P-STreeD: A Multithreaded Approach for DP Optimal Decision Trees

__Author: Albert-Alexandru Sandu (A.A.Sandu@student.tudelft.nl) __ Responsible Professor: Dr. Emir Demirović __Supervisor: Jacobus G.M. van der Linden

1__Background

Optimal Decision Trees (ODTs) produce the best tree in terms of accuracy for a given size limit and training dataset. **Dynamic Programming** (DP) based methods were shown to outperform their counterparts [1], though challenges remain with scaling based on tree depth and feature count [2]. Leveraging modern hardware, such as multiple CPU cores, offers a promising solution to improve efficiency. To take advantage of this, **we investigate multithreading**.

2__Research Questions

Which parts of a DP approach for ODTs are the best to parallelize and how?
Is parallelizing the higher levels of the search tree more efficient than parallelizing the lower ones?
How does the speed-up achieved through multithreading scale with the dimensions of the problem? (tree depth and dataset size)

3__Preliminaries

We apply multithreading to **STreeD** [3]. It uses **recursion** in its general case, iterating over the dataset's features. For each feature, it loops through the node budget distribution for the left subtree. It then recursively calculates the misclassification score of the resulting subtrees from these splits and selects the optimal feature split based on these scores. It uses **caching** for previously stored solutions, **similarity based lower bounding** and a specialized algorithm for computing **trees of depth 2** with **incremental solving** (the terminal solver). **90% of STreeD's runtime is spent in the terminal solver**.
Therefore, an aim is to do terminal calls in parallel.

4__P-STreeD

We apply **multithreading to the outer loop** within STreeD's general recursion case. We **split features into blocks** which are assigned to threads in order to compute subproblems. Multithreading starts at the **root of the search tree**. We adjust the following:

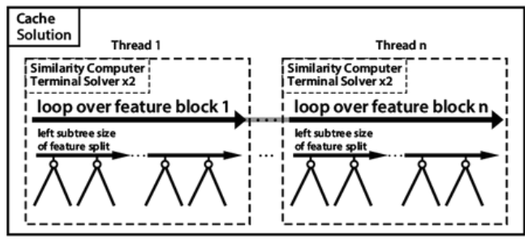
Cache: We use fine-grained locking and synchronize access to specific entries of the cache.

Terminal Solvers: STreeD uses two terminal solvers. We assign two such solvers to each thread.

Similarity lower bound computer: STreeD uses only one, whereas we use one to each thread.

Solution (the best tree): The solution updated by the subproblems is synchronized.

High-level overview of our multithreading approach, featuring local and shared memory. The branches represent recursive calls.



5__Experimental results

We compare **parallelizing at the root versus the leaf nodes**. We find that the **root-based** version generally has **better runtimes**, as well as less terminal solver calls and cache accesses. **P-STreeD** is compared with the state of the art: **DL8.5** [1], **Murtree** [2] and **STreeD** [3]. **Our method outperforms the state of the art in most tests taking more than 1 second**. Finally, we investigate the **speed-up** for up to **four threads**. **We find speed-ups of up to 2.5x for larger datasets.**

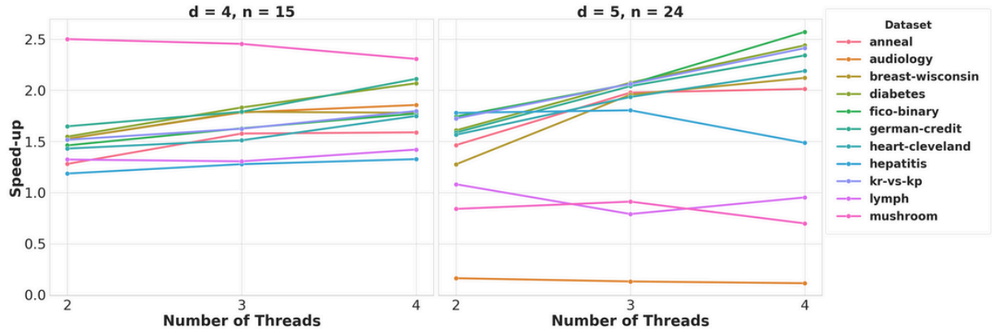
Comparison against the state of the art for depth = 5, n = 31. |D| denotes the number of instances, |F| the number of features. Time measured in seconds.

