BSPC: a novel bitmap-based scalable parallel classifier

Ghada H. Drahem [a], Mohamed S. Abougabal [a], Hesham Sueyllam [a] and Khaled M. Mahar [b]

[a] *Computer Science and Automatic Control Dept., Faculty of Eng., Alexandria University, Alexandria, Egypt*
[b] *College of Computing and information Technology, ASST, Alexandria, Egypt*

Decision trees have been found very effective for classification especially in data mining. Although classification is a well studied problem, most of the current classification algorithms need an in-memory data structure to achieve efficiency. This limits their suitability for mining over large databases. In this paper, a novel Bitmap-based Scalable Parallel Classifier (BSPC) is presented. It removes all the memory requirements needed by existing algorithms. Also, since scalability is a key requirement for any data mining algorithm, it is considered and achieved in the design of BSPC. Additionally, the suggested algorithm has been designed to be easily parallelized, allowing many processors to work together to build a single consistent model. Performance analysis demonstrates that the BSPC outperforms other state of the art algorithms. The superiority of the novel algorithm is demonstrated through the classification of Wisconsin breast cancer dataset.

لقد وجد أن استخدام شجرة القرار فعال جدا فى التصنيف وخاصة فى التنقيب عن البيانات. وعلى الرغم من أن التصنيف هو مسألة تم دراستها جيدا من قبل، فإن معظم خوارزميات التصنيف الحالية تحتاج إلى بناء بيانات بذاكرة داخلية لتحقيق الفاعلية. وهذا يحد من ملائمتها للتنقيب خلال قواعد البيانات الضخمة. فى هذا البحث نقدم خوارزم تصنيف متوازى جديد ذو مقياس متدرج مع حجم البيانات ومبنى على خريطة البت .وهذه الخريطة تزيل كل متطلبات الذاكرة التى تحتاجها الخوارزميات القائمة . وكذلك، فبما أن إمكانية التدرج هى متطلب أساسى لأى خوارزم للتنقيب عن البيانات، فقد أخذت فى الاعتبار وتحققت فى تصميم الخوارزم الجديد ليكون من السهل جعله متوازى، مع السماح لأكثر من معالج من العمل معا لبناء نموذج متجانس واحد. ويظهر تحليل الأداء أن الخوارزم الجديد يفوق غيره من أحدث الخوارزميات الموجودة حاليا ، وقد ظهر ذلك من خلال تصنيفه لبيانات مركز وسكونسن لسرطان الثدى.

Keywords: Data mining, Classification, Decision tree, Bitmap index, Parallel algorithms, Algorithm complexity analysis

1. Introduction

Data mining refers to the mining or discovery of new information and knowledge in terms of patterns or rules from large amounts of data [1]. Data mining techniques can be classified according to the kinds of knowledge to be discovered. In general, the knowledge can be described as association rules, clustering, or classification. Association rules correlate the presence of a set of items with each other [1]. Clustering is the process of grouping objects into classes, based on their features, by using clustering criteria. The criteria are to maximize (minimize) intraclass (interclass) similarity [2]. Unlike clustering, classification uses class-labeled training data to develop a description or a model for each class. Classification is an important problem in the rapidly emerging field of image mining since a significant part of our knowledge is in the form of images. For example, a large amount of geophysical and environmental data comes from satellite photos and a large amount of the information stored on the Web is in the form of images.

Classification problem can be stated as follows. Given a *training dataset* consisting of records where each record is identified by a unique *record id* and consists of a set of *attribute*s. One of the attributes is the *classifying attribute or class* and the values in its domain are called *class label*s. Classification is the process of discovering a model for the class in terms of the remaining attributes. This model is used to classify future test data for which the class labels are unknown.

Algorithms for classification can be categorized as non-decision tree based methods and decision tree based methods. Non-decision

tree based methods include neural networks [3, 4], genetic algorithms [5], and classification by association rules [3, 6]. Although the neural network model is less sensitive to the training database imbalance, it needs high training time to converge to a satisfied solution. Still another disadvantage is the inability to backtrack the decision making process because the decision is dependent on the initialization of the network and the training set. Genetic algorithms may find a global optimum solution of an optimization problem by means of imitating the type of genetic adaptation that occurs in natural evolution. But, they require a good selection for the statistical parameters which guide the search to fast convergence. The disadvantage of the classification by association rules is the building of the best classifier out of the whole set of rules. It would involve evaluating the possible subsets with the rule consequence that gives the least classification errors. Thus, for m rules there are 2^m subsets which are clearly infeasible for large m.

