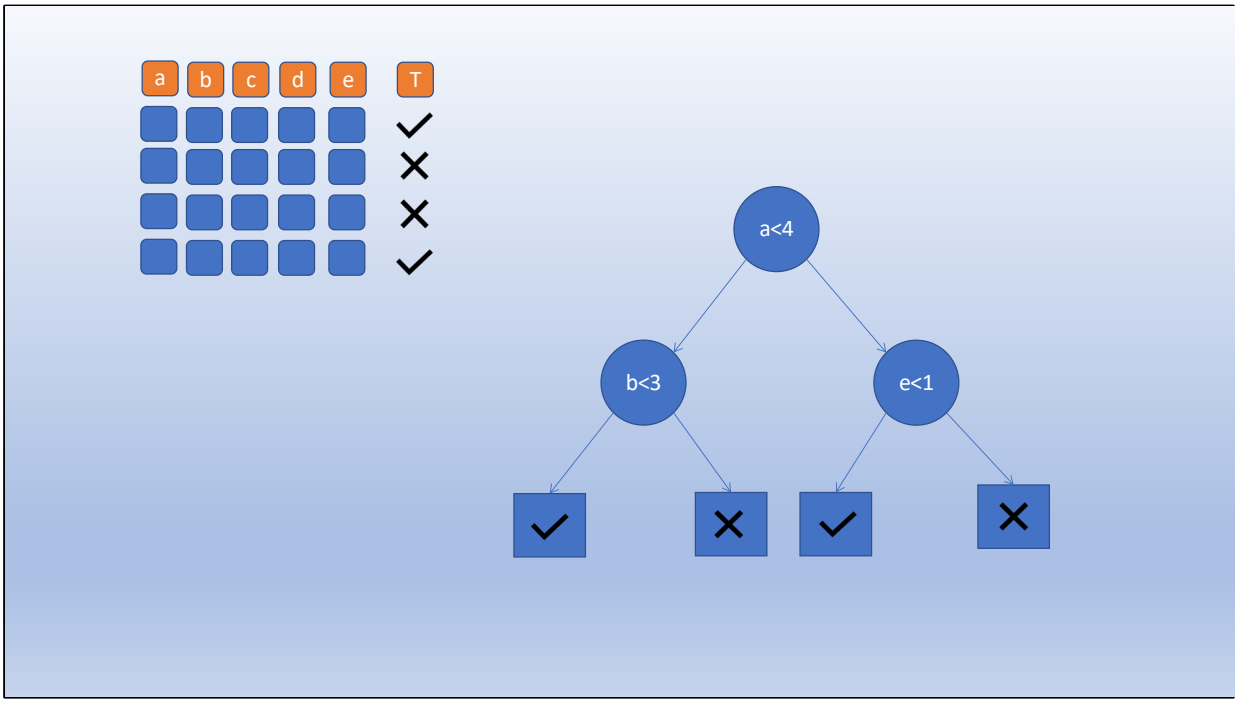
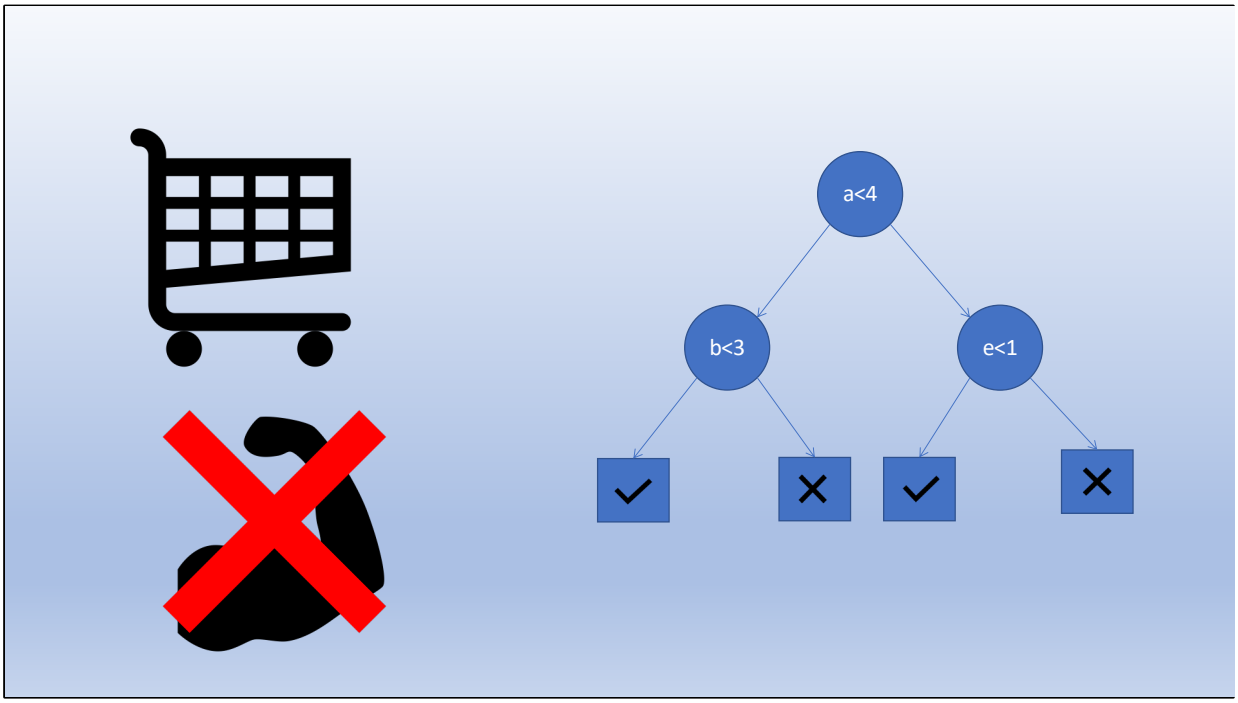
Revisiting Column Generation based Matheuristic for Learning Classification Trees