Dataset	$\|\mathcal{D}\|$	$\|\mathcal{F}\|$	DL8.5	STreeD	P-STreeD	Murtree
anneal	812	93	5	2	**<1**	2
audiology	216	148	9	<1	<1	**<1**
breast-wisconsin	683	120	12	3	**<1**	2
diabetes	768	112	72	35	**14**	20
fico-binary	10459	17	<1	2	<1	2
german-credit	1000	112	114	102	**42**	55
heart-cleveland	296	95	15	7	**3**	4
hepatitis	137	68	2	<1	<1	**<1**
ionosphere	351	445	--	301	370	**135**
kr-vs-kp	3196	73	12	6	**2**	7
lymph	148	68	2	<1	<1	**<1**
mushroom	8124	119	18	<1	<1	**<1**
pendigits	7494	216	--	271	**119**	235
tic-tac-toe	958	27	<1	<1	**<1**	<1
vehicle	846	252	258	146	**77**	137
yeast	1484	89	43	12	**6**	14
Average rank			3.72	2.72	**1.55**	2.00

Comparison between P-STreeD versions for depth = 5, n = 31. 'none' is a version with multithreading disabled. |D| denotes the number of instances, |F| the number of features. Time measured in seconds.

Dataset	$\|\mathcal{D}\|$	$\|\mathcal{F}\|$	Type	Time	D2S Calls	Cache Hits	Cache Misses
anneal	812	93	root	**<1**	9444	1469	30920
			leaf	6	13667	4059	58294
			none	2	8300	1543	27235
diabetes	768	112	root	**14**	96310	22361	215745
			leaf	28	100305	33609	294868
			none	31	95094	20985	208154
heart-cleveland	296	95	root	**3**	32733	2767	85983
			leaf	8	31699	3246	106454
			none	7	27885	2119	74101
hepatitis	137	68	root	**<1**	1427	55	3460
			leaf	<1	1470	68	3920
			none	<1	1273	42	3090
kr-vs-kp	3196	73	root	**2**	9854	584	26953
			leaf	8	9625	980	36721
			none	5	8848	528	24167
mushroom	8124	119	root	<1	1160	12	3143
			leaf	**<1**	383	2	1028
			none	<1	232	<1	603
pendigits	7494	216	root	**119**	56137	1345	148188
			leaf	208	63778	1728	177897
			none	227	47315	1036	124998

Speed-ups for numbers of threads. 'd' denotes depth, 'n' the number of feature nodes. We use our program with multithreading disabled as the baseline.



Wilcoxon- Signed-Rank Test results show that our approach generally outperforms both Murtree and STreeD. The first value is the W-Statistic and the second is the p-value (below 0.05 for statistical significane)

depth	n	P-STreeD vs Murtree
4	15	**14.0 (0.0008)**
5	24	63.0 (0.3465)
5	31	**35.0 (0.0268)**

depth	n	P-STreeD vs STreeD
4	15	1.0 (0.0000)
5	24	**26.0 (0.0077)**
5	31	**30.0 (0.0139)**

6__Discussion and Conclusions

The leaf-based version is much slower than the root-based one. Terminal calls may be done in parallel, with bounds and solutions updated to the cache too late. This results in nodes not being pruned, but also redundant computation. There is also overhead from creating threads or tasks and the leaf-based strategy involves an exponential amount of total tasks.
Our method scales well with larger datasets. This is also where we see the best results. This complements the state of the art which is able to solve the "easier" datasets very quickly. **For depths equal or larger to 5, we generally see improved runtimes.**

7__Future Work

We recommend investigating the use of a **single cache for each thread**, as well as the integration of a **thread pool**.
The applicability of P-STreeD to a supercomputer using **MPI** should be explored. This involves abstracting the threads into MPI processes and possibly setting up a dedicated MPI process for the management of the shared memory (solution and the cache).

References

[1] - Aglin, G., Nijssen, S., & Schaus, P. (2020). Learning optimal decision trees using caching branch-and-bound search. In Proceedings of AAAI-20 (pp. 3146–3153).
[2] - Demirović, E., Lukina, A., Hebrard, E., Chan, J., Bailey, J., Leckie, C., . . . Stuckey, P. J. (2022). Murtree: Optimal classification trees via dynamic programming and search. CoRR, abs/2007.12652
[3] - van der Linden, J. G. M., de Weerdt, M. M., & Demirović, E. (2023). Necessary and sufficient conditions for optimal decision trees using dynamic programming. Journal of Machine Learning Research. doi: 10.48550/arXiv.2305.19706

TU Delft
_CSE3000 Research Project