The decision tree models are found to be most useful in the domain of data mining because they have a number of advantages over other classification methods. First, they yield comparable or better accuracy as compared to other classification models [7]. Second, they can be constructed relatively fast compared to other methods. Finally, tree models are simple and easy to understand [8] and they can be easily converted into SQL statements that can be used to access databases efficiently [9].

A drawback of most existing decision tree classifiers is that they scan the database more than once to build the tree. Another drawback is that they require the entire dataset or special data structure to be memory-resident [10]. In this paper, a novel algorithm is suggested to overcome these drawbacks by creating bitmap indices for all attributes in one pass, and then it uses these bitmap indices in creating the decision tree.

The remainder of this paper is organized as follows. In section 2 a survey of the existing decision tree algorithms is presented. The serial and parallel versions of the proposed algorithm are outlined in section 3. Comparison between the performance analysis of the novel algorithm and the state of the art algorithms is presented in section 4. Experimental results of testing and comparing the classification accuracy of the novel algorithm and some recent algorithms are reported in section 5. Finally conclusions and direction for future work is the subject of section 6.

2. Existing decision tree classifiers

A decision tree is a representation of classification knowledge where each non-leaf (internal) node tests an attribute, its branches correspond to attribute values, and each leaf node assigns a classification flag. The construction of a decision tree requires two steps: tree induction and tree pruning. In the induction step, there are two major issues which differ from an algorithm to another. First, how to find the split points that define node tests. Second, how to use the best split point to partition the data. The tree built in the first phase, tree induction, completely classifies the training data set. This implies that branches are created in the tree even for spurious 'noise' data and statistical fluctuations. These branches can lead to errors when classifying test data. Tree pruning is aimed at removing these branches from the decision tree by selecting the subtree with the least estimated error rate [11].

CART [7], ID3 [8], C4.5 [12] and J4.8(used in Weka) [13] need sorting the continuous attributes at each node, and they require the entire data to fit in the memory. Therefore, they are computationally complex and not suitable for large databases. Thus, will not be compared to Bitmap-based Scalable Parallel Classifier (BSPC).

One idea of modifying tree classifiers, to enable them to classify large datasets, is based on sampling of data at each tree node [8][9]. This method decreases classification time significantly but reduces the classification accuracy. Other idea is the partitioning of the input data and then building a classifier for each partition [14, 15]. The outputs of the multiple classifiers are then combined to get the final classification. The results show that the classification using multiple classifiers never achieves the accuracy of a single classifier that can classify all of the data.

SLIQ [11] does not need sorting the continuous attributes at each node but it sorts them only once at the beginning. In this method, the classification tree is grown in a breadth-first manner and the dataset is not physically split among nodes. Instead, it creates a class list for the class labels attached to all training examples. SLIQ assumes that there is enough memory to keep the class list memory-resident and this limits the size of largest training set.

A Scalable Parallel Classifier for Data Mining (SPRINT) [16] improves the performance of the SLIQ by avoiding the problem of keeping the class list in the memory. But it, in turn, needs a hash table to do the partition. The hash table is repeatedly queried by random access to determine how the entries should be partitioned. If the hash table does not fit in memory (mostly true for large datasets), it will be built in parts so that each part fits and multiple expensive *I/O* passes over the entire dataset may be needed resulting in highly nonlinear performance. SPRINT's design allows it to parallelize the first phase which determines the splitting point effectively. The parallel formulation proposed for the second phase which splits the data is inherently unscalable in both memory requirements and run time [17].

MINing in Databases classifier (MIND), [10] rephrases classification as a classic database problem of summarization and analysis thereof. It leverages SQL (Structured Query Language) by reducing the solution to manipulations of SQL statements embedded in a small program written in C. MIND is similar to SLIQ during the tree induction phase, where it is grown in a breadth-first fashion and the dataset is not physically split among nodes. But instead of using class list, it uses a computed variable and a static array when the tree grows. The value of the variable changes to indicate that the record is moved to a new node by applying a split. Since most modern database servers have strong parallel query processing capabilities, MIND runs in parallel at no extra cost. But since the SQL has not the ability to form multiple inserts into different tables concurrently, the algorithm uses user defined function which is written by C to reduce the *I/O* complexity.

A Scalable and Efficient Parallel Classification Algorithm for Mining Large Datasets, ScalParC, [17] is truly scalable in both runtime and memory requirements. Like SPRINT, ScalParC sorts the continuous attributes only once in the beginning. The key difference is that it employs distributed hash table to implement the splitting phase. The communication structure used to construct and access this hash table introduces a new parallel hashing paradigm. The paradigm gives mechanisms to construct and search a distributed hash table, when many values need to be hashed at the same time. The detailed analysis of applying this paradigm shows that the overall communication overhead does not exceed $O(N)$, and the memory required does not exceed $O(N/p)$ per processor, where N is the number of records and p is the number of processors [17].

