Bayesian Decision Trees Inspired from Evolutionary Algorithms

Efthyvoulos Drousiotis[1], Alexander M, Phillips[1], Paul G, Spirakis[2], and Simon Maskell[1]

[1] Department of Electrical Engineering and Electronics, University of Liverpool, Liverpool L69 3GJ, UK; {E.Drousiotis, A.M.Philips, S.Maskell}@liverpool.ac.uk

[2] Department of Computer Science, University of Liverpool, Liverpool L69 3BX, UK P.Spirakis@liverpool.ac.uk

Abstract. Bayesian Decision Trees (DTs) are generally considered a more advanced and accurate model than a regular Decision Tree (DT) as they can handle complex and uncertain data. Existing work on Bayesian DTs uses Markov Chain Monte Carlo (MCMC) with an accept-reject mechanism and sample using naive proposals to proceed to the next iteration. This method can be slow because of the burn-in time needed. We can reduce the burn-in period by proposing a more sophisticated way of sampling or by designing a different numerical Bayesian approach. In this paper, we propose a replacement of the MCMC with an inherently parallel algorithm, the Sequential Monte Carlo (SMC), and a more effective sampling strategy inspired by the Evolutionary Algorithms (EA). Experiments show that SMC combined with the EA can produce more accurate results compared to MCMC in 100 times fewer iterations.

Keywords: Swarm Particle Optimisation · Bayesian Decision Trees · Machine Learning

1 Introduction and Relevant Work

Obtaining and calculating random samples from a probability distribution is challenging in Bayesian statistics. Markov Chain Monte Carlo (MCMC) is a widely used method to tackle this issue. MCMC can characterise a distribution without knowing its analytic form by using a series of samples and it has been used to solve problems in various domains, including psychology [9], forensics [24], education [10, 11], and chemistry [15], among other areas. Monte Carlo applications are generally considered embarrassingly parallel since each chain can run independently on two or more independent machines or cores. However, this method is not very effective. Even though you increase the number of chains and decrease the number of samples produced from each chain, the burn-in process for each chain would remain unchanged. Nevertheless, the principal problem is that processing within each chain is not embarrassingly parallel. When the feature space and the proposal are computationally expensive, we can only do a little to improve the running time.

Much research has been done to improve the efficiency of MCMC methods. The improvements can be grouped into several categories [20]. One of these categories is based on understanding the geometry of the target density function. An example of this is Hamiltonian Monte Carlo [14] (HMC), which uses an auxiliary variable called momentum and updates it using the gradient of the density function. Different methods, such as symplectic integrators of various precision levels, have been developed to approximate the Hamiltonian equation [3]. HMC tends to generate less correlated samples than the Metropolis-Hastings algorithm, but it requires the gradient of the density function to be available and computationally feasible.

Another approach involves dividing complex problems into smaller and more manageable components. For example, as discussed earlier, using multiple parallel MCMC chains to explore the parameter space and then combining the samples obtained from these chains [19]. However, this approach does not achieve faster convergence of the chains to the stationary distribution. This is because all chains have to converge independently of each other in MCMC. It is also possible to partition the data space, which is already implemented in the context of Bayesian DTs [13], or partition the parameter space [1] into simpler pieces that can process independently, which are proven not to be much effective.

MCMC DTs simulations can take a long to converge on a good model when the state space is large and complex. This is due to both the number of iterations needed and the complexity of the calculations performed in each iteration, such as finding the best features and structure of a DT. MCMC, in general, and as will be explained detailed in section 2, generates samples from a specific distribution by proposing new samples and deciding whether to accept them based on the evaluation of the posterior distribution. Because the current sample determines the next step of the MCMC, it cannot be easy to process a single MCMC chain simultaneously using multiple processing elements. A method [12] is proposed to parallelise a single chain on MCMC Decision trees but the speedup is not always guaranteed.