Krunal Patel
Guy Desaulniers
Andrea Lodi

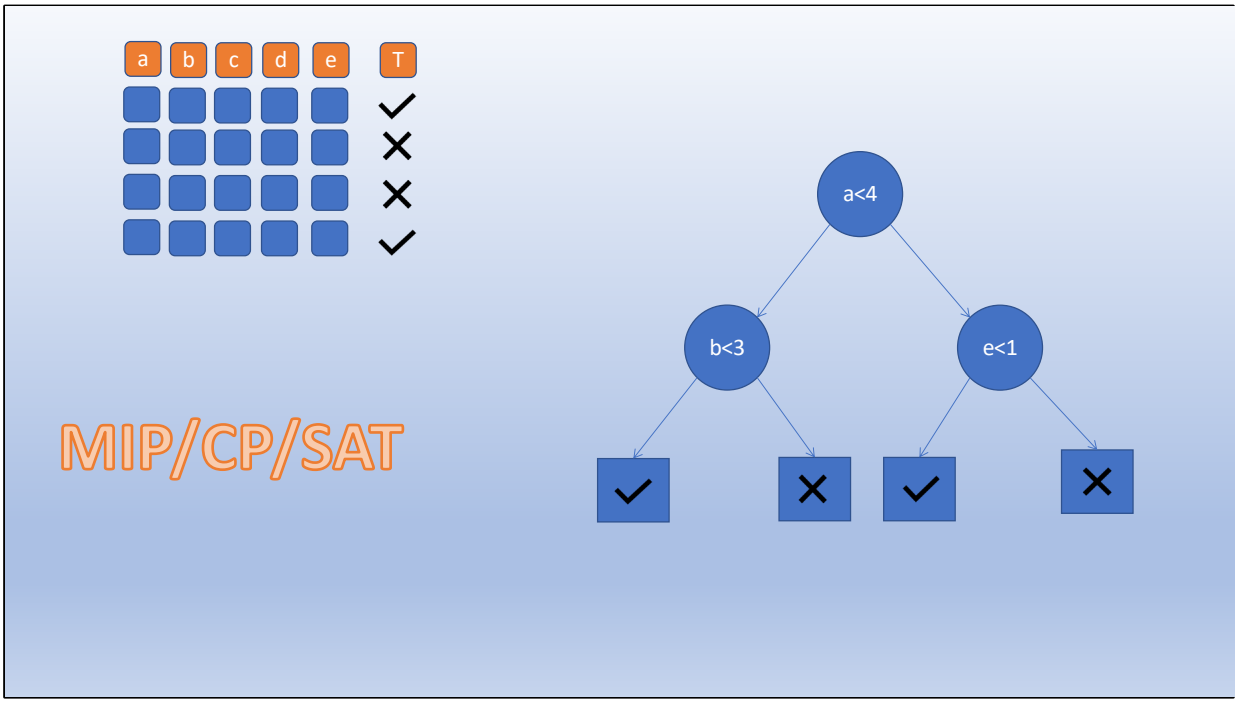
POLYTECHNIQUE MONTRÉAL
TECHNOLOGICAL UNIVERSITY



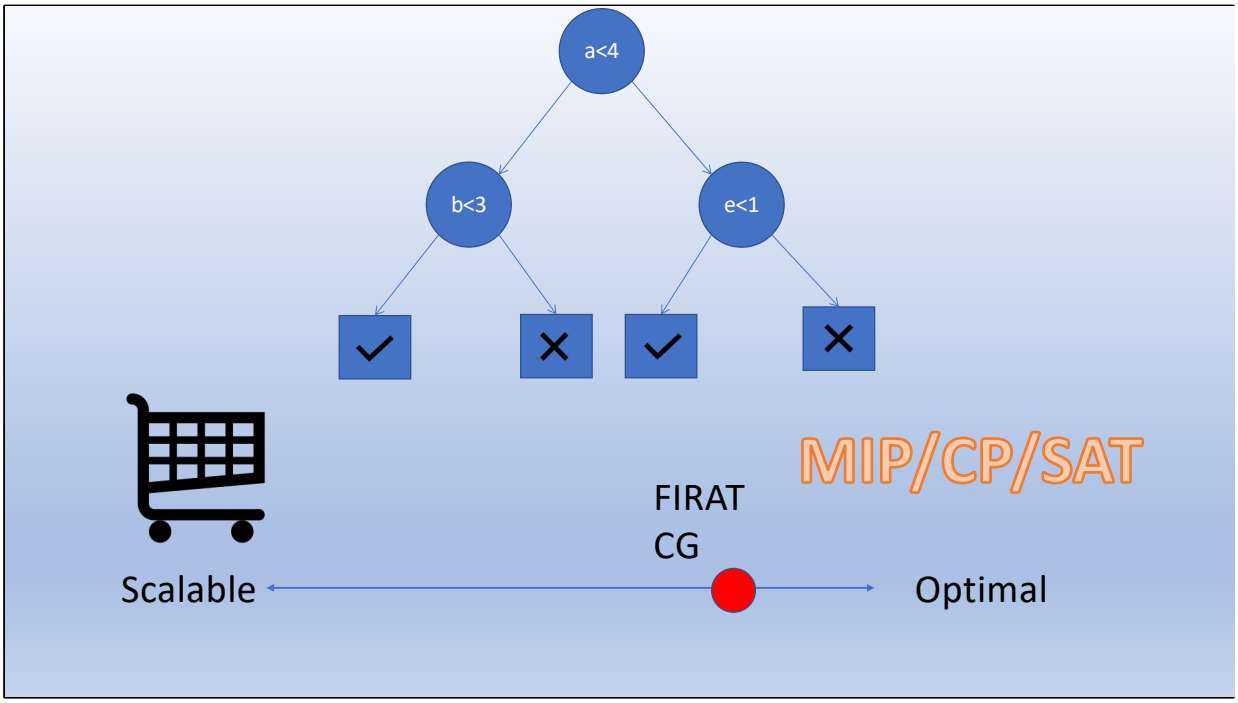
Let's say we have a dataset for a classification problem, and we want to construct a decision tree to predict the class. Each node has a specific question (which we will call a split check) based on a feature of the dataset and two branches. Each leaf has a target class.