Elegant Decision Tree Algorithm (EDTA) [18] aims at improving the performance of the SLIQ algorithm. The improvement has been proposed to reduce the computational complexity associated with the computation of the used splitting criterion (gini index). In EDTA, the gini index is computed not for every successive pair of values of an attribute but over different ranges of attribute values.

3. The novel algorithm

In this paper, a tree induction phase is focused on because it is computationally more expensive than pruning. The induction phase needs the data to be scanned multiple times. But pruning requires one access to the fully-grown decision tree. Therefore, for pruning phase, the algorithm used by SLIQ [11] can be employed.

As mentioned before, a drawback of most existing algorithms is that they scan the database more than once to build the decision tree. Another drawback is that they require all the entire dataset or special data structure to be memory-resident. The proposed BSPC avoids these problems by creating bitmap indices [19] for all attributes in one pass, and then it uses these bitmap indices in creating the decision tree. The use of bitmap indices eliminates the need for the presorting phase, as will be described later in this paper. More-

over, bitmap indices are not required to remain in memory during processing. BSPC uses gini index as splitting criterion to choose the best split for each node because it is argued that the accuracy is sufficient [18]. The proposed algorithm can handle both continuous and categorical attributes. For a continuous attribute, it computes the gini index not every successive pair of values of the attribute but over different ranges of attribute values as in EDTA algorithm. To partition the bitmap indices, it is straightforward to do the process in one pass without building any additional data structure.

For C classes, the value of gini index at node t is calculated as [11, 16, 18]:

$$GINI(t) = 1 - \sum_{j=1}^{C} (p(j/t))^2 , \quad (1)$$

where, $p(j/t)$ is the relative frequency of class j at node t.

Usually, when the gini index is used, the splitting criterion is to minimize the gini index of the split. When a node e is split into k partitions, the quality of the split is computed as:

$$GINI_{split} = \sum_{i=1}^{k} \frac{r_i}{r} GINI(i), \quad (2)$$

where, r is the number of records at node e, and r_i is the number of records at node (partition) i.

In the next subsections, the used data structures and the two major issues (how to find the best split point and how to partition the data) of the proposed algorithm will be described in more details. Also, the required analysis to parallelize the algorithm will be explained.

3.1. Bitmap index

A bitmap index on an attribute consists of one vector of bits (i.e., bitmap) per attribute value. The bitmaps are encoded such that the $(i,j)^{th}$ location is set to 1 if the i^{th} record contains the j^{th} value of the indexed attribute, and the other locations are set to 0. This is called a Value-List index (see fig. 1). Unlike the SLIQ and SPRINT algorithms, which sort the data before processing, BSPC eliminates the sorting step because the bit vector maintains this information implicitly.

As shown in fig. 1, the value-list index is a set of bitmaps, one per attribute value (note that the continuous attribute (age) is partitioned into ranges). In other words, if one views this set as a two-dimensional bit matrix, the focus is on the columns. If the focus moves on the rows, however, then the value-list index can be seen as the list of attribute values or as the list encoded in some particular way. Thus, b bits are needed to represent the distinct actual values of each discrete attribute or the number of ranges of a continuous attribute. In addition, c bits are needed to encode the class labels. Consequently, $\sum_{i=1}^{n} b_i + c$ bits are needed to encode a complete record, where n is the number of attributes. To reduce the total number of bits for every record, the algorithm encodes a new value V from v_i's (the encoded value of the bits of the attribute number i) and b_i's as follow:

$$V = v_1 + v_2 b_1 + v_3 (b_1 b_2) + \ldots + v_{n+1} (b_1 b_2 b_3 \ldots b_n), \quad (3)$$

$$V = v_1 + \sum_{i=1}^{n} \left(v_{i+1} \prod_{j=1}^{i} b_j \right). \quad (4)$$

Fig. 2 shows an example of encoded list index. It considers the class number c_i as the first attribute, i.e. v_1. If the number of attributes is large and can not be encoded as one integer, more than one encoded list is generated; one for each part of the attributes. During the creation of the encoded-list index, the number of records belong to each class and the number of records belong to each class for each value of each attribute are counted. These counts will be used to find the split point of a node.

