Generating C4.5 Production Rules In Parallel

Richard Kufrin
National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign
Urbana, IL 61801
rkufrin@ncsa.uiuc.edu

Abstract

Induction systems that represent concepts in the form of *production rules* have proven to be useful in a variety of domains where both accuracy and comprehensibility of the resulting models are important. However, the computational requirements for inducing a set of rules from large, noisy training sets can be enormous, so that techniques for improving the performance of rule induction systems by exploiting parallelism are of considerable interest. Recent work to parallelize the C4.5 rule generator algorithm is described. After presenting an overview of the algorithm and the parallelization strategy employed, empirical results of the parallel implementation that demonstrate substantial speedup over serial execution are provided.

Introduction

Algorithms for supervised learning have been widely studied in the artificial intelligence community over the past several decades. With the recent surge of interest in systems for knowledge discovery in databases (KDD), many of these algorithms have been incorporated within KDD systems and have been described as the *data mining* components of a knowledge discovery system (Fayyad, Piatetsky-Shapiro, & Smyth 1996). *Rule induction* systems are particularly valuable for those applications where both accuracy and comprehensibility of the generated models are important (Langley & Simon 1995).

One of the most widely-used types of rule induction classifiers is the *decision tree*, which employs a recursive divide-and-conquer strategy to partition training instances into disjoint regions of the hypothesis space according to preassigned class labels. The techniques and heuristics employed in the well-known C4.5 and CART algorithms have laid the foundation for the many of these types of classifiers in use today. This paper assumes the reader is familiar with the basic ideas underlying such algorithms; thorough discussions may be found in (Quinlan 1993; Breiman *et al.* 1984).

Among the most frequently-cited advantages of decision trees over alternative classification algorithms are speed of execution and comprehensibility of the generated model. However, these advantages may diminish as the induction task becomes more complex; Quinlan (1993) acknowledges this phenomenon with respect to the comprehensibility of C4.5-generated trees:

> ...it is often possible to prune a decision tree so that it is both simpler and more accurate. Even though the pruned trees are more compact than the originals, they can still be cumbersome, complex, and inscrutable. If we want our classifiers to provide insight as well as accurate predictions on unseen cases, there is still some way to go!

The remedy to the problem of overly-complex decision tree models that is pursued in the C4.5 system is to generate *production rules* from decision trees to obtain more compact and comprehensible representations that (hopefully) do not sacrifice classification accuracy on unseen instances (Quinlan 1993). An additional benefit is the ability to combine multiple decision trees into a single rule set (Quinlan 1987). Further, production rule formulations can also alleviate the *replication* problem associated with decision trees, in which subtrees are duplicated within the tree (Pagallo & Haussler 1990).

Figure 1 is an example of a complex (pruned) decision tree induced from a relatively small (5,000-example) real-world training set. The problem domain involves classification of sleep stages based on various physiological measurements of patients admitted to a sleep disorders clinic (Ray *et al.* 1986). Even after simplification by pruning, the tree contains 433 leaves — a considerable improvement over the unpruned tree (763 leaves), but still rather difficult to comprehend. Processing the decision tree with the production rule generator C4.5RULES yields a set of 99 rules that show a modest (1.3%) improvement in classification accuracy over the pruned decision tree when applied to a test set of 2,500 unseen cases.

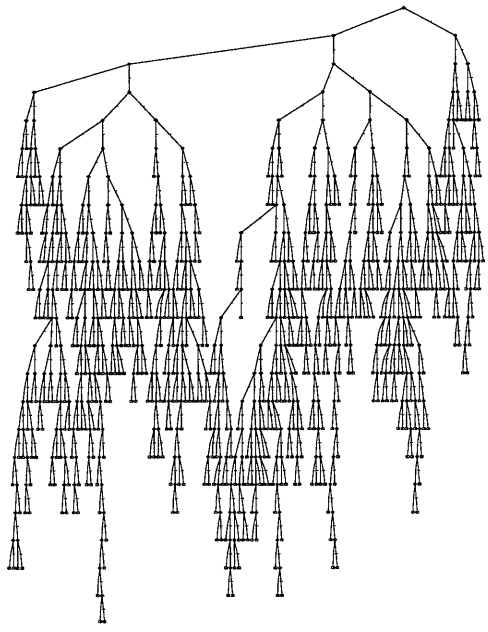

Figure 1: Simplified decision tree induced from a portion of the sleep stage data set.

