Single MCMC Chain Parallelisation on Decision Trees

Efthyvoulos Drousiotis[1] and Paul G. Spirakis[2,3]

[1] Department of Electrical Engineering and Electronics, University of
E.Drousiotis@liverpool.ac.uk
[2] Department of Computer Science, University of Liverpool, Liverpool L69 3BX, UK
spirakis@liverpool.ac.uk
[3] Department of Computer Engineering and Informatics, University of Patras, 26504
Patras, Greece

Abstract. Decision trees are highly famous in machine learning and usually acquire state-of-the-art performance. Despite that, well-known variants like CART, ID3, random forest, and boosted trees miss a probabilistic version that encodes prior assumptions about tree structures and shares statistical strength between node parameters. Existing work on Bayesian decision trees depend on Markov Chain Monte Carlo (MCMC), which can be computationally slow, especially on high dimensional data and expensive proposals. In this study, we propose a method to parallelise a single MCMC decision tree chain that enables us to reduce its run-time through multi-core processing while the results are statistically identical to conventional sequential implementation. We also calculate the theoretical and practical reduction in run time, which can be obtained utilising our method on multi-processor architectures. Experiments showed that we could achieve 18 times faster running time provided that the serial and the parallel implementation are statistically identical.

Keywords: Parallel algorithms, Machine Learning, MCMC Decision Tree

1 Introduction

In Bayesian statistics, it is a common problem to collect and compute random samples from a probability distribution. Markov Chain Monte Carlo (MCMC) is an intensive technique commonly used to address this problem when direct sampling is often arduous or impossible. MCMC using Bayesian inference is often used to solve problems in biology [13], forensics [12], education [5], and chemistry [6], among other areas making it one of the most widely used algorithms when a collection of samples from a probability distribution is needed. Monte Carlo applications are generally considered embarrassingly parallel since each chain can run independently on two or more independent machines or cores. Despite that, the main problem is that each chain is not embarrassingly parallel, and when the feature space and the proposal are computationally expensive, we can not do much to improve the running time and get results faster. When we have to handle huge state-spaces and complex compound states, it takes significant time for an MCMC simulation to converge on an adequate model not only in terms of the number of iterations required but also the complexity of the calculations occurring in each iteration(such as searching for the best features and tree shape of a decision tree). For example, running an MCMC on a single chain Decision tree for a dataset of 400000 datapoints and 15 features took upwards of 6 hours to converge when run on a $2.3 - 5.10GHz$ Intel Core i7-10875H. In [3], an approach aiming to parallelise a single chain is presented, and the improvement achieved is at its best 2.2 times faster. The functionality of this kind of solution is therefore limited as in real-time, and life applications run time is critical. The work presented in this paper aims to find methods to significantly reduce the MCMC Decision tree's runtime by emphasising on the implementation of MCMC rather than the statistical algorithm itself. The remainder of this paper is organised as follows. Section 2 explains the MCMC in General and the Most Recent Work. Section 3 presents the MCMC in Decision trees. Our method is outlined in section 4, with the possible theoretical improvements. We introduce the case study in which we applied our method and reviewed results in section 5. Section 6 concludes the paper.

2 Markov Chain Monte Carlo in General and Most Recent Work

One of the most widely used algorithms is the Metropolis [9] and its generalisation (see algorithm1), the Metropolis-Hastings sampler (MH) [8]. Given a partition of the state vector into components, i.e., $x = (x_1, ..., x_k)$, and that we wish to update the i_th component, the Metropolis-Hastings update proceeds as follows. We first have a density for producing candidate observation x', such that $x'_i = x_i$, which is denoted by $q(x, x')$. Given the chains ergodic condition, the definition of q is arbitrary, and it has a stationary distribution π which is selected so that the observations may be generated relatively easily. After the new state generation $x' = (x_1, ..., x_{i-1}, x_i, x_{i+1}, ..., x_k)$ from density $q(x, x')$, the new state is accepted or rejected using the Rejections Sampling principle with acceptance probability $\alpha(x, x')$ given by equation 1. If the proposed state is rejected, the chain remains in the current state.

It is worth mentioning that acceptance probability in this form is not unique, considering there are many acceptance functions that supplies a chain with the required properties. Nevertheless, Peskun(1973) [10] proved that MH is the optimal one where the proper states are rejected least often, which maximises the statistical efficiency meaning that more samples are collected with fewer iterations.

$$a(\chi, \chi') = \min(1, \frac{\pi(x')}{\pi(x')} \frac{q(\chi|\chi')}{q(\chi'|\chi)}) \tag{1}$$

On a Markov process, the next step depends on the current state, which makes it hard for a single Markov chain to be processed contemporaneously by several processing elements. Byrd [2]. proposed a method to parallelise a single Markov chain(Multithreading on SMP Architectures), where we consider backup move “B” in a separate thread of execution as it is not possible to determine whether move “A” will be accepted. If “A” is accepted, the backup move ‘B’ - whether accepted or rejected - must be discarded as it was based upon a now supplanted chain state. If “A” is rejected, control will pass to “B”, saving much of the real-time spent considering “A” had “A” and “B” been evaluated sequentially. Of course, we may have as many concurrent threads as desired.