Another method of speeding up MCMC focuses on enhancing the proposal function, which is the approach we pursue in this paper. This can be achieved through techniques such as simulated tempering [18], adaptive MCMC [8], or multi-proposal MCMC [17]. Having a good proposal function in Markov Chain Monte Carlo (MCMC) and, in general, Monte Carlo methods is crucial for the efficiency and accuracy of the algorithm. Poor proposals can lead to slow convergence, poor mixing, and biased estimates. Good proposal functions can efficiently explore the target distribution and reduce the correlation between successive samples.

Several papers describe novelties specifically on Bayesian DTs, focusing on different improvements. For example, [12, 13, 23], contributed towards the runtime enhancement, [27] explored a new novel proposal move called "radical restructure," which changes the structure of the tree (T), without changing the number of leaves or the partition of observations into leaves. Moreover, [4] proposed different criteria for accepting or rejecting the T on MCMC, such as posterior probability,

marginal likelihood, residual sum of squares, and misclassification rates. The general approach to improving the proposal function in Monte Carlo methods has been introduced previously. However, in the context of Bayesian DTs, there is still enough space for exploration.

Our contribution is then:

– We describe for the first time a novel algorithm inspired by the Evolutionary Algorithms to improve the proposal function of the Bayesian DTs that uses an inherently parallel algorithm, a Sequential Monte Carlo (SMC) sampler, to generate samples.

2 Bayesian Decision Trees

A DT operates by descending a tree T. The process of outputting a classification probability for a given datum starts at a root node (see Figure 1). At each non-leaf node, a decision as to which child node to progress to is made based on the datum and the parameters of the node. This process continues until a leaf node is reached. At the leaf node, a node-specific and datum-independent classification output is generated.

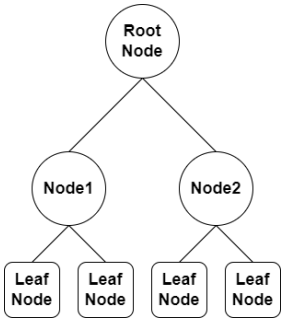

Fig. 1. Decision Tree

Our model describes the conditional distribution of a value for Y given the corresponding values for x, where x is a vector of predictors and Y the corresponding values that we predict. We define the tree to be T such that the function of the non-leaf nodes is to (implicitly) identify a region, A, of values for x for which $p\left(Y|x \in A\right)$ can be approximated as independent of the specific value of $x \in A$, i.e. as $p(Y|x) \approx p(Y|\phi_j, x \in A)$. This model is called a probabilistic classification tree, according to the quantitative response Y.

For a given tree, T, we define its depth to be $d(T)$, the set of leaf nodes to be $L(T)$ and the set of non-leaf nodes to be $\bar{L}(T)$. The T is then parameterised by: the set of features for all non-leaf nodes, $k_{\bar{L}(T)}$; the vector of corresponding thresholds, $c_{\bar{L}(T)}$; the parameters, $\phi_{L(T)}$, of the conditional probabilities associated

with the leaf nodes. This is such that the parameters of the T are $\theta(T) = [k_{\bar{L}(T)}, c_{\bar{L}(T)}, \phi_{L(T)}]$ and $\theta(T)_j$ are the parameters associated with the jth node of the T, where:

$$\theta(T)_j = \begin{cases} [k_j, c_j] & j \in \bar{L}(T) \\ \phi_j & j \in L(T). \end{cases} \tag{1}$$

Given a dataset comprising N data, $Y_{1:N}$ and corresponding features, $x_{1:N}$, and since DTs are specified by T and $\theta(T)$, a Bayesian analysis of the problem proceeds by specifying a prior probability distribution, $p(\theta(T), T)$ and associated likelihood, $p(Y_{1:N}|T, \theta(T), x_{1:N})$. Because $\theta(T)$ defines the parametric model for T, it will usually be convenient to adopt the following structure for the joint probability distribution of N data, $Y_{1:N}$, and the N corresponding vectors of predictors, $x_{1:N}$:

$$p(Y_{1:N}, T, \theta(T)|x_{1:N}) = p(Y_{1:N}|T, \theta(T), x_{1:N})p(\theta(T), T) \tag{2}$$

$$= p(Y_{1:N}|T, \theta(T), x_{1:N})p(\theta(T)|T)p(T) \tag{3}$$

which we note is proportional to the posterior, $p(T, \theta(T)|Y_{1:N}, x_{1:N})$, and where we assume