Computational Requirements

Unfortunately, these gains may come at greatly increased computational expense. Figure 2 shows the growth of both decision tree induction and rule generation for the sleep stage data as the training set is increased from 2,000 to 32,000 examples. While the time required for tree induction remains tolerable, growing to only 100 seconds for the largest training set, rule generation consumes considerably larger amounts of time, requiring over 76,000 seconds (over 21 hours) to process the decision tree induced from the largest training set. Similar results have been observed for both artificial and real-world data sets (Cohen 1995; Domingos 1996).

Clearly, more efficient implementations of the rule generation strategy of C4.5 are required to allow its use on large, noisy training sets. One approach to enhancing performance is the use of parallel processing; the remainder of this paper describes a parallel implementation of C4.5RULES that can be of use when confronted with computationally-demanding problem domains. After providing an overview of the C4.5 rule generation algorithm, a strategy for parallel execution is presented, followed by empirical results on both artificial and real-world training sets.

Rule Generation in C4.5

Unlike many strategies that employ a *separate-and-conquer* approach to finding a rule set that covers the training instances (Fürnkranz 1996), C4.5 *extracts* rules from existing decision trees.

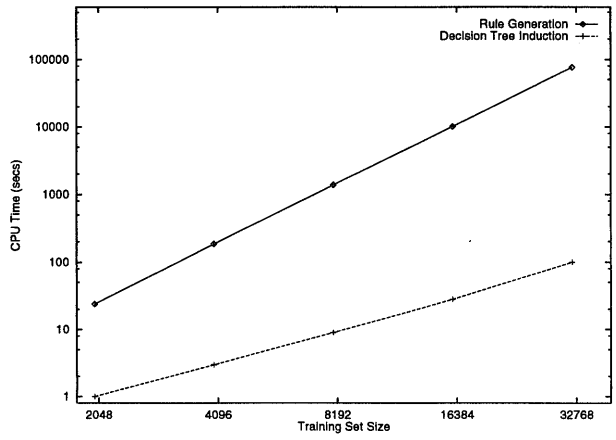

Figure 2: Comparison of time required for decision tree induction and rule generation (log scale). Sleep data used as the training set.

The goal of the procedure is to produce an ordered set of **if-then** rules of the form:

if LHS **then** RHS

where LHS is a conjunction of attribute-value tests, called *conditions*, and RHS is a class assignment. During classification of an instance, the first rule for which all conditions specified in the LHS are satisfied by the instance "fires", and the class label specified in the RHS is assigned to the instance.

Because the hypothesis space is usually not completely covered by the rule set, a *default class* is also provided – this is the class label assigned to any instances that are not covered by any rule in the rule set. The four steps in the process of rule generation are:

1. **Pruning**. First, an initial rule set is constructed by converting each path from the root of the tree to a leaf into a rule, where each condition in the LHS of the rule corresponds to an internal node of a path in the decision tree. A decision tree with l leaves thus yields an initial rule set of l rules. Each of these rules in examined in turn, and conditions are removed from the LHS that do not appear to contribute to the estimated accuracy of the rule, resulting in a more general, "pruned" rule that is added to the intermediate rule set if it does not duplicate an existing rule.
2. **Selection**. The pruned rules constructed in the previous step are grouped into *class rulesets*, one for each of the k classes in the training set (i.e., the RHS of each rule in subset i is class C_i). Each of these rulesets is further scrutinized to select a subset of these rules that will maximize predictive accuracy for the class associated with the ruleset.
3. **Ordering**. The k rulesets are ordered according to the frequency of false positive errors and a default

class is established by determining which training cases are no longer covered by any of the current rules: the most frequent class of these cases is designated the default.

4. **Evaluation**. The ruleset as a whole is evaluated against the training set to determine if any rules affect classification error adversely. If so, the "worst" such rule is removed and evaluation is repeated until no further improvement is possible.

Parallelization Strategy

Of the four steps for rule generation outlined in the previous section, pruning and selection are typically the most time-consuming, with the latter often dominating total execution time.