3.2. Count matrices

With each decision-tree node, that is under consideration for splitting, there is a count matrix for each attribute. Its dimensions represent the class labels and the attribute values. The use of these count matrices is to capture the class distribution of the records

Age	Car type	Insurance risk
23	family	1
17	sports	1
43	sports	1
65	family	0
32	truck	0
20	family	1

(a)

B_{65-57}	B_{-49}	B_{-41}	B_{-33}	B_{-25}	B_{-17}
0	0	0	0	0	1
0	0	0	0	0	1
0	0	1	0	0	0
1	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	1

(b)

B_{truck}	B_{sports}	B_{family}
0	0	1
0	1	0
0	1	0
0	0	1
1	0	0
0	0	1

(c)

Fig. 1. Example of a value-list index. (a) Training set. (b) Value-list index of age continuous attribute. (c) Value-list index of car type categorical attribute.

Age	Car type	Class
23	family	1
17	sports	1
43	sports	1
68	family	0
32	truck	0
20	family	1

(a)

Encoded-list index
1
13
19
10
26
1

(b)

Fig. 2. Example of an encoded-list index. (a) Training set. (b) Encoded–list index.

for each attribute value. These matrices are calculated during the splitting process.

3.3. Tree induction

While growing the tree, the goal at each node is to determine the split point that best divides the training records belonging to that leaf. As stated above, BSPC uses the gini index. The advantage of using this index is that its calculation requires only the distribution of the class labels in each node, which is the content of the count matrices. From all attributes, it selects the attribute-value that gives the lowest gini index.

Once the best split point has been found for a node, the split is executed by creating child nodes and dividing encoded-list index between them. The algorithm scans the encoded list index of the node and decodes each row by using eqs. (5) and (6) to find c_i and v's (the class labels and the attribute values of each attribute, respectively), then applies the split test to determine which rows will be moved to the new encoded-list indices that correspond to the new child nodes .

$$c_i = V \text{ modulo } b_1, \tag{5}$$

where, V is the row value of the bitmap index, b_1 is the number of classes, and

$$v_i = \text{remainder}((V / \prod_{j=1}^{i-1} b_j) / b_i). \tag{6}$$

During the splits it also builds the count matrices for each new leaf, as stated above. These matrices are used to evaluate the split-points in the next pass.

Figs. 3 and 4 show an example for decoding and finding the count matrix for the continuous and the discrete attributes of fig. 2.

3.4. Parallelization of the algorithm

In the literature, three techniques for using multiple processors have been considered [20, 21]. These techniques are the shared memory, the distributed memory and the combined architecture. Also, the types of parallelism were defined as data parallelism and task parallelism. In data parallelism, the database is partitioned among P processors where each processor works on its local partition of the database but performs the same computation.

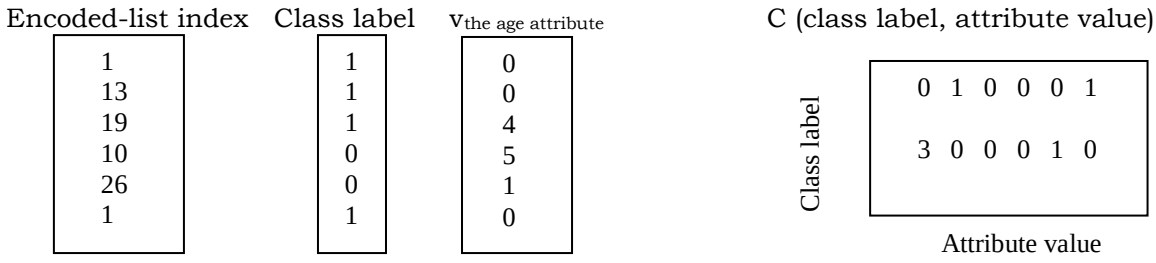

Fig. 3. Decoding and finding the count matrix for the continuous attribute age.

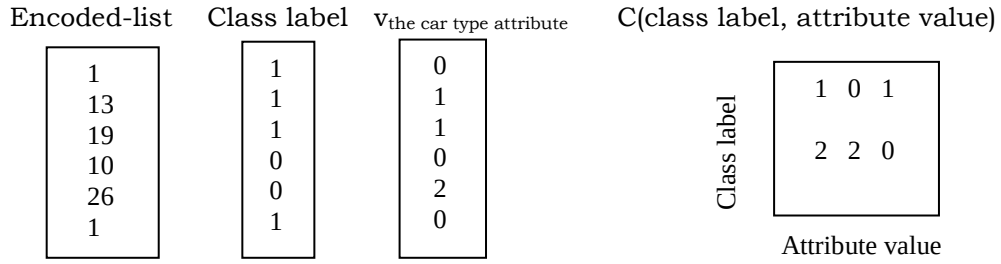

Fig. 4. Decoding and finding the count matrix for the discrete attribute car type.

Task parallelism corresponds to the case where the processors perform different computations independently but have or need access to the entire database.