$$p(Y_{1:N}|T, \theta(T), x_{1:N}) = \prod_{i=1}^{N} p(Y_i|x_i, T, \theta(T)) \tag{4}$$

$$p(\theta(T)|T) = \prod_{j \in T} p(\theta(T)_j|T) \tag{5}$$

$$= \prod_{j \in T} p(k_j|T)p(c_j|k_j, T) \tag{6}$$

$$p(T) = \frac{a}{(1 + d(T))^{\beta}} \tag{7}$$

Equation 4 describes the product of the probabilities of every data point, Y_i, being classified correctly given the datum's features, x_i, the T structure, and the features/thresholds, $\theta(T)$, associated with each node of the T. At the jth node, equation 6 describes the product of possibilities of picking the k_jth feature and corresponding threshold, c_j, given the T structure. Equation 7 is used as the prior for the T. This prior is recommended by [5] and three parameters specify this prior: the depth of the T, $d(T)$; the parameter, a, which acts as a normalisation constant; the parameter, $\beta > 0$, which specifies how many leaf nodes are probable, with larger values of β reducing the expected number of leaf nodes. β is crucial as this is the penalizing feature of our probabilistic T which prevents an algorithm that uses this prior from over-fitting and allows convergence to occur [21]. Changing β allows us to change the prior probability associated with "bushy" trees, those whose leaf nodes do not vary too much in depth.

An exhaustive evaluation of equation 2 over all trees will not be feasible, except in trivially small problems, because of the sheer number of possible trees.

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

$$T_0, T_1, T_2, ..., T_n \tag{8}$$

which converge in distribution to the posterior, which is itself proportional to the joint distribution, $p(Y_{1:N}|T, \theta(T), x_{1:N})p(\theta(T)|T)p(T)$, specified in equation 2. We choose to have a simulated sequence that gravitates toward regions of the higher posterior probability. Such a simulation can be used to search for high-posterior probability trees stochastically. We now describe the details of algorithms that achieve this and how they can be implemented.

2.1 Stochastic Processes on Trees

To design algorithms that can search the space of trees stochastically, we first need to define a stochastic process for moving between trees. More specifically, we consider the following four kinds of move from one T to another:

- Grow(G) : we sample one of the leaf nodes, $j \in L(T)$, and replace it with a new node with parameters, k_j and a c_j, which we sample uniformly from their parameter ranges.
- Prune(P) : we sample the jth node (uniformly) and make it a leaf.
- Change(C) : we sample the jth node (uniformly) from the non-leaf nodes, $\bar{L}(T)$, and sample k_j and a c_j uniformly from their parameter ranges.
- Swap(S) : we sample the j_1th and j_2th nodes uniformly, where $j_1 \neq j_2$, and swap k_{j_1} with k_{j_2} and c_{j_1} with c_{j_2}.

We note that there will be situations (e.g. pruning from a T with a single node) when some moves cannot occur. We assume each 'valid' move is equally likely, which makes it possible to compute the probability of transition from one T, to another, T', which we denote $q\left(T', \theta(T')|T, \theta(T)\right)$.

3 Our Approach on Evolutionary Algorithms

Evolutionary Algorithms (EA) mimic living organisms' behavior, using natural mechanisms to solve problems [2]. In our approach, the optimisation problem is represented as a multi-dimensional space on which the population lives (in our case, the population is the total number of trees). Each location on the solution space corresponds to a feasible solution to the optimisation problem. The optimal solution is found by identifying the optimal location in the solution space.

Pheromones play a crucial role in evolutionary algorithms as they are used for communication among the population [7, 16]. When a member of the population moves, it receives pheromones from the other member of the population and uses them to determine its next move. Once all members of the population reach new locations, they release pheromones; the concentration and type of pheromones released depend on the objective function or fitness value at that location. The

solution space is the medium for transmitting pheromones, allowing individuals to receive and be affected by the pheromones released by other individuals, creating a global information-sharing model.