There are several possible dimensions for parallel execution. Assume a k-class problem with m training instances, r rules, and p available processors. *Rule-based* parallelism associates each processor with r/p of the currently-active rules. An *instance-based* approach instead divides the training set into "blocks" of size m/p and assigns each block to a separate processor. *Class-based* parallelism is an alternative that involves associating each of k processors with the class ruleset corresponding to one of the possible classes.

The parallel implementation of C4.5RULES described in this section provides for parallel execution of all steps except rule ordering. The decomposition strategy varies, employing instance-based parallelism for both rule pruning and evaluation, while using rule-based parallelism for rule selection. We next discuss the rationale and tradeoffs as applied to each phase of the algorithm.

Parallel Pruning

The simplest strategy for pruning the initial rule set is the rule-based approach, but on closer examination it becomes apparent that load-balancing can become a major source of inefficiency. In particular, decision trees may be quite unbalanced, such that certain branches of the tree have a significantly longer path length than others (leading to rules with many more conditions on the LHS that must be considered). A more subtle source of load imbalance arises during the construction of contingency tables that are used to assess the effect of removing a condition from the LHS. For any given training instance, the examination of the current rule may terminate prematurely (before considering all conditions), resulting in potential load imbalance. These problems are avoided by following an instance-based strategy for pruning — all processors participate in the pruning of a single rule by determining the contribution of their block of data to the aforementioned contingency tables. After this is done, a single processor determines which (if any) conditions may be pruned from the rule and the process continues.

Parallel Selection

Selecting the "best" subset of rules from those contained within each target class' ruleset is an optimization problem that is handled by one of several approaches. As noted earlier, this portion of the rule generation algorithm can account for a substantial percentage of total execution time. There are several opportunities for concurrent execution that can be considered.

Parallelization Over Classes Class-based parallelism could be applied to the selection phase (recall that each class ruleset is handled in turn), but this approach has some significant limitations. Most restrictive is that concurrency is limited by k, the number of classes in the training set. Further, it seems likely that individual class rulesets will require varying amounts of CPU time to select a good subset, therefore the class-based approach will almost certainly exhibit poor load-balancing.

Parallelization Over Trials If a class ruleset contains a small (≤ 10) number of rules, exhaustive search is used; for larger collections, C4.5RULES performs eleven trials of hill-climbing, starting with an empty initial rule subset and repeatedly adding or removing rules to improve overall accuracy. Each successive trial expands the size of the subset that represents the starting point for hill-climbing by an additional 10%[1]. Executing each trial in parallel is a possibility although the lower bound for the execution time for this step is then limited by the most time-consuming trial. Additionally, this does not address the problem of handling situations where exhaustive search or simulating annealing are appropriate.

Parallelization Over Rules Regardless of the mechanism used to generate candidate subsets, C4.5RULES evaluates the effectiveness of competing subsets using an approach based on the Minimum Description Length (MDL) principle (Rissanen 1983). The use of MDL with respect to C4.5 rule generation is described in detail in (Quinlan 1993; 1995) — for our purposes, it is sufficient to note that MDL requires an estimate of the combined cost of communicating a *theory* as well as exceptions to that theory (in this context, this translates to the cost of communicating a rule subset and the identity of training instances that are misclassified by the subset).

The cost of communicating the content of each rule within a subset is the number of bits required to transmit the individual conditions in the LHS under some

[1]Earlier releases of C4.5 used simulated annealing to determine an optimal subset of rules, rather than the multiple-trial approach. This is no longer the default, but remains supported through a user-selected option.

(agreed-upon) encoding scheme. Since each rule remains unchanged following the earlier pruning step, the cost associated with each rule need not be recalculated during selection, so the major computational effort of this step is due to identifying the exceptions to the candidate theory, followed by the determination of the total number of bits needed to transmit their identities. These operations are performed sequentially for each rule in the candidate subset, and may be evaluated independently using a rule-based parallel approach; this is the decomposition used in the current implementation, with a slight enhancement: experiments have shown improved load-balancing by using a *dynamic scheduling* scheme, in which the workload is partitioned at a finer granularity than r/p, and blocks of work (rules) are dynamically assigned to processors as they become idle.

Parallel Evaluation

The final step of rule refinement, evaluation of the ordered class rulesets, is simply the "re-classification" of the training set in the context of the current set of rules. Although normally representing a very small percentage of the total execution time of C4.5RULES, if a substantial number of rules are removed during this step and the training set is extremely large, this can involve a substantial amount of CPU time. A simple, effective approach for concurrent execution is instance-based parallelism using a static assignment of training cases to processors.