In shared memory architecture, processors have access to the entire data. But in distributed memory architecture, each processor has its own local memory which only that processor can access directly. However, the process of accessing the database of other processor can involve selective replication or explicit communication of the local portion of that processor.

The design of the BSPC algorithm assumes a shared-nothing parallel environment, a distributed memory architecture. The partition is achieved by first distributing the training-set examples horizontal and equally among all processors and each processor then generates its own encoded-list index.

Finding split points in the parallel algorithm is very similar to that of the serial algorithm. In serial version, the processor scans the encoded list to compute the count matrices. This is not changed in the parallel algorithm. But with getting the full advantage of having N processors, each processor processes independently and simultaneously $1/N$ of the total data. The difference between the serial and parallel versions arises now. Since the count matrices that are built by each processor are based on "local" information only. They must be exchanged to get the "global" counts. This is done by choosing a coordinator to collect the count matrices from all processors and then sums the local matrices to get the global count matrix. As in the serial algorithm, the global matrix is used to find the best split by using the gini index.

Having determined the winning split point, splitting the encoded list index for each leaf is identical to the serial algorithm with each processor responsible for splitting its own encoded list index into two partitions.

4. Performance analysis

There are two important metrics to evaluate the quality of a classifier: classification accuracy and classification time. Regarding the classification accuracy, BSPC has the same accuracy as EDTA, which is better than the other algorithms, because it uses the gini index over different ranges of attribute values and chooses the best split as EDTA does. But for the classification time analysis, the following subsections will cover, in some details, the computations of the I/O complexity and time complexity for both the serial and the parallel versions of BSPC.

4.1. Complexity analysis of the serial version

The BSPC algorithm scans the database once to create the encoded-list index and at each node, it will perform the following operations:
1. scan the encoded list index one by one to find the best split for each leaf node, and
2. partition the encoded list index for each leaf node.

4.1.1. I/O complexity

By using the parameters listed in table 1, and noting that all sizes are measured in bytes, the analysis proceeds as follows. Each row in the encoded-list $r_v=(n \log V + \log C)/8$ and hence $D_k=N/8(n \log V + \log C)$. Since B is the size of the buffer, the algorithm will need to read D_k/B times from disk at each level and this leads to the fact that the *I/O* complexity is $O(LN/B)$.

4.1.2. Computational time complexity

BSPC can be regarded as a two parts algorithm. The first part is the initialization which builds the encoded list and finds the count matrix at the same time. This part consists of a loop which contains some simple operations repeated *N* time. This means that the initialization needs $O(N)$ units of time. The second part is the recursion part where there are the time to find the split point and the time to do the splitting of the input encoded list. Here the count matrix contains all information to compute the gini index and therefore, it does not need to pass on the encoded list. Thus, it only needs one loop to pass on the encoded list in order to separate it into two encoded lists and to find the count matrix. This loop contains some of simple operations and it needs *O* (the number of the rows in the input encoded list, which is *N*) in each depth of the tree. Hence, the algorithm needs *O(LN)* unit time.

Table 1
Parameters used in analysis

B	Size of the disk block
N	# of records in database
n	# of attributes
M	Size of internal memory
C	# of distinct class labels
D_k	The total size of all encoded-list index at depth k
V	# of distinct values for all attributes
r_a	Size of each attribute in database
R	Size of each record in database
r_v	Size of each row in encoded-list index
L	Depth of the final classifier

4.1.3. Comparison to other serial algorithms

4.1.3.1. Comparing the I/O complexity. EDTA and SLIQ are not considered in this comparison because they have the disadvantage of using a class list which is required to be in-memory all the time during processing for efficient performance and this limits the size of largest training set. SPRINT starts by sorting all attribute lists, and then at each node it performs the following operations:
1. scan the attribute lists one by one to find the best split for each leaf node,
2. according to the best split found for each leaf node, form the hash tables and write them to disk,
3. partition the attribute list of the splitting attribute for each leaf node, and
4. partition the attribute lists for the n-1 non-splitting attributes for each leaf node.

Among these operations, the last one incurs the most *I/O* cost. Thus, there are two major parts in SPRINT: the pre-sorting of all attribute lists and the constructing/searching of the corresponding hash tables during partition. It is unrealistic to assume that *N* is small enough to allow hash tables to be stored in memory. Actually, hash tables need to be stored on disk and brought into memory during the partition phase. It is true that hash tables will become smaller at deeper levels and thus fit in memory, but at the upper levels they are very large. A careful analysis shows that the estimation for the *I/O* complexity of SPRINT is $O(nN^2 \log N / BM)$ [10].