The population will gradually gain a rough understanding of global information through their movements, which can significantly benefit the optimisation process. In our approach, the EA can use the solution space as a memory to record the best and worst solutions produced in each iteration. Once the positioning stage is finished, all pheromones on the solution space are cleared. To guide the optimisation process, the most representative extreme solutions are selected from the recorded solutions, and the corresponding locations are updated with permanent pheromones. Unlike permanent pheromones, temporary pheromones only affect the movements of trees in the next iteration.

4 Methods

4.1 Conventional MCMC

One approach is to use a conventional application of Markov Chain Monte Carlo to DTs, as found in [12].

More specifically, we begin with a tree, T_0 and then at the ith iteration, we propose a new T' by sampling $T' \sim q\left(T', \theta(T')|T_i, \theta(T_i)\right)$. We then accept the proposed T' by drawing $u \sim U([0,1])$ such that:

$$T_{i+1} = \begin{cases} T' \ u \leq \alpha(T'|T) \\ T_i \ u > \alpha(T'|T) \end{cases} \tag{9}$$

where we define the acceptance ratio, $\alpha(T', T)$ as:

$$\alpha(T'|T) = \frac{p(Y_{1:N}|T, \theta(T), x_{1:N})}{p(Y_{1:N}|T', \theta(T'), x_{1:N})} \frac{q\left(T, \theta(T)|T', \theta(T')\right)}{q\left(T', \theta(T')|T, \theta(T)\right)} \tag{10}$$

.

This process proceeds for n iterations. We take the opportunity to highlight that this process is inherently sequential in its nature.

4.2 Evolutionary Algorithm in Bayesian Decision Trees

Initializing population The initial population plays a crucial role in the solutions' quality. An initial population with a good mix of diversity and a substantial number of trees can help improve the optimisation performance of the algorithm. To create a diverse initial population, a random method is often employed. This approach helps to ensure that the algorithm can perform a global search efficiently, is easy to use, and has a diverse set of individuals in the initial population.

The population size of trees is invariable, denoted as n. The location of every T_i in the D - dimensional space can be described as : $T = (T_1^1, T_2^2, \ldots, T_n^d, \ldots, T_N^D)$. According to the value of our objective function $p(Y1:N, T, \theta(T)|x1:N)$, the

fitness value of location d and T_i can be measured. By comparing the current location of each T_i in the initial population, the optimal location and the worst location in the initial population were obtained, and the value of the objective function of the optimal location in the initial population was recorded.

Positioning stage In the positioning stage, in our use case, trees release permanent and temporary pheromones. The solution space records the locations of the terrible solution and the excellent solution produced by each iteration. While all trees move to the new location, the pheromones are updated differently, which will be discussed in section 4.2. The process in the positioning stage is shown in Figure 2. In our case, the possible movements of the T_i are those described in section 2.1. When the proposed move is the *Grow*, we search for possible solutions in a higher dimensional space; when the proposed move is the *Prune*, we explore for possible solutions in lower dimensional space; and when the proposed moves are *Change* and *Swap*, we search for solutions on the current dimensional space.

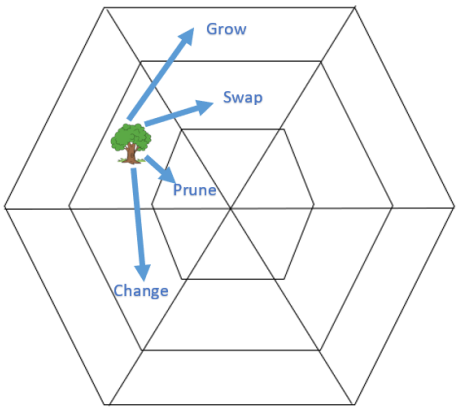

Fig. 2. Positioning Stage

Permanent Pheromones *Permanent pheromones* have persistent effects. On each iteration, we evaluate each T_i to compute the value α_n (see equation 10). If α_n is greater than a uniform number between $[0, 1]$, we store the T_i and the stochastic move associated with *positive exploration* and $effective\ moves$, respectively. If α_n is less than a uniform number between $[0, 1]$ we store the T_i and the stochastic move associated, on *negative explartion* and $innefective\ moves$ respectively (*negative explartion* and $innefective\ moves$ will be used on *temporary pheromones*). We repeat the procedure above for all