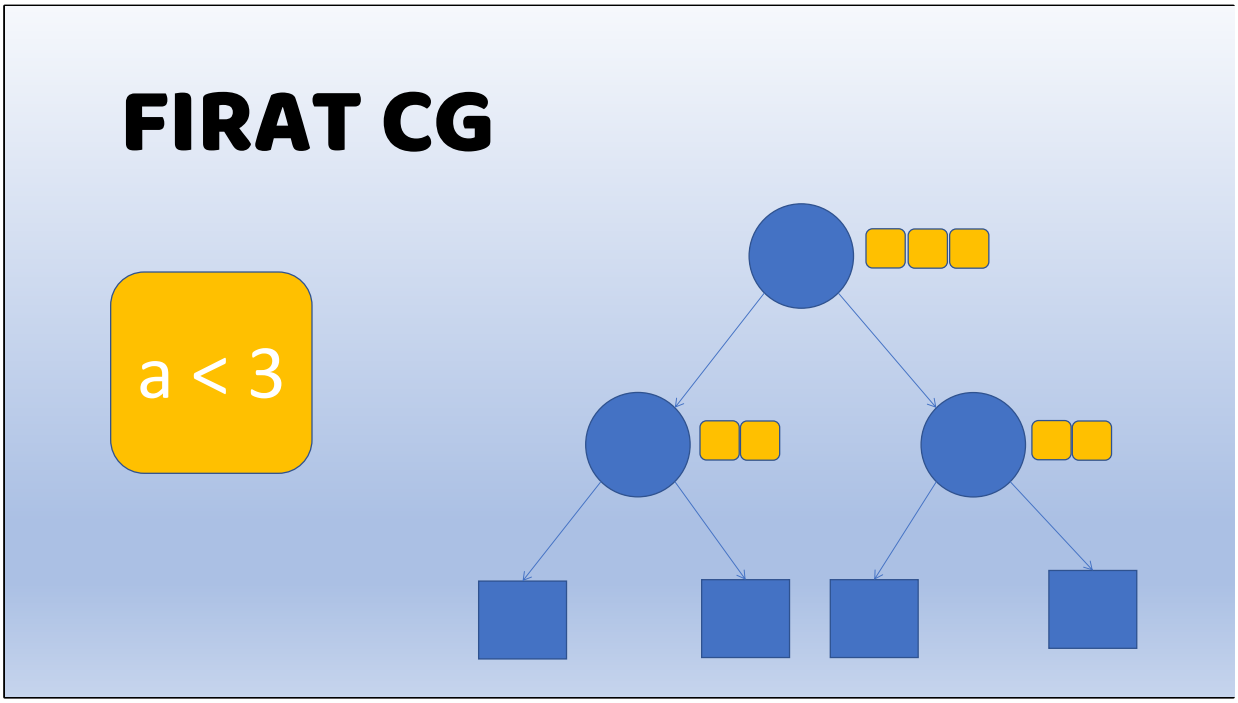
Now standard ML algorithms, for example, CART, are very fast at generating trees because they use heuristics to compute the best split check for each node without considering the overall tree structure. Goes without saying that they, in general, do not generate optimal decision trees, i.e. the trees with the best accuracy.