MIND needs to read the data set once at each level. Each record in the DETAIL table (the working data structure of MIND) has n attribute values of size r_a, plus class label that may take one byte. Thus, the record size R is equal to $nr_a + 1$. Hence, $R = O(N)$ and the *I/O* complexity of MIND is $O(LnN/B)$[10].

4.1.3.2. Comparing the computational time. In the initialization phase SPRINT needs to create the attribute lists and sort them. This means that it needs $O(nN \log N)$ as a minimum time to sort all of them. In the recursion part, in

order to decide which attribute is to be split at each node, the gini indices have to be computed for each successive pair of values for all attributes which have not been classified yet. This needs $O(nN)$ times at the first depth and $O((n-1)N)$ times in the second depth and so on. As a result, this sums to $O(n/(n+1)N/2) \cong O(n^2N)$ unit time . Also, the time for splitting is more than that of BSPC because SPRINT must split each attribute list. The winning attribute in the SPRINT needs $O(N)$ unit time to be determined, but the other attributes need search on the hash table for each value of the attribute and this needs $O((n-1)NlogN) \cong O(nNlogN)$ unit time. Consequently, SPRINT needs $O(nNlogN)$ *plus* $O(n^2N)$ unit time.

For the computational time of MIND, if we ignore the implicit sorting embedded in some of the used SQL statements used in its implementation, it has almost the same time complexity as the novel algorithm.

4.2. Complexity analysis of the parallel version

The parallel runtime consists of computational time and the parallelization overhead. If T_s is the serial runtime of the algorithm and T_p is the parallel runtime on a p processor, the parallelization overhead is given by $T_o = pT_p - T_s$. For runtime scalability, the overhead, T_o should not exceed $O(T_s)$ [22]; i.e. the parallelization overhead per processor should not exceed $O(T_s/p)$. For the classification problem at hand the serial runtime is $T_s = O(N)$ for the majority of tree levels. BSPC is designed such that none of the components of the overall communication overhead of the classification process exceeds $O(N)$ at any level; i.e. the processor communication overhead does not exceed $O(N/p)$ per level so it is scalable.

It is easy to show that the total work of BSPC algorithm $W = 2*pO(N)+O(N)$ and the efficiency is $O(N)/(2p*O(N)+O(N)) = 1/(2p+1)$.

4.2.1. Comparison with SPRINT and ScalParC

In the parallel formulation of SPRINT, the hash table is required for each processor to split its local copies of all the attribute lists as in the sequential algorithm. Since each processor has to receive the entire hash table, the amount of communication overhead per processor is proportional to the size of the hash table, which is $O(N)$. Hence, this approach is not scalable in runtime. Also, it is not scalable in terms of memory requirements, because the hash table size in each processor is $O(N)$ for top node as well as for nodes at the upper levels of the tree.

ScalParC treats these problems by using the scalable parallel hashing paradigm. Thus, it is scalable but the parallel hashing paradigm adds a computation overhead to the algorithm. Also, ScalParC still presorts the attribute lists and calculates the gini index for every continuous attribute value like SPRINT. *Thus, the above analysis demonstrates that BSPC is the fastest algorithm.*

5. Experimental results

The classification accuracy of BSPC algorithm was test using Wisconsin breast cancer dataset and compared with the accuracy of SLIQ algorithm, EDTA algorithm and the Neural Network technique. 250 patterns with 9 attributes were considered for training as in [18] to compare between the algorithms. Note that the number of patterns is small because EDTA and SLIQ algorithms require that the class list be in-memory all the time for efficient performance. This limits the size of largest training set, and hence the induced decision tree. The attributes are as follows:

# *Attribute*	*Domain*
1. Clump thickness	1-10
2. Uniformity of cell size	1-10
3. Uniformity of cell shape	1-10
4. Marginal adhesion	1-10
5. Single epithelial cell size	1-10
6. Bare nuclei	1-10
7. Bland chromatin	1-10
8. Normal nucleoli	1-10
9. Mitoses	1-10

There are two classes namely Benign and Malignant denoted by 0 and 1. The induced decision tree by BSPC algorithm is shown in fig. 5. The first value in the elliptical boxes denotes the attribute number and the second value denotes the splitting value of the attrib-

ute. For a 20 patterns test data, the classification accuracy using BSPC algorithm is as follows:

Target classes: 00000000001111111111
Output classes: 00000000001111111111

which is 100% classification accuracy, while the classification accuracies of original SLIQ algorithm, EDTA algorithm, and the Back propagation algorithm are 75%, 100%, and 90% respectively [18].

