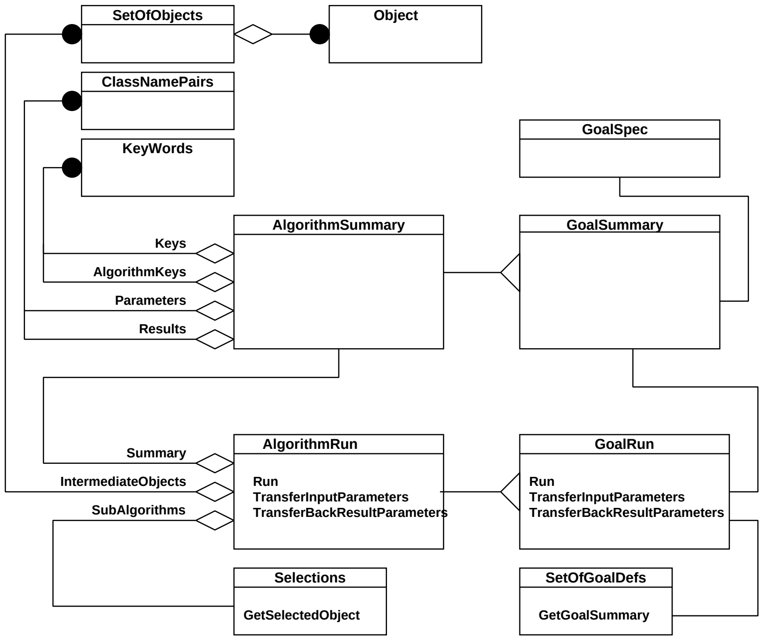

Figure 6: The object structure of the goal and algorithm framework is shown. The goal structures encapsulates algorithm information. The `Keywords` is a list of string names and `ClassNamePairs` is a list of pairs of string names, the first being a parameter name and the second the data type class of the parameter.

5.1 Algorithms

The `AlgorithmSummary` holds the essential information for a complete algorithm specification. This specification consists of keywords (`KeyWords`) or lists of name pairs (`ClassNamePairs`):

Keys To guide the execution of an algorithm, keywords can be used to make decisions. This is a list of string labels.

AlgorithmKeys This is the set of keywords used to traverse the algorithm selection tree to select the subalgorithms and operations used by the algorithm.

Parameters Some algorithms need extra parameters which, in principle, can be any data type. This specification gives a list of labels (and their corresponding type) to be accessed from the dataset descriptions when the algorithm is run.

Results This the list of data structures, created by the algorithm, that will be transfered back to the dataset.

Instantiation of the `AlgorithmRun` class uses the `AlgorithmSummary` class to select parameters from the attributes, specified by **Parameters**, and the needed algorithms, selected by **AlgorithmKeys** (stored in `Selections`). During the execution of the algorithm the intermediate results are stored within this class and the results, specified by **Results**, of the algorithm come from this class. The essential member functions run the algorithm, `Run`, transfer the needed input parameters from the dataset, `TransferInputParameters`, and transfer back the resulting parameters, `TransferBackResultParameters`.

5.2 Goals

The goal framework inherits the algorithm framework and coordinates the interaction between algorithms and gives a level of characterization to the algorithm. Associated with each goal is an algorithm and a set of prerequisite goals. The `GoalSpec` is a simple specification of a specific goal. Within the `GoalRun` the set of prerequite goals are specified in the `SetOfGoalDefs`, a list of `GoalSpec` classes.

The prerequisites of a goal would be 'resolved' (several levels of resolution may take place because the immediate prerequisite goals themselves may have goals to satisfy) and satisfied (through execution of their corresponding algorithms) before the current goal algorithm is executed. The current goal is executed by executing the corresponding algorithm. At present the use of goals and goal satisfaction is an organization tool managing algorithmic flow. Figure 7 shows an example of a set of goals with prerequisites leading to an execution order. The goal organization is not defined at compile time, but rather is a database of goals specified at runtime.

Goals and their resolution provide a higher order means of decision making. In its simplest form, a goal can be used as a higher level command representing a common task. It is a prerequisite for the use of 'reasoning' in deciding which tasks to perform. Goals are an efficient way to express high order tasks that are to resolved in the details of actual execution. It provides a mechanism to shield the user from details. For example, it is the simplest form of task resolution. Before a specific algorithm can be executed, usually specific data structures need to be present. The management of tasks to be done and those already done can be simplified.

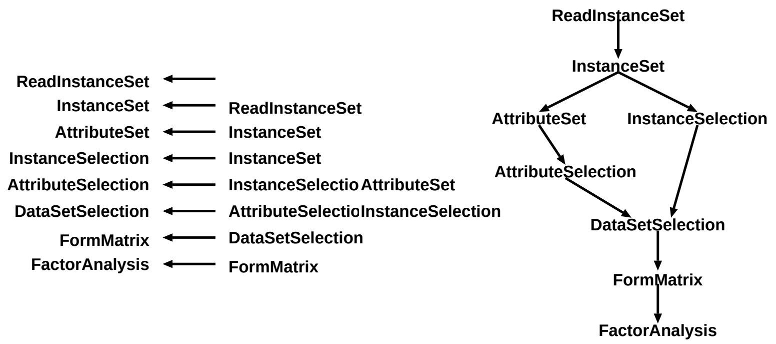

Figure 7: The set of goals as a set of rules and the execution sequence they represent is shown.