At this point, it is worth mentioning that the single chain parallelisation can become quickly problematic as the efficiency of the parallelisation is not guaranteed, especially for computationally cheap proposal distributions. Also, we need to consider that nowadays, computers make serial computations much faster than in 2008, when the single parallelisable chain was proposed.

Another way of making faster MCMC applications is to reduce the convergence rate by requiring fewer iterations. Metropolis-Coupled MCMC($(MC)^3$) utilised multiple MCMC [1] chains to run at the same time, while one chain is treated as the "cold" where its parameters are set to normal while the other chains are treated as "hot", which are expected to accept the proposed moves. The space will be explored faster through the "hot" chains than the "cold" as they are more possible to make disadvantageous transitions and not to remain at near-optimal solutions. The speedup increased when more chains and cores were added.

Our work is focused on achieving a faster execution time of the MCMC algorithm on Decision trees through multiprocessor architectures. We aim to reduce the number of iterations while the number of samples collected is not affected. Multi-threading on SMP Architectures and $(MC)^3$ differs from our work as the former targets rejected moves as a place for optimisation, and the latter requires communication between the chains. Moreover, the aims are different as $(MC)^3$ expands the combination of the chain, enhancing the possibilities of discovering different solutions and assisting avoid the simulation getting stuck in local optima.

2.1 Probabilistic trees packages and level of parallelism

Most of the existing probabilistic tree packages are only supported by the R programming language.

BART [4] software included in the CRAN package [4] supports multi threading based on OpenMP, where there are numerous exceptions for operating systems, so it is difficult to generalise. Generally, Microsoft Windows lacks OpenMP detection since the GNU autotools do not natively exist on this platform and Apple macOS since the standard Xcode toolkit is also not provided. The parallel package provides multi-threading via forking, only available in Unix. BART under CRAN uses parallelisation for the predict function and running concurrent chains.

[4] https://cran.r-project.org/web/packages/BART/index.html

BartMachine [5], which is written in Java and its interface is provided by rJava package, which requires Java Development Kit(JDK), provides multi-threading features similar to BART. BartMachine is recommended only for those users who have a firm grounding in the java language and its tools to upgrade the package and get the best performance out of it. Similar to BART, its parallelisation is based on running concurrent chains.

The rest of the available packages, BayesTree[6],dbarts[7],Bartpy[8],XBART[9] and imptree[10] does not support any kind of parallelisation.

Concurrent chains can not solve the problem of long hours of execution time. For example, if a single chain needs 50 hours to execute, 5 chains will still need 50 hours if run concurrently. In contrast, in our case, a chain that serially needs 50 hours now takes approximately 2 hours for each chain. Moreover, we can to run concurrent chains where each chain is parallelised. If our implementation is compared to a package like BartMachine and BART, the runtime improvement we achieved is around 18 times faster, and if we compare it with a package that does not offer any parallelisation like most of the existing ones, the run time improvement for 5 chains is around 85 times faster.

3 Markov Chain Monte Carlo in Decision Tree

A decision tree typically starts with a root node, which branches into possible outcomes. Each of those outcomes leads to additional decision nodes, which branch off into other possibilities ending up in leaf nodes. This gives it a treelike shape.

Our model describes the conditional distribution of y given x, where x is a vector of predictors $[x = (x_1, x_2, ..., x_p)]$. The main components of the $tree(T)$ includes the depth of the tree$(d(T))$, the features$(k(T))$ and the thresholds$(c(T))$ for each node where $k(T), c(T) \in \theta$, and the possibilities $p(Y|T, \theta, x)$ for each leaf node$(L(T))$. If x lies in the region corresponding to the i_th terminal node, then $y|x$ has distribution $f(y|\theta_i)$, where f represents a parametric family indexed by θ_i. The model is called a probabilistic classification tree, according to the quantitative response y.