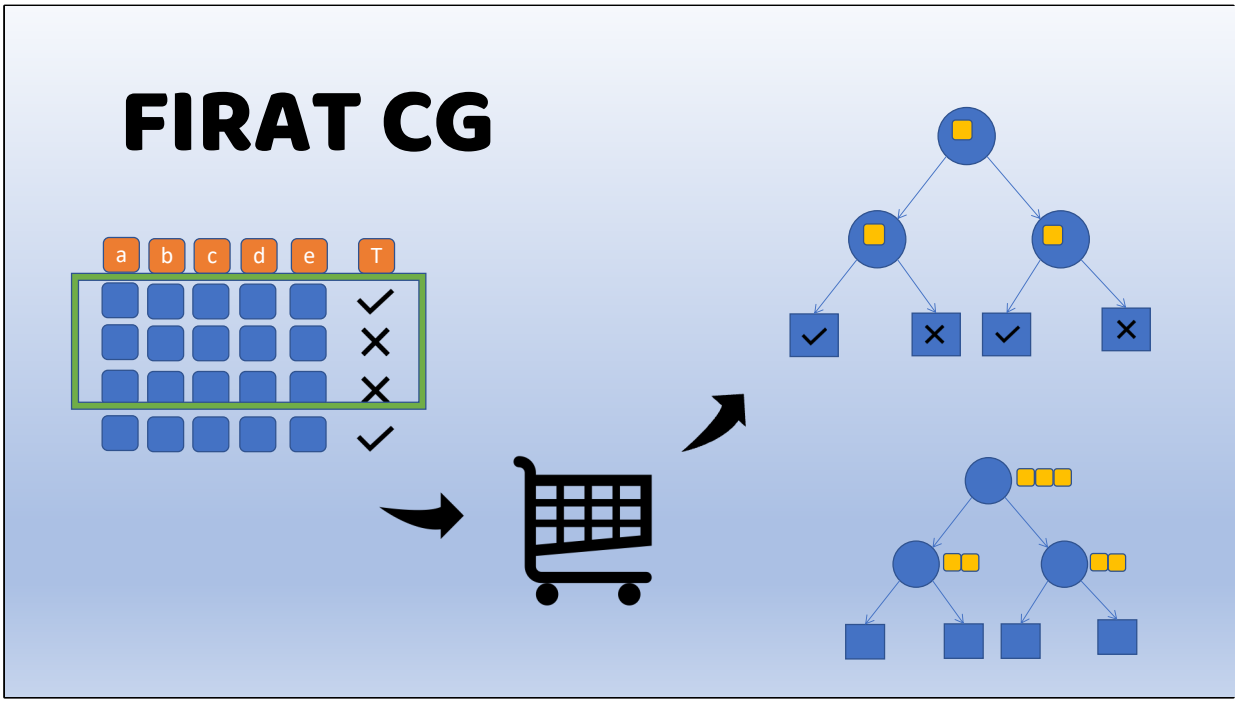
So, many people have worked on different optimization models that address this optimality issue. The problem is that they only work on small datasets and small trees.



