Finite State Classifiers

There is a lesser known NP-complete problem that has a great bearing on AI and language acquisition, a problem I call the Regular Classification Problem. The task is as follows

Regular Classification:
Given a partition $P_0, \ldots, P_{n-1}$ of finite subsets of X^* (where X is finite); find the simplest Finite State Classifier that correctly classifies all the strings in all the partitions.

The output is a series of Finite State Classifiers (FSC) each of which is no larger than the previous. The last Finite State Classifier is a solution with the minimal number of states. All the preceding solutions in the series with the same number of states represent all the other minimal state FSC's.

A *Finite State Classifier* over an alphabet $X = \{x_0, x_1, \ldots, x_{m-1}\}$ of size $m \geq 0$ and a set $C = \{[0], [1], \ldots, [n-1]\}$ of size $n \geq 0$ is a system of linear equations over the set of states $Q = \{Q_0, Q_1, \ldots, Q_{q-1}\}$ of the form

$$Q_j = c_j + \sum_{i \in m} x_i q_{ij} \quad (j \in q)$$

where the $c_j \in C$ for each $j \in q$, and the $q_{ij} \in Q$ for each $(i, j) \in m \times q$.

The solution to this system of equations in terms of the state Q_0 is a regular expression of the form:

$$Q_0 = P_0[0] + P_1[1] + \ldots + P_{n-1}[n-1]$$

where

$$P_0 + P_1 + \ldots + P_{n-1} = X^*, \quad i \neq j \rightarrow P_i \cap P_j = \emptyset .$$

The regular sets, $P_0, P_1, \ldots, P_{n-1}$, represent the partition generated by the FSC, which I call a *Regular Classification*.

A finite state automaton is a special case of a finite state classifier in which there are 1 or 2 classes: accept and/or reject. This generalization allows a more uniform and symmetric treatment of the problem. Thus, as a special case of this problem is the problem of finding the automaton that best fits a sample set consisting of both positive and negative examples – Regular Language Acquisition.

A solution is guaranteed to exist in which the number of states is equal to the input size (measured as the total of all the lengths of all the strings.) A brute force search algorithm can then be used to search through all the automata with this number of states or less that will correctly generate the partition given. This process is exponential in the input size in the worst case.

In the output, some entries may be left blank. These can be filled in with any appropriate entry. An state transition left blank can be filled in by any of the automaton's states, and an empty state class can be filled in with any of the partitions of the input set.

Input Format

The input consists of a list of strings, one each to a line. On the first column of the line is a single character used to represent the string's class. The string itself is listed on the remaining columns of the line. Any character following a backslash is treated literally. This is used particularly to allow the end of line to be included in a string: any line ending in a backslash is considered to continue on the next line with the intervening end of line marker included in the string.

No string can exceed 256 characters in length. Empty lines are ignored. No string can be listed more than once unless each listing puts the string in the same class.

Output Format

A minimal Finite State Classifier is constructed that generates the classification given. This classification is then listed in equational form. The classification character is listed within square brackets, e.g. [4], [*].

The Algorithm

Depth First Search is used to find a solution, though not necessarily the best solution. Initially, the FSC is assumed to have one state (Q_0). The strings are processed in increasing order of length and the algorithm will trace through the FSC as it reads through each string.

When a point is reached where a state transition is undefined a branch occurs. This transition can be safely assumed to be either to one of the existing states or a new state. Therefore if there are currently N states, there will be $N+1$ branches. The order in which the transitions will be tried is $Q_0, Q_1, \ldots, Q_N$. Whenever the choice for Q_N is forced by exclusion, the number of states will be increased to $N+1$.

When the end of a string is reached, the string's class will be compared to the class of the current state. If this state has no class, one will be defined. If the state's class differs from the string's class, a mismatch occurs.

When a mismatch occurs, all the actions carried out from the most recent branch are undone and the next branch is tried. If necessary during backtracking the number of states may drop back down to its earlier levels. If there are no more branches left to try, then the branch point before the most recent is retreated to. Backtracking continues as far as necessary until either a branch point is reached with an untried branch or until all the branch points have been exhausted. In the latter case the algorithm is complete.

After a solution is found it is saved. If it contains N states then from here on out all branch points are pruned to N states, including the ones already active. Backtracking is then performed just as before and further solutions are sought out.

The result is a series of solutions each with no more states in it than the previous, possibly fewer. The last solution found before the algorithm completes will be one with the fewest number of states. Others before this may have the same number of states, as well. At least one solution will be found.

The method of search represents a compromise between Best First Search, which tends to "thrash", and pure Depth First Search, which tends to get stuck on primrose paths. Its worst case behavior may be exponential, and definitely is if $\mathrm{P} < \mathrm{NP}$, but I haven't proven it to be (nor have I proven that $\mathrm{P} = \mathrm{NP}$).