As Decision Trees are identified by (θ, T), a Bayesian analysis of the problem proceeds by specifying a prior probability distribution $p(\theta, T)$. Because θ indexes the parametric model for each T, it will usually be convenient to use the relationship

$$p(Y_1 :_N, T, \theta | x_1 :_N) = p(Y|T, \theta, x)p(\theta|T)p(T) \tag{2}$$

[5] https://cran.r-project.org/web/packages/bartMachine/index.html
[6] https://cran.r-project.org/web/packages/BayesTree/BayesTree.pdf
[7] https://cran.r-project.org/web/packages/dbarts/index.html
[8] https://pypi.org/project/bartpy/
[9] https://jingyuhe.com/xbart.html
[10] https://cran.r-project.org/web/packages/imptree/index.html

In our case it is possible to analytically obtain eq 2 and calculate the posterior of T as follows:

$$p(Y|T,\theta,x) = \prod_{i=1}^{N} p(Y_i|x_i,T,\theta) \tag{3}$$

$$p(\theta|T) = \prod_{j\in(T)} p(\theta_j|T) = \prod_{j\in(T)} p(k_j|T)p(c_j|k_j,T) \tag{4}$$

$$p(T) = \frac{a}{(1+d)^{\beta}} \tag{5}$$

Equation 3 describes the product of the probabilities of every data point(Y_i) classified correctly given the datapoints features(x_i), the tree structure(T), and the features/thresholds(θ) on each node on the tree. Equation 4 describes the product of possibilities of picking the specific feature(k) and threshold(c) on every node given the tree structure(T). Equation 5 is used as the prior for tree T_i. This formula is recommended by [4] and three aspects specify it: the probability that a node at depth $d(=0.1.2....)$ is nonterminal, the parameter $a \in 0,1$ which controls how likely a node would split, with larger α values increasing the probability of split, and the parameter $\beta > 0$ which controls the number of terminal nodes, with larger values of β reducing the number of terminal nodes. This feature is crucial as this is the penalizing feature of our probabilistic tree which prevents it from overfitting and allowing convergence to the target function $f(X)$ [11], and it puts higher probability on "bushy" trees, those whose terminal nodes do not vary too much in depth.

An exhaustive evaluation of equation 2 over all trees T will not be feasible, except in trivially small problems, because of the sheer number of possible trees, which makes it nearly impossible to determine precisely which trees have the largest posterior probability.

Despite these limitations, Metropolis-Hastings algorithms can still be used to explore the posterior. Such algorithms simulate a Markov chain sequence of trees such as:

$$T_0, T_1, T_2, ..., T_n \tag{6}$$

which are converging in distribution to the posterior $p(Y|T,\theta,x)p(\theta|T)p(T)$ in equtaion 2. Because such a simulated sequence will tend to gravitate toward regions of higher posterior probability, the simulation can be used to search for high-posterior probability trees stochastically. We next describe the details of such algorithms and their implementation.

3.1 Specification of the Metropolis-Hastings Search Algorithm on Decision Trees

The Metropolis-Hastings(MH) algorithm for simulating the Markov chain in Decision trees (see equation 7) is defined as follows. Starting with an initial tree T_0, iteratively simulate the transitions from T_i to $T_i + 1$ by these two steps:

1. Generate a candidate value T' with probability distribution $q(T_i, T')$.
2. Set $T_{i+1} = T'$ with probability

$$a(T_i, T') = \min(1, \frac{\pi(Y_1 :_N, T', \theta' | x_1 :_N)}{\pi(Y_1 :_N, T, \theta | x_1 :_N)} \frac{q(T, \theta | T', \theta')}{q(T', \theta' | T, \theta)}) \tag{7}$$

Otherwise set $T_{i+1} = T_i$.

To implement the algorithm, we need to specify the transition kernel q. We consider kernels $q(T, T')$, which generate T' from T by randomly choosing among four steps:

- Grow(G) : add a new $D(T)$ and choose uniformly a $k(T)$ and a $c(T)$
- Prune(P) : choose uniformly a $D(T)$ to become a leaf
- Change(C) = choose uniformly a $D(T)$ and change randomly a $k(T)$ and a $c(T)$
- Swap(S) = choose uniformly two $D(T)$ and swap their $k(T)$ and $c(T)$

The rules are chosen by picking a number uniformly between 0 and 1 and each action have its own interval. For example, $p(G) = 0.3, p(P) = 0.3, p(C) = 0.2, p(S) = 0.2, [0, 0.3, 0.6, 0.8, 1]$

The probabilities (see equation 8) represent the sum of the probabilities of every accepted forward move. P(G), p(P), p(C), p(S) are set by the user who chooses how often each move wants to be proposed.