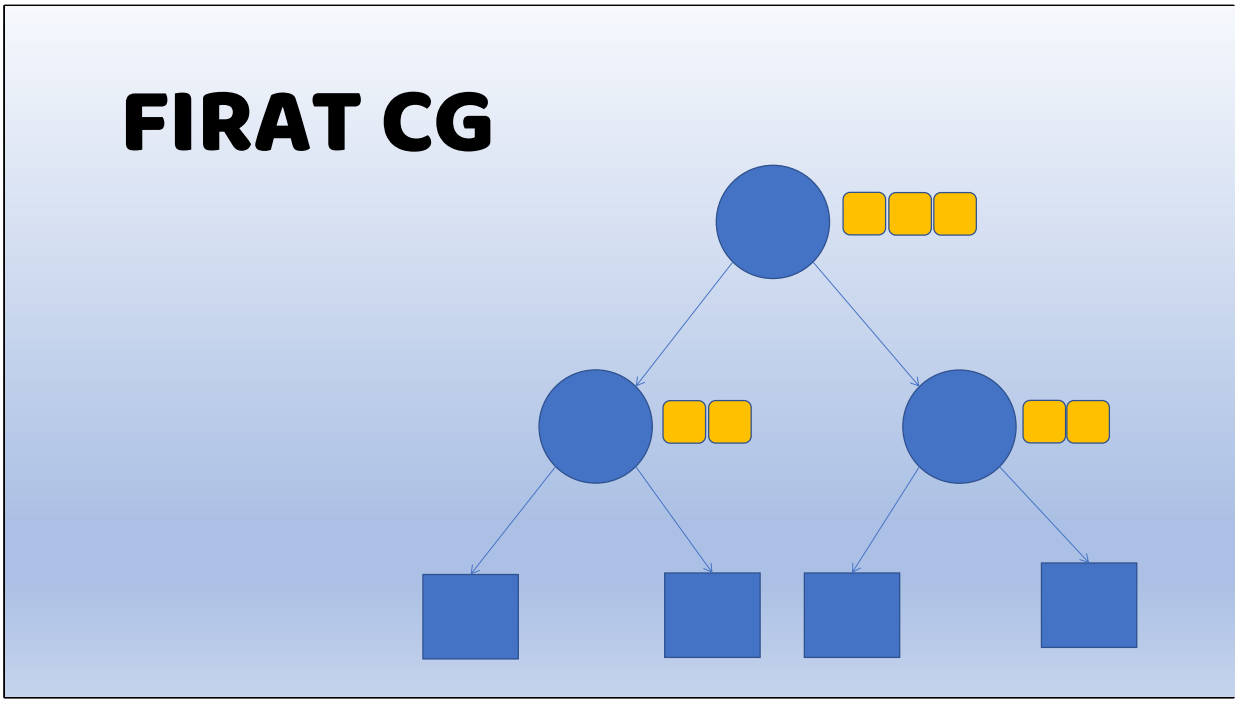
Firat et al. proposed a column generation model that somewhat mitigates this scalability issue. Now, their approach is also a heuristic. They do not generate optimal trees. But they come quite close to it and far better than CART.



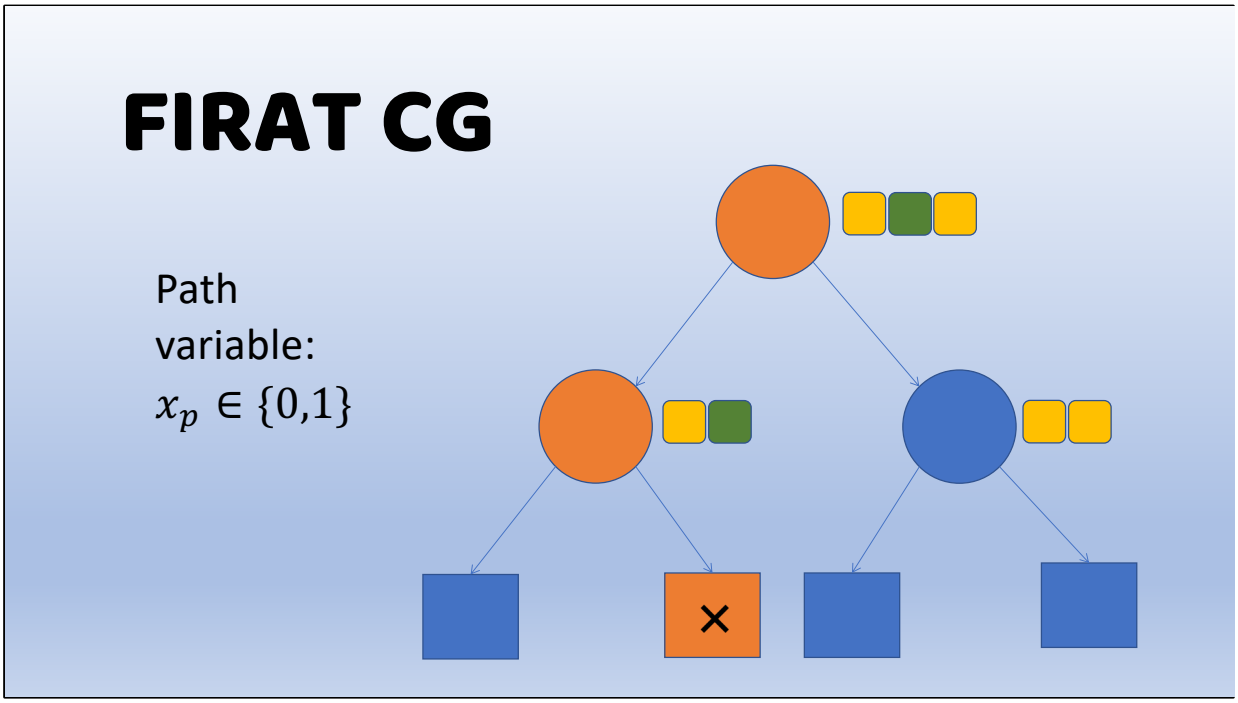
So, let's see what they proposed. They start with a restricted problem definition. We want to find a decision tree for the given dataset. In other words, if we fix the topology of the tree, then we want to find the best-split checks for each node. Now, they said, we will not look at all possible split checks. Instead, we will focus on a subset of them and select the ones that give the best accuracy. Of course, we also need to assign a target class to each leaf as well.