6. Conclusions and future extension

The proposed algorithm solves the problem of classification by using bitmap indices and reduces the *I/O* complexity significantly. The performance measurements show that the algorithm demonstrates scalability with respect to the number of examples in training sets and the number of parallel processors. The algorithm is fast because it scans the database once to build the decision tree and for continuous attribute we compute the gini index not every successive pair of values of the attribute but over different ranges of attribute values. BSPC algorithm reduces the *I/O* complexity because it does not do the presort phase required by most existing algorithms and bitmap indices are not required to be in memory during processing. In other words, BSPC does not require all the entire dataset or special data structure to be memory-resident. It uses gini index as splitting criterion to choose the best split for each node where it is

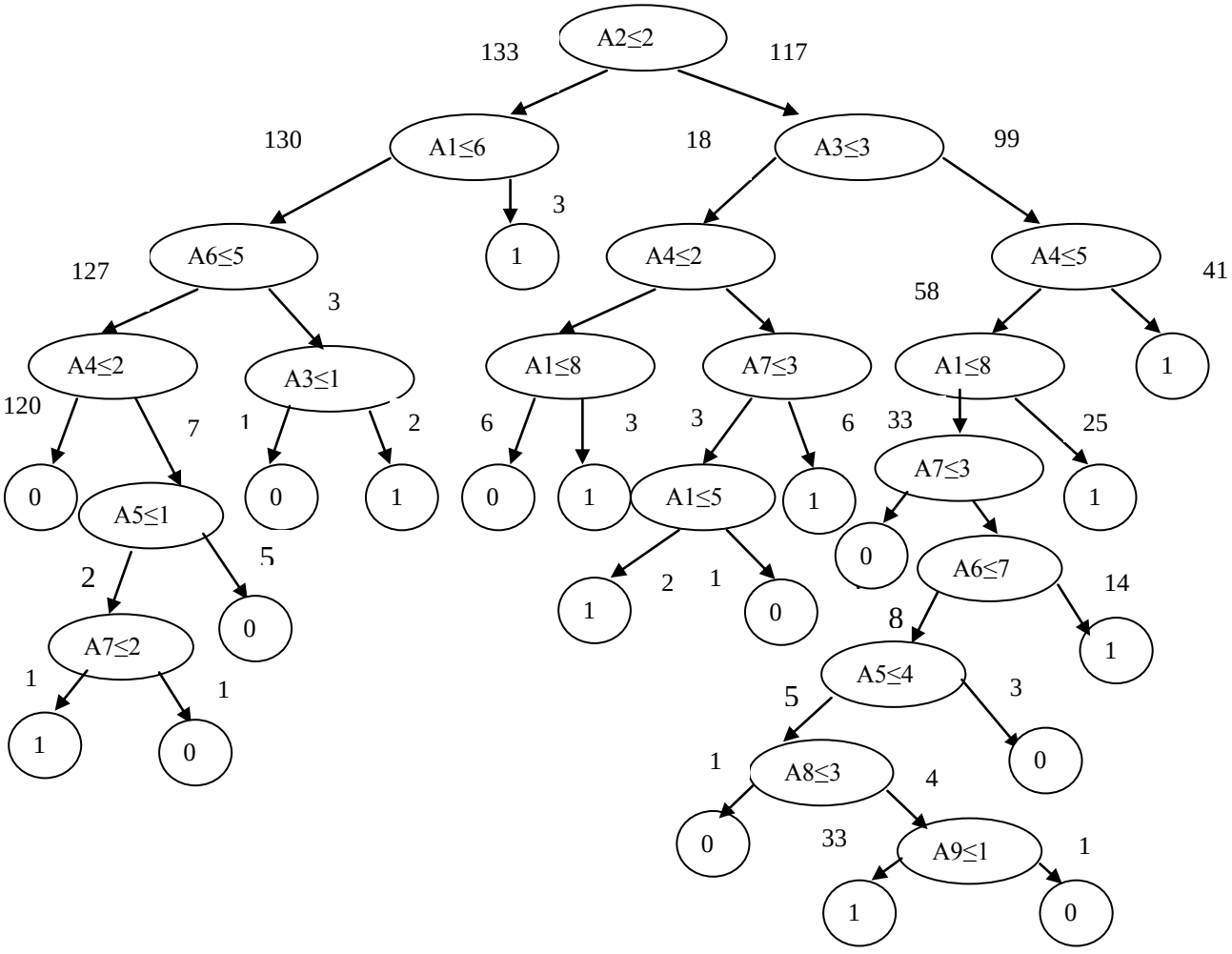

Fig. 5. Decision tree using BSPC algorithm (breast cancer data).

argued that the accuracy is sufficient. The algorithm has also been designed to be easily parallelized, allowing many processors to work together to build a single consistent model.