$$q(T', \theta' | T, \theta) = q(T' | T) q(\theta' | T') = \sum_a q(a) q(T' | T, a) q(\theta' | T', \theta, a) \tag{8}$$

where :

$$q(G) q(T' | T, G) q(\theta' | T', \theta, G) = p(G) \times \frac{1}{c} \times \frac{1}{k} \times \frac{1}{|L(T)|} \tag{9}$$

$$q(P) q(T' | T, P) q(\theta' | T', \theta, P) = p(P) \times \frac{1}{|D(T)| - 1} \tag{10}$$

$$q(C) q(T' | T, C) q(\theta' | T', \theta, c) = p(C) \times \frac{1}{|D(T)|} \times \frac{1}{c} \times \frac{1}{k} \tag{11}$$

$$q(S) q(T' | T, S) q(\theta' | T', \theta, S) = p(S) \times \frac{1}{(|D(T)|(|D(T)| - 1))/2} \tag{12}$$

Equation 9 can be described as the possibility of proposing the grow move including the probability of choosing the specific feature(k), threshold(c) and leaf node($|L(T)|$) to grow. P(G) is multiplied by the number of features(k), the unique number of datapoints(c) and the number of leaf nodes($|L(T)|$). For example, given a dataset with 100 unique datapoints(c), 5 features(k), a tree structure(T) with 7 leaf nodes($|L(T)|$) and a $p(G) = 0.3$ eq9 will be $0.3 \times \frac{1}{100} \times \frac{1}{5} \times \frac{1}{7}$

Equation 10 is the possibility of proposing the prune move, where p(P) is multiplied by the number of decision nodes subtracting one($(|D(T)| - 1)$ we are not allowed to prune the root node). In practise, given a $p(P) = 0.3$ and a tree structure(T) with 7 decision nodes($|D(T)|$) eq10 will be $0.3 \times \frac{1}{10-1}$

Equation 11 is the possibility of proposing the change move where p(C) is multiplied by the number of decision nodes($|D(T)|$), the number of features(k), and the number of unique datapoints(c). For example, given a dataset with with 100 unique datapoints(c), 5 features(k), a tree structure(T) with 12 decision nodes($|D(T)|$) and a $p(G) = 0.2$ eq9 will be $0.2 \times \frac{1}{100} \times \frac{1}{5} \times \frac{1}{12}$

Equation 12 is the possibility of proposing the swap move where p(S) is multiplied by the number of paired decision nodes($|D(T)|$).In practise, given a tree structure(T) with 12 decision nodes($|D(T)|$) and a $p(S) = 0.2$ eq12 will be $0.2 \times \frac{1}{((12)(12-1))/2}$

Theorem 1. *Transition kernel(see equation 13) yields a reversible Markov chain, as every step from T to T' has a counterpart that can move from T' to T.*

$$q(T', \theta' | T, \theta) \tag{13}$$

Proof. Assume a Markov chain, starting from its unique invariant distribution π. Now, take into consideration that for every sample $T_0, T_1, ..., T_n$ have the same joint probability mass function(p.m.f) as their time reversal $T_n, T_{n-1}, ..., T_0$, so as we can call the Markov chain reversible, as well as its invariant distribution π is reversible. This can be explained as a simulation of a reversible chain that looks the same if it runs backward.

The first thing we have to look for is if the Markov chain starts at π, and it can be checked by equation14

$$\begin{aligned}
&P(T_k = i | T_{k+1} = j, T_{k+2} = i_{k+2}, ..., T_n = i_n) \\
&= \frac{P(T_k = i, T_{k+1} = j, T_{k+2} = i_{k+2}, ..., T_n = i_n)}{P(T_{k+1} = j, T_{k+2} = i_{k+2}, ..., T_n = i_n)} \\
&= \frac{\pi P_{ij} P j i_{k+2} .... P_{i_{n-1} i_n}}{\pi P j i_{k+2} .... P_{i_{n-1} i_n}} \\
&= \frac{\pi P_{ij}}{\pi_j}
\end{aligned} \tag{14}$$

Equation14 is only dependent on i and j where this expression for reversibility must be the same as the forward transition probability $P(X = T_{k+1} = i | X = T_k = j) = P_{ji}$. If, both original and the reverse Markov chains have the same transition probabilities, then their p.f.m must be the same as well.

An example for our probabilistic tree is the following:

Assume a tree structure(T) with 5 leaf nodes($|L(T)|$) and 11 decision nodes($|D(T)|$) sampling from a given dataset with 4 features(c) and 100 unique datapoints(k) for each feature.

If for example the forward proposal($q(T', \theta'|T, \theta)$) = ("*change*") with p(C) = 0.2, we end up with the following equation: $0.2 \times \frac{1}{11} \times \frac{1}{4} \times \frac{1}{100}$. At the same time the reverse proposal(going from the current position to the previous) ($q(T, \theta|T', \theta')$) equation looks exactly the same as the forward proposal. Given the above practical example we have strengthen our proof which shows that ($q(T', \theta'|T, \theta)$) = ($q(T, \theta|T', \theta')$), which shows in practise the reverse transition kernel nature of our model.