How do we find this subset? Easy. Just run CART on a few random samples of the dataset and whichever splits checks CART selects are all in our candidate list.



Okay, so that's now the problem definition. We have to find the best split check for each node from a list of candidate split checks to maximize accuracy. And of course, the targets for each leaf.



Let's say that this is a problem instance. We have a tree and a bunch of candidates for each node. Let's focus on a root to leaf path with one split check selected for each node in the path and a target for the leaf. They created a binary variable for each such path. If the variable is 1, that path is selected.

Now we can write a bunch of constraints to make sure that the selected paths define a valid tree. But, as you might have noticed already, the number of such possible path variables is just too large. That's where we use column generation. We start with a

few and then generate these path variables as needed through subproblems.

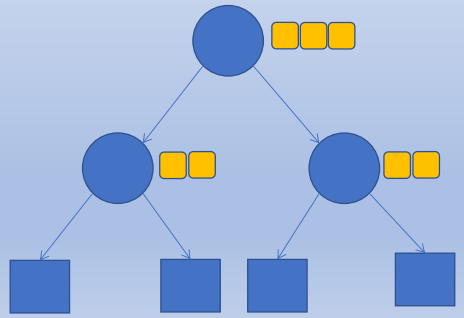
Master Problem

Objective: Maximize accuracy

α: For each leaf L, exactly one path with leaf L is selected.

β: For each row R, exactly one path satisfied by R is selected.

γ: Any pair of selected paths must agree on the common node split checks.



So, let's see a quick overview of the master problem. There are three main constraints.

First, we need to make sure that exactly one path is selected for each leaf node.
Second, we need to make sure that each row in our dataset follows exactly one selected path.
And third, we need to ensure that the selected paths agree with each other on common nodes. That is any two paths have the same split check on all common nodes. Now, this requires some additional variables but they are not too many. So, only path variables are generated through column generation.

Obviously the objective is to maximize accuracy.

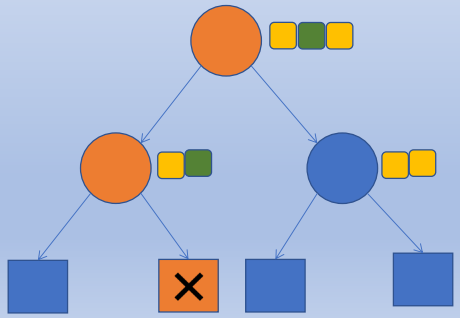
Sub Problem

Branch-and-Price?

Objective: Maximize reduced cost

Main Constraints: Ensure that the accuracy of the generated path is correctly computed.

One subproblem
for each leaf
and each target.



And how are these path variables generated? As usual in column generation. We collect the dual costs of these constraints and compute the expression of reduced cost of a path variable. That's our objective. We maximize it.
This expression involves how many rows are correctly classified. So, we need some variable to represent that a row is following the path and is correctly classified. And we just add a few constraints to ensure that.

And finally, we only solve root node with column generation. Whatever columns are generated by

then, use take them and solve the integer program. So, not the complete branch and price. Same as FIRAT.

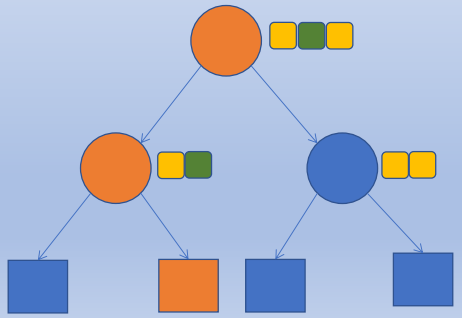
Merged Sub Problem

Objective: Maximize reduced cost

Main Constraints: Ensure that the accuracy of the generated path is correctly computed.
and compute target.

One subproblem
for each leaf

Larger sub problem:
Involves extra constraints and variables.



In their model, they defined one subproblem for each leaf and target pair. But it turns out that we can add a few more variables and constraints to move the target computation inside the subproblem.

This basically reduces the number of subproblems. But at the same time it makes each subproblem a bit harder to solve. Precisely, this involves adding 2 times the number of rows + 1 extra constraint and the number of rows plus the number of targets extra variables.

So, is it worth it? That we will discuss when I show

the results.

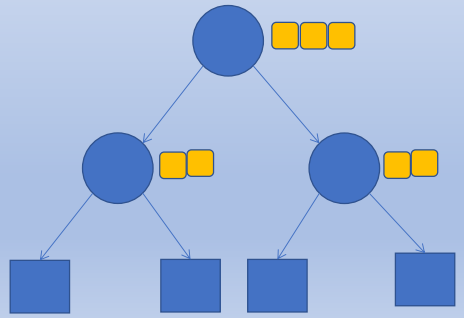
Master problem

Objective: Maximize accuracy.