Empirical Results

To evaluate the performance of the parallel implementation of C4.5RULES, C4.5 (release 8) software was modified for parallel execution and several benchmarks were performed, using four data sets (two real-world, two artificial), as follows:

shuttle-n20 The NASA shuttle data set, described in (Catlett 1991) and available from the UCI repository (Merz & Murphy 1996), contains 43,500 training instances, each with 9 continuous attributes and assigned to one of seven classes. The data set was altered by introducing 20% classification noise as in (Domingos 1996).

sleep A classification task concerning the study of sleep disorders. The training set contains 30,000 examples taken from 38 all-night studies of patients admitted to a sleep disorder clinic. Each instance consists of 13 integer-valued attributes. There are five class labels, corresponding to specific stages of sleep (awake, light/intermediate/deep sleep, or rapid eye movements). The domain is more fully discussed in (Bentrup 1992; Ray *et al.* 1986).

art1 An artificial data set with 10,000 randomly-generated training instances representing the Boolean concept $ab \vee bcd \vee defg$. Following (Cohen 1995), the problem is made more difficult by adding 12 irrelevant attributes and 20% classification noise.

art2 One of ten synthetic benchmark tasks defined in (Agrawal, Imielinski, & Swami 1993). Each instance consists of nine attributes (six continuous, three nominal) and a binary class label. Attribute noise is added to the training set in the form of a "perturbation factor" applied to the continuous attributes in order to model "fuzzy" boundaries. The target concept ("function 5") is determined by ranges on three of the continuous attributes (the remaining six attributes do not appear in the target concept). 50,000 training instances were generated using a perturbation factor of 5%.

Domain	N	P	Time	S	E
shuttle-n20	43500	1	98226		
		2	51095	1.92	0.96
		4	26784	3.67	0.92
		8	14907	6.59	0.82
sleep	30000	1	63545		
		2	32517	1.95	0.98
		4	17985	3.53	0.88
		8	9422	6.74	0.84
art1	10000	1	4535		
		2	2224	2.04	1.02
		4	1139	3.98	1.00
		8	629	7.21	0.90
art2	50000	1	16992		
		2	8791	1.93	0.97
		4	4638	3.66	0.92
		8	2525	6.73	0.84

Table 1: CPU seconds for rule generation in parallel, with speedup and efficiency.

All runs were done on a Silicon Graphics Challenge equipped with eight 200 MHz MIPS R4400 processors and 1 GB memory. Default parameter settings for both C4.5 and C4.5RULES were used.

Table 1 summarizes the results from the benchmark runs for each data set and shows that the parallel implementation provides substantial speedup over serial execution, achieving over 7 times speedup on 8 processors for the **art1** data set. Efficiency remains quite high, exceeding 80% on 8 processors for all benchmarks. The (slight) superlinear speedup obtained for the 2-processor **art1** benchmark is probably due to improved cache utilization (repeated runs of this benchmark duplicate this result, which we consider a pleasant anomaly).

Figure 3 shows the breakdown of CPU time for individual phases of the computation (**sleep** data set) using a linear scale. The equivalent logarithmic plot of Figure 4 shows that all portions of the algorithm that were parallelized scale quite well as additional processors are used. In both figures, it is evident that the

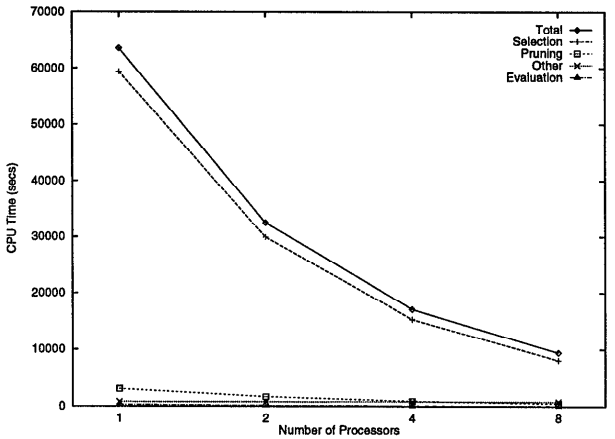

Figure 3: Breakdown of computational phases for the **sleep** data set (linear scale).