Algorithm 1 The General Metropolis-Hashting Algorithm

```
Initialize $X_0$
for $i = 1$ to $N$ do
    sample $\chi'$ from $q(\chi'|\chi_{t-1})$
    Calculate $\alpha(\chi_i, \chi_{t-1}) = \min(1, \frac{\pi(x')}{\pi(x')} \frac{q(\chi|\chi')}{q(\chi'|\chi)})$
    Draw $u$ from $u[0,1]$
    if $u < \alpha(\chi_i|\chi_{t-1})$ then
        $\chi_i = \chi'$
    else
        $\chi_i = \chi_{t-1}$
    end if
end for
```

4 Parallelising a single Decision Tree MCMC Chain

Given a Decision's Tree MCMC chain with N iterations, we propose a method that utilises C number of cores aiming to enhance the running time of a single chain by at least an order of magnitude. As stated in Section1 and Section3, at each iteration, a new sample χ' is drawn from the proposal distribution. Our method requires sampling from C number of cores, $S(C = S)$ number of samples in parallel. We then accept the sample with the greatest $a(T_i, T')$ and repeat the same method until the Markov Chain converges to a stationary distribution. In our method, we check the Markov chain convergence when the F1-score fluctuates less than $\pm 3\%$ for at least 100 iterations. Once the Chain has converged, we proceed to the second phase of our method. We now keep producing samples using C cores, but we can now collect more than one sample which satisfies $a(T_i, T') >= u$. (u is a random uniform number$[0, 1]$) From this point, we will propose new samples from the sample with the greatest $a(T_i, T')$ until we are happy with the number of samples collected. Using this method, we can collect the same number of samples and explore the feature space as effectively as the serial implementation, but 18 times faster.

Our algorithm reduces the number of iterations and explores the feature space faster as we use more cores. This provides us with a significant run time improvement up to 18 times faster when the feature space is big and the proposal is expensive. The following sections will evaluate the running time improvement and the quality of the samples produced.

Algorithm 2 Single Chain parallelisation on MCMC Decision Trees

```
Initialize $T_0$
for $i = 1$ to $N$ do
    sample $C$ number of $T'$ from $Q({T^j}'|{T^j}_{t-1})$
    Calculate in parallel $\alpha({T^j}_i|{T^j}_{t-1}) = \min(1, \frac{p(Y_{1:N}, T', \theta'|x_{1:N})}{p(Y_{1:N}, T, \theta|x_{1:}} \frac{q(T, \theta|T', \theta')}{q(T', \theta'|T, \theta)})$ for each
sample
    Store every sampled posterior $\alpha({T^j}_t|{T^j}_{t-1})$ value
    Until converge $T' = \max \alpha({T^j}_i|{T^j}_{t-1})$
    if Markov Chain converged then
        Draw $u$ from $uniform[0, 1]$
        for $j = 1$ to $j$ do                                ▷ For loop run in parallel
            if $\alpha({T^j}_i|{T^j}_{t-1}) > u$ then
                Collect Sample
                $T' = \max \alpha({T^j}_i|{T^j}_{t-1})$
            end if
        end for
    end if
end for
```

Theoretical gains Using C cores simultaneously, the programme cycle consists of repeated "steps," each performing the equivalent of between 1 and n iterations. We need to calculate the number of iterations based on the acceptance rate to produce the same number of samples(S) when we increase C. The moves are considered in parallel, where they are accepted or rejected. Given that the average probability of a single arbitrary move being rejected is p_r, the probability of the i_{th} in every single concurrent core is pr. This step continues for i iterations where equations 15, 16, and 17 show the iterations needed, run time, and speedup improvement, respectively, given a time(t) in minutes for each iteration. Theoretical Run time(see figure2) and speedup (see figure3) were plotted for varying cores.

$$i = \frac{S/C}{pr} \tag{15}$$

$$Runtime = \frac{i}{t} \tag{16}$$

$$Speedup = \frac{Runtime}{C} \tag{17}$$

For example, given a single MCMC Decision Tree chain running for 10000 iterations and an acceptance rate of 70%, after 3000 burn-in iterations, we end up with 4900 samples.