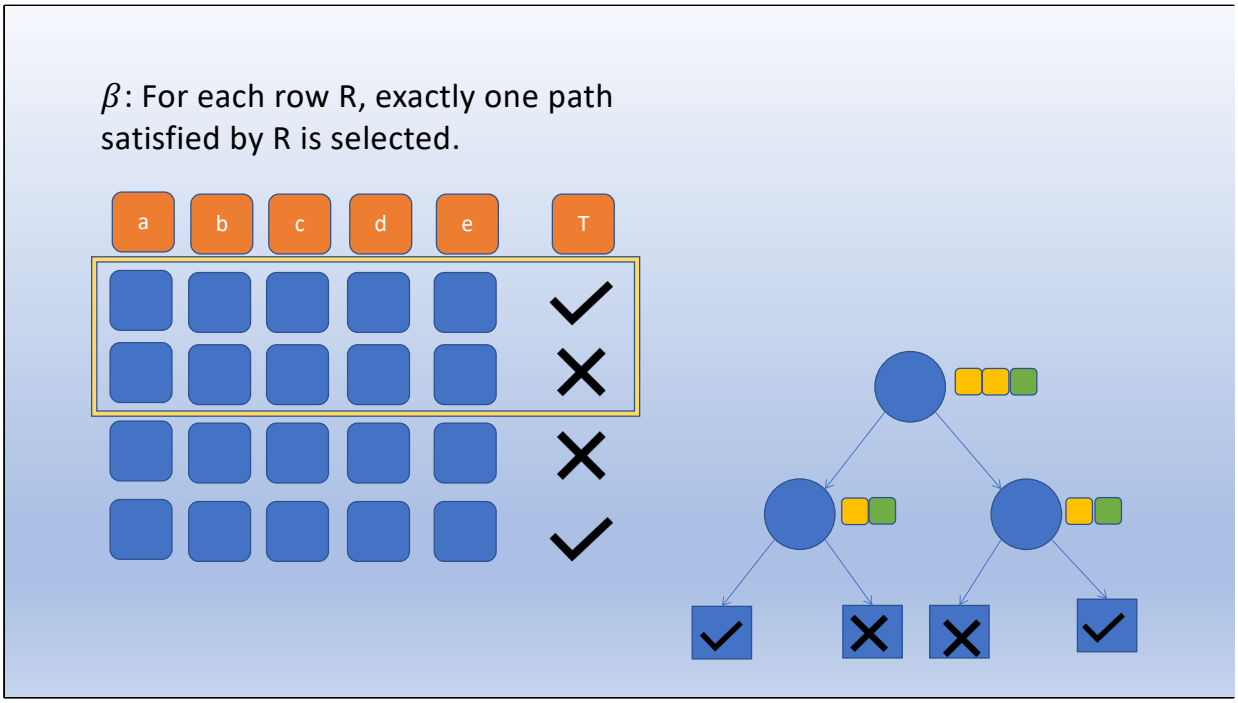
α: For each leaf L, exactly one path with leaf L is selected.

β: For each row R, exactly one path satisfied by R is selected.

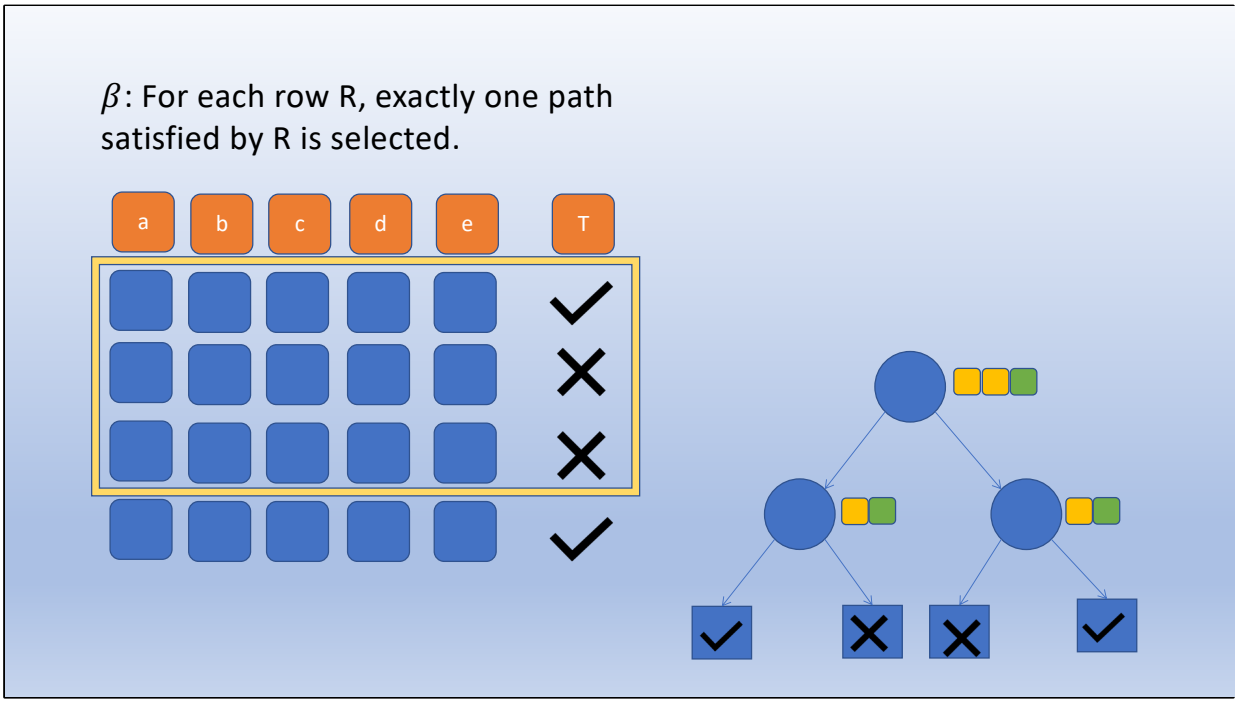
γ: Any pair of selected paths must agree on the common node split checks.