trees. We then calculate the *permanent pheromones* given the $effective\ moves$. *Permanent pheromones* is a single list with 4 real numbers, representing the possibilities of each one of the 4 stochastic moves to be selected on the next iteration. We update the *permanent pheromones* by adding to each possibility on the *permanent pheromones*, the number of times each move is in the $effective\ moves$ list, and we then divide each element of the list by the sum of the $effective\ moves$ (we normalise to add up to 1). We have also designed a mechanism to avoid having a biased *permanent pheromones* list. For example, in the first stages of the algorithm, it is common the *Grow* move to be a more useful proposal compared to the *Prune*. In such cases, if a specific move has more than 80% possibilities to be chosen, we re-weight the list by setting the dominant move having 40% possibilities to be selected and the rest having 20% possibilities to be chosen on the next iteration.

Temporary Pheromones *Temporary pheromones* can only affect the movements of trees in the next iteration. We discussed above how we end up with a list called $effective\ moves$ and a list called $ineffective\ moves$. When the iteration ends, *temporary pheromones* clear, and each T_i will release new *temporary pheromones*, which will be recorded. Algorithm 1 shows how we produce and store pheromones.

4.3 Sequential Monte Carlo with EA

We are considering a Sequential Monte Carlo(SMC) sampler [6] to handle the problem of sampling the trees from the posterior. After we have collected all the useful information from the movement of the trees, we now need to sample using the pheromones produced. As the algorithm 2 shows, there are three possible sampling techniques where one has a subcategory. We choose the sampling technique by drawing a uniform number between $[0, 1]$. At this point, we need to specify that on each iteration, we draw a new uniform number, so each T_i has the possibility to sample with a different strategy.

The first sampling technique uses the *temporary pheromones*, and it has a possibility of 45% to be chosen. As mentioned earlier, *temporary pheromones* are produced during the previous iteration. This sampling technique has a subcategory, where the samples classified in *positive exploration* use a different sampling technique from those listed on *negative exploration*. If the T_i is listed in *positive exploration*, we pick a stochastic move m from the list with the $ineffective\ moves$ uniformly. We then remove all the identical m from the $ineffective\ moves$ and sample uniformly with equal probabilities from the remaining $ineffective\ moves$. If the T_i is in *negative exploration*, we pick uniformly a stochastic move m from the $effective\ moves$ list to sample the particular T_i.

The second sampling technique uses the *permanent pheromones*, and it has a possibility of 45% to be chosen. As mentioned earlier, *permanent pheromones* are updated dynamically on every iteration, considering all the previous iterations.

We sample the T_i using the possibilities in the list we describe in subsection *permanent pheromones* in section 4.2.

The last sampling technique is straightforward. We use a list with stochastic moves, where each move has a uniform probability of being chosen. This sampling technique is unaffected by the pheromones. The main reason for including this technique is to ensure our algorithm is not biased, as this is the most common way of sampling in Bayesian DTs.

Algorithm 1 Pheromones Production Stage

Initialise n trees(T)
Sample trees $[T_0, T_1, ..., T_n]$
Initialise initial possibilities = $[p(G), p(P), p(C), p(S)] = [0.25, 0.25, 0.25, 0.25]$
Initialise permanent pheromones = $[p(G), p(P), p(C), p(S)]$ ▷ permanent pheromones
Initialise *positive exploration* list
Initialise $effective\ moves$ list ▷ Temporary Pheromones
Initialise *negative exploration* list
Initialise $ineffective\ moves$ list ▷ Permanent Pheromones
iterations = 10
for $(i \leq iterations, i++)$ **do**
    Evaluate trees $[T_0, T_1, ..., T_n]$
    Store their acceptance probability $[\alpha_0, \alpha_1, ..., \alpha_n]$
    for $(s \leq n, s++)$ **do**
        Draw a uniform number $u_1 \sim u[0, 1]$
        if $\alpha_s > u_1$ **then**
            append T_s to list *positive exploration*
            append $T_s move$ to list $effective\ moves$
        else
            append T_s to list *negative expolation*
            append $T_s move$ to list $ineffective\ moves$
        end if
    end for
    update *permanent pheromones* list given the $effective\ moves$ list
end for