For the parallel MCMC chain, given the same settings as the serial one, with a 30% burn-in period and 25 cores, we will collect the same number of samples with 500 iterations. This provides us with a 25 times faster execution time. Algorithm 2 indicates the part implemented on a parallel environment, and figure 3 the maximum theoretical benefits from utilising our method. Considering the communication overhead and the parts of the algorithm that are not parallelised, in practise the speedups of this order are not achievable. Therefore, we will test it in practise and find out how it performs on real-life scenarios respect to the accuracy and the runtime improvement

5 Results

5.1 Quality of the samples between serial and parallel implementation

We have used the Wine dataset from scikit learn datasets[11] repository as well as Pima Indians Diabetes and Dermatology from UCI machine learning repository [12], which are publicly available, to examine the quality of the samples on several testing hypotheses, including the different number of cores per iteration given the average F1-Score(see figure 1). Precision and Recall were also calculated for more depth and detailed insights about the performance and quality of the samples. The results, including the F1-Score, Precision, Recall, and Accuracy(see table2, table3 and table4) produced through 25-Fold Cross-Validation, ensure that every observation from the original dataset has the possibility of appearing in the training and test sets and also reduce any statistical error. Before any performance comparison, we need to examine whether the samples produced for each test case(25 cores and 40 cores) have any statistical difference from the serial implementation. We next examine if extracted samples by utilising 25 and 40 cores are representative of the family of the data they come. We use as ground truth data the F1-scores on each sample collected on every fold of the serial implementation, and we compare them with the corresponding collected samples from the other two test cases. In order to check any statistical difference between the samples, we performed the two-sample t-test for unpaired data[7], which is defined as follows:

$$T = \frac{Y_1 - Y_2}{\sqrt{\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2}}} \tag{18}$$

$$|T| > t_{1-a/2,\nu} where : \nu = \frac{\left(\dfrac{s_1^2}{n_1} + \dfrac{s_2^2}{n_2}\right)^2}{\dfrac{(s_1^2/n_1)^2}{n_1 - 1} + \dfrac{(s_2^2/n_2)^2}{n_2 - 1}} \tag{19}$$

[11] https://scikit-learn.org/stable/datasets/toy$_{d}ataset.html$
[12] https://archive.ics.uci.edu/ml/index.php

Formula18 is used to calculate the t-Test statistic equation where N_1 and N_2 are the sample sizes, Y_1 and Y_2 are the sample means, and s_1^2 and s_2^2 are the sample variances. The null hypothesis is rejected when equation 19 holds, which is the critical value of the t distribution with ν degrees of freedom.

For our first dataset(Wine), we first examine the serial implementation with the parallel using 25 cores. The absolute value of the t-Test, 0.60, is less than the critical value of 1.964, so we prove the null hypothesis and conclude that the samples drawn by using 25 cores have not any statistical difference at the 0.05 significance level. We then compare the serial implementation with the parallel using 40 cores. In this case, the absolute value of the t-Test, 27.73, is greater than the critical value of 1.964, so we reject the null hypothesis and conclude that the samples drawn by using 40 cores have a statistical difference at the 0.05 significance level. We also examine the parallel implementations between them(using 25 and 40 cores accordingly). In this case, the absolute value of t-Test, 53.54, is greater than the critical value of 1.964, so we reject the null hypothesis and conclude that the samples drawn using 40 cores(in comparison with the samples drawn with 25 cores) have a statistical difference at the 0.05 significance level. The results for the rest of the datasets are presented explicitly in table 1

Datasets	1 vs 25 cores	1 vs 40 cores	25 vs 40 cores	Critical T value
Pima Indians Diabetes	0.51	36.15	32.31	1.97
Dermatology	1.95	16.41	32.31	1.97
Wine	0.60	27.72	53.54	1.97

Table 1. Datasets Critical Values

T-test proves that when we sample with 25 cores, it is less possible to end up with samples that are not statistically the same as the serial implementation. Table 1, table 2, and table 3 shows that when we sample in parallel using less than 25 cores, the accuracy and the F1-score remain on the same levels as the serial implementation. On the other hand, when we sample with more than 25 cores (see graph 1), results and the t-Test(see Table 1) indicate that the quality of the samples is not as accurate as required, affecting the F1-Score and the overall quality of the samples. Our recommendation is to scale up to 25 cores the algorithm we propose, whereas the user can choose a greater number of cores with a trade-off between improved time and less accurate samples and results.

Tables 1, 2, 3, 4 and graph 1 indicate that a single chain on MCMC Decision Trees can not be an embarrassingly parallel algorithm as we can only improve the running time of a single chain by utilising a specific number of cores. The running time improvement we achieved($\times 18$ faster) is the maximum run-time enhancement we can achieve on an MCMC decision tree to maintain the high metrics produced by the serial implementation. If we try to extract more than 25 samples per iteration, it is highly probable to get samples that affect the final results negatively. According to our results, the maximum number of cores can that be used is 25. Furthermore, Precision, Recall, and F1-score metrics indicate that no overfit is observed even when more than 3 labels exist, proving the samples' quality even in multi class classification problems.