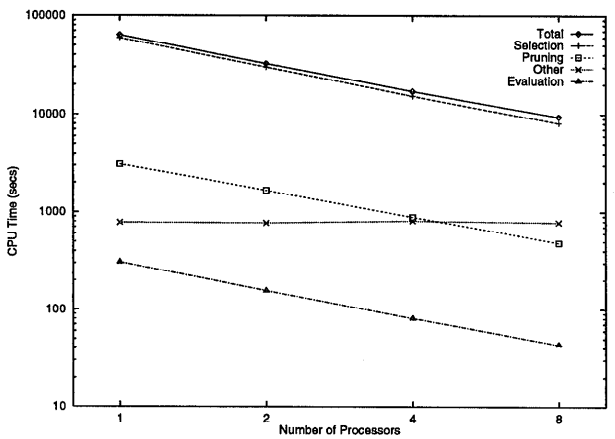

Figure 4: Breakdown of computational phases for the **sleep** data set (log scale).

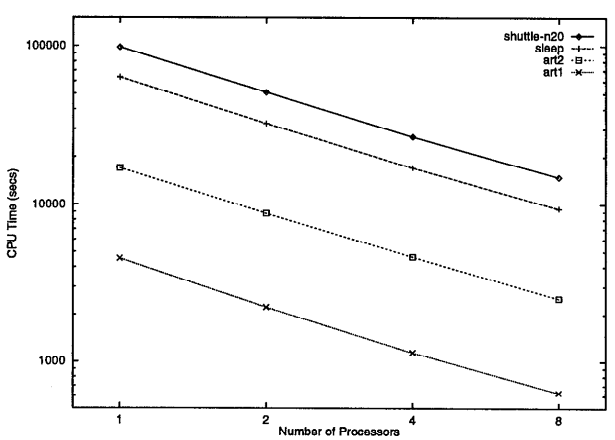

Figure 5: Total execution time for benchmark data sets (log scale).

rule selection phase represents the vast majority of the computational burden — in fact, when more than four processors are used for this problem, the non-parallel portions of the code are exceeded in time requirements only by rule selection.

The execution profiles for all other benchmark runs are similar to that of **sleep**, although the crossover point at which the time for serial portions of the algorithm begins to exceed that required for pruning and evaluation varies. As with **sleep**, the serial portion for the **art1** benchmark exceeds all phases except selection after 4 processors; for the **art2** and **shuttle-n20** benchmarks, pruning continues to require more CPU time than the serial portion up through 8 processors.

Figure 5, which shows the total execution time for all benchmarks, provides another view of the scalability of the parallelized algorithm across problem domains. This figure shows that the performance profile is quite uniform across benchmarks for all processor totals used in these experiments.

Related Work

To the best of our knowledge, no work has been done previously with parallelization of the rule generation component of C4.5, although several researchers have reported excellent results with parallel implementations of other algorithms for rule induction. Parallel variants of the RL learning algorithm have been described for both networks of workstations (Provost & Hennessy 1996) and for massively-parallel SIMD architectures (Provost & Aronis 1996). Strategies for combining the results of multiple instances of learning programs have shown promise for dealing with massive data sets (Chan & Stolfo 1995). A SIMD implementation of the AQ rule learning program provided for parallelization of multiple phases of the algorithm and demonstrated impressive speedup (Cook & Holder 1990). We refer the interested reader to a more complete survey of parallel approaches to rule induction by Provost and Aronis (1996).

Conclusions

We have presented an approach to exploiting opportunities for parallelism in the rule generator component of C4.5. Initial results were shown to be promising, with the overall performance of the parallel version of C4.5RULES demonstrating excellent speedup. Several portions of the algorithm have been identified as computationally-demanding, and we have focused our efforts towards parallelizing these portions. Empirical evaluation of the efficacy of the parallel implementation has shown substantial benefits in general across several problem domains.

We note that large portions of the serial version of the software remain unchanged, and can expect that Amdahl's Law will mandate attention to these portions if further improvements are to be achieved with larger collections of processors than have been used to date;

parallelization of additional portions of the algorithm will be ongoing.