Algorithm 2 SMC with EA

```
for i ≤ n, i++ do
    Draw a uniform number u_2 ~ u[0,1]
    if u_2 ≤ 0.45 then
        if T_i in positive exploration then
            pick uniformly a move m from ineffective moves
            Remove every identical m from the ineffective moves
            Sample T_i uniformly given the updated ineffective moves
        end if
        if T_i in negative exploration then
            Pick uniformly a move m from effective moves
            Sample T_i by applying the selected move m
        end if
    else if u_2 > 0.45 and u_2 ≤ 0.9 then
        Sample T_i using permanent pheromones
    else
        Sample T_i using initial possibilities
    end if
end for
Empty positive exploration
Empty effective moves                          ▷ Temporary Pheromones
Empty negative exploration
Empty ineffective moves                        ▷ Permanent Pheromones
```

5 Experimental Setup and Results

To demonstrate the accuracy and the run time efficiency improvements we achieved through our proposed methods, we experiment on three publicly available datasets[3] listed in Table 1. We acknowledge that the size of the datasets we conduct the experiments is small. The main reason is to have a fair comparison between the MCMC algorithm and SMC, as experiments show [13] that the former struggles to converge on an adequate time on big datasets, compared to the latter. We also aim to show that SMC, an inherently parallel algorithm combined with EA, can be a great fit within the context of big data. For each dataset, we have multiple testing hypotheses. Firstly, we compare the SMC-EA with MCMC on 1000 iterations and 10 chains for MCMC and 10 T for the SMC-EA, 100 iterations and 100 chains for MCMC and 100 T for the SMC-EA, and 10 iterations with 1000 chains for MCMC and 1000 T for the SMC-EA.

This section presents the experimental results obtained using the proposed methods with a focus on accuracy improvement and the ability of the SMC-EA to evolve smoothly in a very short period of iterations. We obtained the following results using a local HPC platform comprising twin SP 6138 processors (20 cores, each running at 2GHz) with 384GB memory RAM. We use the same hyper-parameters α and β for every contesting algorithm for testing purposes and a fair comparison and evaluation.

[3] https://archive.ics.uci.edu/ml/index.php

Table 1. Datasets description

Dataset	**Attributes**	**Instances**
Heart Disease	75	303
Lung Cancer	56	32
SCADI	206	70

We tested both MCMC and SMC-EA with a 5-Fold Cross-Validation. Results indicate what is discussed in Section 4.2. When we initialise more trees, we introduce a more diverse set, which helps improve the algorithm's optimization performance. More specifically, SMC-EA with 1000 T and 10 iterations running SCADI dataset has an accuracy improvement of $\sim 2\%$ and $\sim 6\%$ compared to having 100 T with 100 iterations, and 10 T with 1000 iterations respectively. On the Heart Disease dataset SMC-EA with 1000 T and 10 iterations has an accuracy improvement of $\sim 3\%$ and $\sim 4\%$ compared to having 100 T with 100 iterations and 10 T with 1000 iterations respectively. On the Lung Cancer dataset SMC-EA with 1000 T and 10 iterations, has an accuracy improvement of $\sim 8\%$ and $\sim 14\%$ compared to having 100 T with 100 iterations, and 10 T with 1000 iterations respectively.

On the other hand, MCMC performs poorly when we have fewer iterations and more chains compared to SMC-EA. This is expected as MCMC needs adequate time to converge. When MCMC ran for more iterations, the algorithm made better predictions, and the accuracy achieved cannot be accepted as an acceptable threshold. SMC-EA has a $\sim 12\%$ better predictive accuracy on the SCADI dataset compared to MCMC, $\sim 7\%$ on Heart Disease, and $\sim 17\%$ on Lung Cancer(see Tables 2, 3 and 4).