Labels	Precision			Recall			F1-score		
	1 core	20cores	40cores	1 core	20cores	40cores	1 core	20cores	40cores
0	0.85	0.88	0.90	1.00	1.00	0.83	0.92	0.94	0.86
1	1.00	1.00	0.67	0.78	0.78	0.89	0.88	0.88	0.76
2	1.00	0.93	1.00	1.00	1.00	0.69	1.00	0.96	0.82
accuracy							**0.93**	**0.93**	**0.81**

Table 2. Results for Wine Dataset

Labels	Precision			Recall			F1-score		
	1 core	20cores	40cores	1 core	20cores	40cores	1 core	20cores	40cores
0	0.83	0.84	0.92	0.86	0.83	0.68	0.84	0.84	0.78
1	0.65	0.63	0.54	0.61	0.65	0.86	0.63	0.64	0.66
accuracy							**0.78**	**0.78**	**0.73**

Table 3. Results for Pima Indian Diabetes Datasets

Labels	Precision			Recall			F1-score		
	1 core	20cores	40cores	1 core	20cores	40cores	1 core	20cores	40cores
0	0.93	0.93	0.93	1.00	1.00	1.00	0.97	0.97	0.93
1	0.94	0.94	0.77	1.00	1.00	1.00	0.97	0.97	0.77
2	1.00	1.00	1.00	0.96	0.96	0.96	0.98	0.98	1.00
3	0.94	0.94	0.92	0.94	0.94	0.75	0.94	0.94	0.92
4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
5	1.00	1.00	1.00	0.67	0.67	0.50	0.80	0.80	1.00
accuracy							**0.96**	**0.96**	**0.93**

Table 4. Results for Dermatology Datasets

5.2 Practical gains

Figure 2 demonstrates our proposed method's theoretical and practical run time improvement regarding how many hours are needed to collect a specific number of samples. Figure 3 presents the theoretical and practical speedup achieved given the number of cores used. Both figures 2, and 3 show a remarkable runtime improvement, especially when the feature space is ample and the proposal

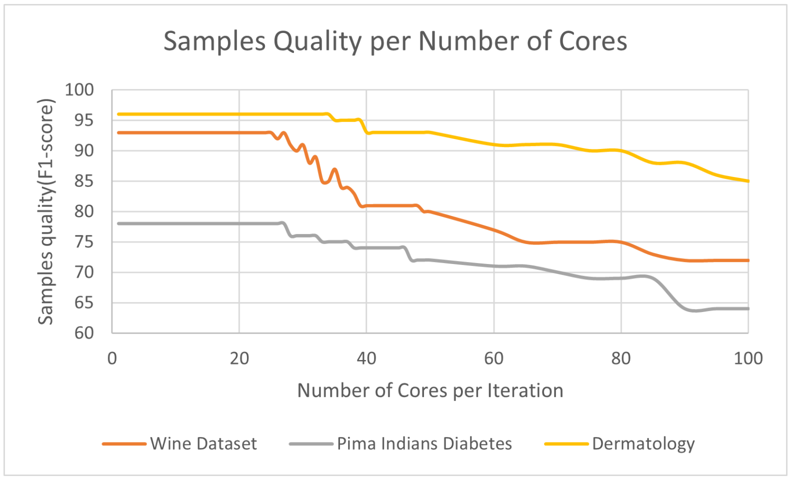

Fig. 1. Quality of Samples Based of Number of Cores per Iteration

is expensive. Figure 3 demonstrates that in practise, theoretical speedups of this order can not be achieved for various reasons, including communications overhead, and as well as the cores do not receive constant utilisation. The practical improvement achieved used the novel method we proposed, speeds up the process up to 18 times depending on the number of cores the user may choose to utilise. To the best of our knowledge it is the first time where a single chain in general, specifically on decision trees, is parallelised with our proposed method.

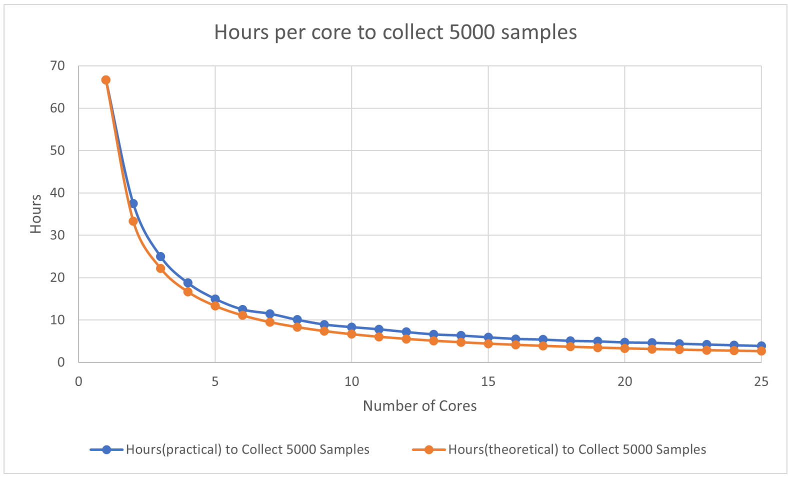

Fig. 2. Total Hours to Collect 5000 Samples Given a Dataset with 500 000 Datapoints