6 Discussion and Future Directions

The ANALYSIS++ system is a framework supporting the multistrategy approach. An important aspect of this support is the modularization and configuration of tasks. Each individual task is kept small enough to be manageable. Furthermore, breaking down large tasks into smaller tasks promotes flexibility in combining algorithms in different ways as needed. All configurations, whether it be structuring the set of algorithms, configuring the algorithms and goal relationships or executing a specific analysis problem, are done using keywords which can come from either an ASCII or GUI interface.

The division of the `Object`, `Operation`, `Algorithm` and `Goal` components allows, in an implementation, the developer to concentrate on a single task and other details are transparent or assumed. For example, in the development of a new machine learning algorithm, details of the dataset setup, parameter availability, prerequisite operations, how it is called and other tasks not directly associated with the specific algorithm can be assumed and the developer can concentrate on the details of the new algorithm. Another aspect of the modularization is a (restricted) interface of function definitions. This includes not only the allowed functions at each level of a hierarchy of objects, but also, a more artificial, breaking down of a function definition into its component parts.

6.1 Non-Uniqueness

One of the difficulties in an initial implementation of machine learning algorithms is that the (hierarchial) structure of the algorithms is definitely not unique. A fundamental decision that needs to be made is how many levels of classes are needed. Often, in a redesign, an extra class, which facilitates the implementation of more variations of the derived algorithms, will be inserted in the hierarchy. For example, for the `CobwebClusterTree` class, it was initially determined that at least `EvaluationTree` and `ClusterTree` classes have commonalities with other tree producing algorithms. In a redesign, since the clustering methods divide into basically two types, divisive, where the cluster is formed top-down, i.e. the entire set of instance are divided up into clusters forming the next level, and aglomerative, where the clusters are formed bottom-up, i.e. clusters are made by pairing individual instances and then pairing groups. For the `CobwebClusterTree` this would insert class `DivisiveClusterTree`.

A more difficult problem is that a given hierarchial structure is made from a specific operational viewpoint of the algorithms. Sometimes these viewpoints are not separable and one must choose one structure or the other. For example, machine learning algorithms can be viewed in terms of how the instances are introduced, either one at a time, i.e. incremental methods, or all at once, non-incremental methods. For example, the COBWEB method in the example is actually an incremental method and most clustering methods are non-incremental. But this viewpoint cannot be introduced in the current hierarchial structure. This means that a design decision was made to emphasize the fact that these derived algorithms are producing tree structures and not the incremental or non-incremental aspects.

References

[1] Walter F. Bischof and Terry Caelli. Learning structural descriptions of pattern: A new technique for conditional clustering and rule generation. *Pattern Recognition*, 27:689–697, 1994.

[2] Garry Briscoe and Terry Caelli. *A Compendium of Machine Learning.* Ablex Publishing Corporation, Norwood, New Jersey, 1996.

[3] John H. Gennari, Pat Langley, and Doug Fisher. Models of incremental concept formation. In J. G. Carbonell, editor, *Machine Learning: Pardigms and Methods*, pages 11–61. MIT Press, 1990.

[4] David E. Goldberg. *Genetic Algorithms.* Addison-Wesley, 1989.

[5] Pat Langley. *Elements of Machine Learning.* Morgan Kaufmann Publishers, Inc., 1996.

[6] R.S. Michalski, I. Mozetic, J. Hong, and N. Lavrac. The multi-purpose incremental learning system aq15 and its testing application to three medical domains. In *Proceedings of the American Association for Artificial Intelligence Conference (AAAI)*, pages 1041–1045, 1986.

[7] Ryszard S. Michalski and Robert E. Stepp. *Machine Learning: An Artificial Intelligence Approach*, chapter Chapter 11: Learning From Observation: Conceptual Clustering. Morgan Kaufmann Publ. Inc., 1983.

[8] Ryzard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell, editors. *Machine Learning: An Artificial Intelligence Approach.* Morgean Kaufmann Publishers, Inc., 1983.

[9] Tom M. Mitchell. *Machine Learning.* WCB/McGraw-Hill, 1997.

[10] J.R. Quinlan. Induction of Decision Trees. *Machine Learning*, 1:81–106, 1986.

[11] D.E. Rumelhart and J.L. McClelland. The basic ideas in neural nets. *Communications of the ACM*, 37(3):87–92, 1994.

[12] J.T. Tou and R.C. Gonzalez. *Pattern Recognition Principles.* Addison-Wesley, 1974.

[13] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.