On the SMC-EA algorithm, trees do not have a leading T_i, and their movements are guided by interactions between them rather than one individual T dominating the group of trees. This helps to prevent individualism and stagnation in the population's evolution. Furthermore, a diverse population of trees is more effectively handled by the approach we suggest, as we avoid falling into local optimisation. The stochastic nature of the SMC-EA algorithm helps in escaping local optimisation and achieving global optimisation. Combining positive and negative feedback from the different pheromones can incorporate the benefits of successful solutions while mitigating the negative effects of poor solutions. SMC-EA algorithm fully uses all the information on the solution space, avoiding unnecessary waste or duplication of information. The EA algorithm updates the positions of all trees by using a combination of their current and past positions within the population. This allows the algorithm to maintain a history of information, preventing rapid jumps and leading to a smooth algorithm evolution.

Table 2. SCADI dataset

Chains_Trees	Iterations	MCMC	SMC-EA
10	1000	85	90.2
100	100	63	94.2
1000	10	57	96.6

Table 3. Heart Disease dataset

Chains_Trees	Iterations	MCMC	SMC-EA
10	1000	75.7	78.9
100	100	73.5	79.2
1000	10	67.7	82.7

Table 4. Lung Cancer dataset

Chains_Trees	Iterations	MCMC	SMC-EA
10	1000	70.1	73.9
100	100	69.7	79.1
1000	10	69.1	87.2

Due to the small size of the datasets we are using to conduct this study, we can only show the effectiveness of our method in exploring the solutions space faster. However, previous studies [13] have shown that the SMC DT algorithm can improve the runtime compared to MCMC DT by up to a factor of 343. We are optimistic that we can achieve the same results, as SMC and EA are inherently parallel algorithms. The main bottleneck of the SMC-EA algorithm is when we evaluate a big number of trees; for example, see the test case of 10 iterations and 1000 T. We can overcome this problem by the distributed implementation, as we can distribute the trees on many nodes and evaluate the trees concurrently.

6 Conclusion

Our study has shown that by combining two novel algorithms, the SMC and EA can tackle major problems on Bayesian Decision Tress and open the space for more research. According to our experimental results, our novel approach based on Sequential Monte Carlo and Evolutionary Algorithms explores the solution space with at least 100 times fewer iterations than a standard probabilistic algorithm, for example, Markov Chain Monte Carlo. As discussed in section 5, we managed to tackle the problem of the naive proposals, and we suggest a method

that takes advantage of the communication between the trees. We proposed a sophisticated method to propose new samples based on EA and minimised the burn-in period through SMC.

As we already mentioned, both SMC and EA are inherently parallel algorithms with existing parallel implementations [22, 25, 26]. We plan to extend this study by parallelising the SMC-EA, adding more testing scenarios with larger datasets, and showing improvement in run time.

References

1. Guillaume Basse, Aaron Smith, and Natesh Pillai. Parallel markov chain monte carlo via spectral clustering. In Arthur Gretton and Christian C. Robert, editors, *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 1318–1327, Cadiz, Spain, 09–11 May 2016. PMLR.
2. S Binitha, S Siva Sathya, et al. A survey of bio inspired optimization algorithms. *International journal of soft computing and engineering*, 2(2):137–151, 2012.
3. Sergio Blanes, Fernando Casas, and Jesús María Sanz-Serna. Numerical integrators for the hybrid monte carlo method. *SIAM Journal on Scientific Computing*, 36(4):A1556–A1580, 2014.
4. Hugh A Chipman, Edward I George, and Robert E McCulloch. Bayesian cart model search. *Journal of the American Statistical Association*, 93(443):935–948, 1998.
5. Hugh A Chipman, Edward I George, and Robert E McCulloch. Bart: Bayesian additive regression trees. *The Annals of Applied Statistics*, 2010.
6. Pierre Del Moral, Arnaud Doucet, and Ajay Jasra. Sequential monte carlo samplers. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(3):411–436, 2006.
7. Marco Dorigo and Thomas Stützle. *Ant colony optimization: overview and recent advances*. Springer, 2019.
8. Randal Douc, Arnaud Guillin, J-M Marin, and Christian P Robert. Convergence of adaptive mixtures of importance sampling schemes. *The Annals of Statistics*, 35(1):420–448, 2007.
9. Efthyvoulos Drousiotis, Dan W. Joyce, Robert C. Dempsey, Alina Haines, Paul G. Spirakis, Lei Shi, and Simon Maskell. Probabilistic decision trees for predicting 12-month university students likely to experience suicidal ideation. In Ilias Maglogiannis, Lazaros Iliadis, John MacIntyre, and Manuel Dominguez, editors, *Artificial Intelligence Applications and Innovations*, pages 475–487, Cham, 2023. Springer Nature Switzerland.
10. Efthyvoulos Drousiotis, Panagiotis Pentaliotis, Lei Shi, and Alexandra I Cristea. Capturing fairness and uncertainty in student dropout prediction–a comparison study. In *Artificial Intelligence in Education: 22nd International Conference, AIED 2021, Utrecht, The Netherlands, June 14–18, 2021, Proceedings, Part II*, pages 139–144. Springer, 2021.
11. 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.