As for future work, there are two possible extensions. The first is the addition of a computed column to the bitmap index to show the node that each record belongs to. This idea was implemented in SLIQ[11] and MIND [10], but here the purpose is to study the runtime behavior with this change. The second possible extension is to include a pruning phase, which is expected to remove small disjuncts (rules covering small number of examples, noise). This is due to the belief that it is better to capture generalizations than specializations in the training set. Genetic programming shall be applied during that pruning phase in order to derive new rules from the set of removed small disjuncts.

Acknowledgments

We would like to express our gratitude for the referees patience and care in reviewing the paper. We very much appreciate their valuable and helpful comments.

References

[1] R. Elmasri and S. Navathe, Fundamental of Database Systems, Addison-Wesley (2000).

[2] J. Han, Data Mining Techniques, Conference Tutorial, http://db.cs.sfu.ca/DBminer (1996).

[3] M. Antonie, O.R. Zaiane "Application of Data Mining Techniques for Medical Image," Proceedings of ACM SIGKDD Conference, San Francisco, USA, pp. 95-101, Aug (2001).

[4] D. Michie, D.J. Spiegelhalter and C.C. Taylor, Machine Learning, Neural and Statistical Classification, Ellis Horwood (1994).

[5] B.D. Turney, "Cost-sensitive Classification: Empirical Evaluation of a Hybrid Genetic Decision Tree Induction Algorithm," Journal of Artificial Intelligence Research, Vol. 2, pp. 369-409 (1995).

[6] B. Liu, W. Hsu, Y. Ma, "Integrating Classification and Association Rule Mining," Proceedings of 4th International Conference KD. and DM. New York, USA, pp. 80-86 (1998).

[7] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone, Classification and Regression Trees, Wadsworth, Belmont (1984).

[8] J.R. Quuinlan, "Induction of Decision Trees," Machine Learning, Vol. 1, pp. 81-106 (1986).

[9] J. Wirth and J. Catlett, "Experiments on the Costs and Benefits of Windowing in ID3," Proceedings of 5th International Conference on Machine Learning, Ann Arbor, Michigan, USA, pp. 87-99 (1988).

[10] M. Wang, Approximation and Learning Techniques in Database Systems, Ph.D. Thesis, Duke University (1999).

[11] M. Mehta, R. Agarwal and J. Rissanen, "SLIQ: A Fast Scalable Classifier for Data Mining," In Proceedings of 5th International Conference on Extending Database Technology (EDBT), Avignon, France, pp. 18-32, March (1996).

[12] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann, San Mateo, CA, (1993).

[13] The university of Waikato, New Zealand, http://www.cs.waikato.ac.nz/~ml/index.html

[14] P.K. Chan and S.J. Stolfo. "Meta-Learning for Multistrategy and Parallel Learning," In Proceedings of 2nd International Workshop on Multistrategy Learning, Center of AI, George Mason University, USA, pp. 150-165 (1993).

[15] P.K. Chan and S.J. Stolfo. "Experiments on Multistrategy Learning by Meta-Learning," In Proceedings of 2nd International Conference on Information and Knowledge Management, Washington DC, USA, pp. 314-323 (1993).

[16] J. Shafer, R. Agarwal and M. Mehta, "SPRINT: A Scalable Parallel Classifier for Data Mining," In Proceedings of 22th International Conference on Very Large Databases, Mumbai, India, pp. 544-555 (1996).

[17] M.V. Joshi, G. Karypis, and V. Kumar. "ScalParC: A New Scalable and efficient

parallel Classification Algorithm for Mining Large Datasets," 11th international Parallel Processing Symposium, Orlando, USA, pp. 573-579 (1998).

[18] B. Chandra, S. Mazumder, N. Parimi, "Elegant Decision Tree Algorithm for Classification in Data Mining," 3rd International Conference on Web Information Systems Engineering, pp. 160-165 (2002)

[19] C.C. Yannis, E. Ioannidis, "Bitmap Index Design and Evaluation," In Proceedings of ACMSIGMOD International Conference on Management of Data, Seattle, WA., pp. 355-366 (1998).

[20] A. Freitas and S. Lavington, Mining Very Large Databases with Parallel Processing, Kluwer Academic Publishers (1998).

[21] M. Joshi, G. Karypis and V. Kumar, "Parallel Algorithms for Sequential Associations: Issues and Challenges," Symposium Talk at 9th SIAM International Conference on Parallel Processing, San Antonio, pp. 99-103 (1999).

[22] V. Kumar, A. Grama, A. Gupta and G.Karypis, Introduction to Parallel Computing: Algorithm Design and Analysis, Addison Wesley, Redwood City, CA (1994).

Received October 30, 2004
Accepted July 13, 2005