Although we have focused exclusively on aspects of optimization related to parallel processing, it is equally important to note that there are additional strategies that may be employed to gain further increases in speed. For example, there is substantial room for improvement in the form of pure scalar code optimization, not only in C4.5RULES, but also in the companion decision tree generator C4.5. Preliminary experiments with scalar optimizations have shown as much as 15-fold speedup (single-processor) for the problem domains examined here. We are in the process of merging these optimizations with the parallel approaches described and anticipate that a combination of fast scalar code executing on multiple processors will allow for experimentation with much larger databases than previously feasible.

Acknowledgements Many thanks to the anonymous reviewers for numerous helpful comments and suggestions that improved both content and presentation.

References

Agrawal, R.; Imielinski, T.; and Swami, A. 1993. Database mining: A performance perspective. *IEEE Transactions on Knowledge and Data Engineering* 5(6):914–925.

Bentrup, J. 1992. An intelligent assistant for reliable signal classification in a noisy domain. Master's thesis, University of Illinois at Urbana-Champaign.

Breiman, L.; Friedman, J. H.; Olshen, R. A.; and Stone, C. J. 1984. *Classification and Regression Trees.* Belmont, California: Wadsworth International Group.

Catlett, J. 1991. *Megainduction: Machine Learning on Very Large Databases.* Ph.D. Dissertation, University of Sydney.

Chan, P. K., and Stolfo, S. J. 1995. Learning arbiter and combiner trees from partitioned data for scaling machine learning. In Fayyad, U. M., and Uthurusamy, R., eds., *Proceedings, First International Conference on Knowledge Discovery and Data Mining.* Montréal, Québec: AAAI Press.

Cohen, W. W. 1995. Fast effective rule induction. In *Machine Learning: Proceedings of the Twelfth International Conference.* Lake Tahoe, California: Morgan Kaufmann.

Cook, D. J., and Holder, L. B. 1990. Accelerated learning on the Connection Machine. In *Proceedings of the Second IEEE Symposium on Parallel and Distributed Processing,* 448–454.

Domingos, P. 1996. Linear-time rule induction. In *Proceedings, Second International Conference on Knowledge Discovery & Data Mining.* Portland, Oregon: AAAI Press.

Fayyad, U. M.; Piatetsky-Shapiro, G.; and Smyth, P. 1996. From data mining to knowledge discovery: An overview. In Fayyad, U. M.; Piatetsky-Shapiro, G.; Smyth, P.; and Uthurusamy, R., eds., *Advances In Knowledge Discovery and Data Mining.* AAAI Press/The MIT Press.

Fürnkranz, J. 1996. Separate-and-conquer rule learning. Technical Report OEFAI-TR-96-25, Austrian Research Institute for Artificial Intelligence, Vienna, Austria.

Langley, P., and Simon, H. A. 1995. Applications of machine learning and rule induction. *Communications of the ACM* 38(11):55–64.

Merz, C. J., and Murphy, P. M. 1996. UCI repository of machine learning databases. University of California, Department of Computer and Information Science. Irvine, CA. Machine-readable data repository. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

Pagallo, G., and Haussler, D. 1990. Boolean feature discovery in empirical learning. *Machine Learning* 5:71–99.

Provost, F. J., and Aronis, J. M. 1996. Scaling up inductive learning with massive parallelism. *Machine Learning* 23:33–46.

Provost, F. J., and Hennessy, D. N. 1996. Scaling up: Distributed machine learning with cooperation. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence,* volume 1. Portland, Oregon: AAAI Press / The MIT Press.

Quinlan, J. R. 1987. Generating production rules from decision trees. In *Proceedings of the Tenth International Conference on Artificial Intelligence.* Milan, Italy: Morgan Kaufmann.

Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning.* San Mateo, California: Morgan Kaufmann.

Quinlan, J. R. 1995. MDL and categorical theories (continued). In *Proceedings of the Twelfth International Conference on Machine Learning.* Lake Tahoe, California: Morgan Kaufmann.

Ray, S. R.; Lee, W. D.; Morgan, C. D.; and Airth-Kindree, W. 1986. Computer sleep stage scoring - an expert system approach. *International Journal of Biomedical Computing* 19:43–61.

Rissanen, J. 1983. A universal prior for integers and estimation by minimum description length. *Annals of Statistics* 11(2):416–431.