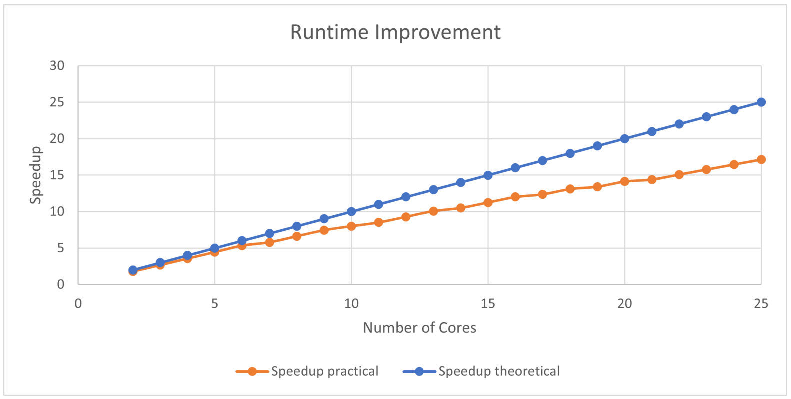

Fig. 3. Speedup achieved by utilising different number of cores

6 Conclusion

Our novel proposed method for parallelising a single MCMC Decision tree chain takes advantage of multicore machines without altering any properties(when run on less than 25 cores) of the Markov Chain. Moreover, our method can be easily and safely used in conjunction with other parallelisation strategies, i.e., where each parallel chain can be processed on a separate machine, each being sped up using our method.

Furthermore, our approach can be applied to any MCMC Decision tree algorithm which needs to process hundreds of thousands of data given an expensive proposal where an execution time of 18 times faster can be easily achieved. As multicore technology improves, CPUs with multiple processing cores will provide speed-ups closer to the theoretical limit calculated. By taking advantage of the improvements in modern processor designs our method can help make the use of MCMC Decision tree-based solutions more productive and increasingly applicable to a broader range of applications. Future work includes exploring the SMC(Sequential Monte Carlo) family of algorithms on Decision trees, focusing on optimal L-kernel.

References

1. Gautam Altekar, Sandhya Dwarkadas, John P Huelsenbeck, and Fredrik Ronquist. Parallel metropolis coupled markov chain monte carlo for bayesian phylogenetic inference. *Bioinformatics*, 2004.
2. Jonathan MR Byrd, Stephen A Jarvis, and Abhir H Bhalerao. Reducing the run-time of mcmc programs by multithreading on smp architectures. In *2008 IEEE International Symposium on Parallel and Distributed Processing*. IEEE, 2008.
3. Jonathan MR Byrd, Stephen A Jarvis, and Abhir H Bhalerao. Speculative moves: multithreading markov chain monte carlo programs. *High-Performance Medical Image Computing and Computer Aided Intervention (HP-MICCAI)*, 2008.
4. Hugh A Chipman, Edward I George, and Robert E McCulloch. Bart: Bayesian additive regression trees. *The Annals of Applied Statistics*, 2010.
5. Efthyvoulos Drousiotis, Lei Shi, and Simon Maskell. Early predictor for student success based on behavioural and demographical indicators. In *International Conference on Intelligent Tutoring Systems*. Springer, 2021.
6. Joffrey Dumont Le Brazidec, Marc Bocquet, Olivier Saunier, and Yelva Roustan. Quantification of uncertainties in the assessment of an atmospheric release source applied to the autumn 2017. *Atmospheric Chemistry and Physics*, 2021.
7. Lloyd Fisher and John Mcdonald. 3—two-sample t-test. *Fixed effects analysis of variance. Probability and mathematical statistics: a series of monographs and textbooks*, 1978.
8. W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. 1970.
9. Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 1953.
10. Peter H Peskun. Optimum monte-carlo sampling using markov chains. *Biometrika*, 60(3):607–612, 1973.

11. Veronika Ročková and Enakshi Saha. On theory for bart. In *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, 2019.
12. Duncan Taylor, Jo-Anne Bright, and John Buckleton. Interpreting forensic dna profiling evidence without specifying the number of contributors. *Forensic Science International: Genetics*, 2014.
13. Gloria I Valderrama-Bahamóndez and Holger Fröhlich. Mcmc techniques for parameter estimation of ode based models in systems biology. *Frontiers in Applied Mathematics and Statistics*, 2019.