12. Efthyvoulos Drousiotis and Paul G Spirakis. Single mcmc chain parallelisation on decision trees. In *Learning and Intelligent Optimization: 16th International Conference, LION 16, Milos Island, Greece, June 5–10, 2022, Revised Selected Papers*, pages 191–204. Springer, 2023.
13. Efthyvoulos Drousiotis, Paul G Spirakis, and Simon Maskell. Parallel approaches to accelerate bayesian decision trees. *arXiv preprint arXiv:2301.09090*, 2023.
14. Simon Duane, Anthony D Kennedy, Brian J Pendleton, and Duncan Roweth. Hybrid monte carlo. *Physics letters B*, 195(2):216–222, 1987.
15. 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.
16. Vijay Kalivarapu, Jung-Leng Foo, and Eliot Winer. Improving solution characteristics of particle swarm optimization using digital pheromones. *Structural and Multidisciplinary Optimization*, 37(4):415–427, 2009.
17. Jun S Liu, Faming Liang, and Wing Hung Wong. The multiple-try method and local optimization in metropolis sampling. *Journal of the American Statistical Association*, 95(449):121–134, 2000.
18. Enzo Marinari and Giorgio Parisi. Simulated tempering: a new monte carlo scheme. *EPL (Europhysics Letters)*, 19(6):451, 1992.
19. Per Mykland, Luke Tierney, and Bin Yu. Regeneration in markov chain samplers. *Journal of the American Statistical Association*, 90(429):233–241, 1995.
20. Christian P Robert, Víctor Elvira, Nick Tawn, and Changye Wu. Accelerating mcmc algorithms. *Wiley Interdisciplinary Reviews: Computational Statistics*, 10(5):e1435, 2018.
21. Veronika Ročková and Enakshi Saha. On theory for bart. In *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, 2019.
22. Urvashi Prakash Shukla and Satyasai Jagannath Nanda. Parallel social spider clustering algorithm for high dimensional datasets. *Engineering Applications of Artificial Intelligence*, 56:75–90, 2016.
23. Matthew A Taddy, Robert B Gramacy, and Nicholas G Polson. Dynamic trees for learning and design. *Journal of the American Statistical Association*, 106(493):109–123, 2011.
24. Duncan Taylor, Jo-Anne Bright, and John Buckleton. Interpreting forensic dna profiling evidence without specifying the number of contributors. *Forensic Science International: Genetics*, 2014.
25. Alessandro Varsi, Simon Maskell, and Paul G Spirakis. An o (logn) fully-balanced resampling algorithm for particle filters on distributed memory architectures. *Algorithms*, 14(12):342, 2021.
26. Alessandro Varsi, Jack Taylor, Lykourgos Kekempanos, Edward Pyzer Knapp, and Simon Maskell. A fast parallel particle filter for shared memory systems. *IEEE Signal Processing Letters*, 27:1570–1574, 2020.
27. Yuhong Wu, Håkon Tjelmeland, and Mike West. Bayesian cart: Prior specification and posterior simulation. *Journal of Computational and Graphical Statistics*, 16(1):44–66, 2007.