Let's look again at the master problem. Some of you might have noticed that the second (beta) constraints are not really needed. In other words, the First (alpha) and the third (gamma) constraints are sufficient to define a valid tree.



Now, instead of discussing a formal proof, I will give you an informal argument. Let's say that we have a dataset. We used this model and generated an optimal tree for given candidate splits. This is a valid tree. We can use it to predict classes on unseen rows.



Now let's say that we now decided to reduce our test dataset size and move some of the rows to our training dataset.

It is possible that this tree that we have generated, is no longer optimal. But it is certainly feasible. It did classify those extra test rows before. So, it can still classify them. In short, the optimality is affected, but not feasibility. That is the corresponding beta constraints are satisfied for the new rows.

Master problem

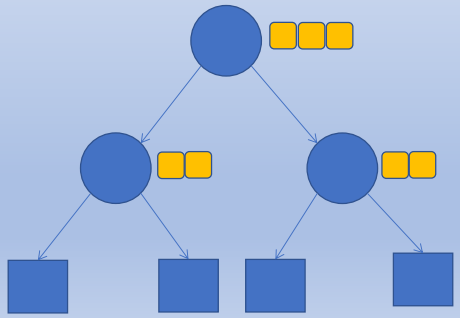
Objective: Maximize accuracy.

α: For each leaf L, exactly one path with leaf L is selected.

~~β: For each row R, exactly one path satisfied by R is selected.~~

γ: Any pair of selected paths must agree on the common node split checks.

Poor LP relaxation.



So, can we remove them? Well, they are satisfied for the integer program. But not necessarily for the LP relaxation. It turns out that these rows are helpful in the sense that they provide a tighter relaxation.

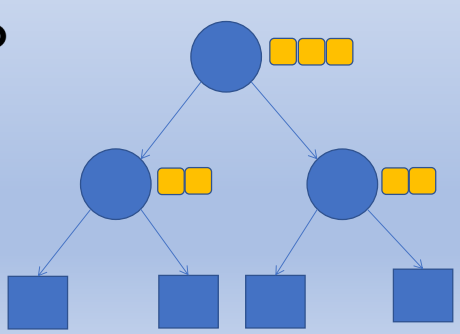
Master problem

Objective: Maximize accuracy.

α: For each leaf L, exactly one path with leaf L is selected.

β: For each row R, exactly one path satisfied by R is selected.

γ: Any pair of selected paths must agree on the common node split checks.



So, we can't remove them. But we can do the next best thing. We can add them as cuts. There are many ways to do this but I did it this way. I would execute my normal column generation iterations. On every 10th iteration, I check if there is any beta constraint violated. And I add a few violated constraints to the model and resolve the restricted master LP again. And continue with the column generation.

Now, this begs a question. These constraints that make the formulation stronger are dependent on the

dataset. If I have a different set of rows, the strength of the model will be different. And that's not so good. I might have a garbage dataset and that results in a bad relaxation. Can we fix this?

Master problem

Objective: Maximize accuracy.

α: For each leaf L, exactly one path with leaf L is selected.

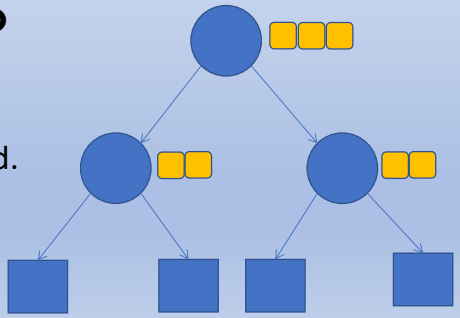
β: For each row R, exactly one path satisfied by R is selected.

Generated β: For each generated row R', exactly one path satisfied by R' is selected.

γ: Any pair of selected paths must agree on the common node split checks.

SAT model. Can be linearized.

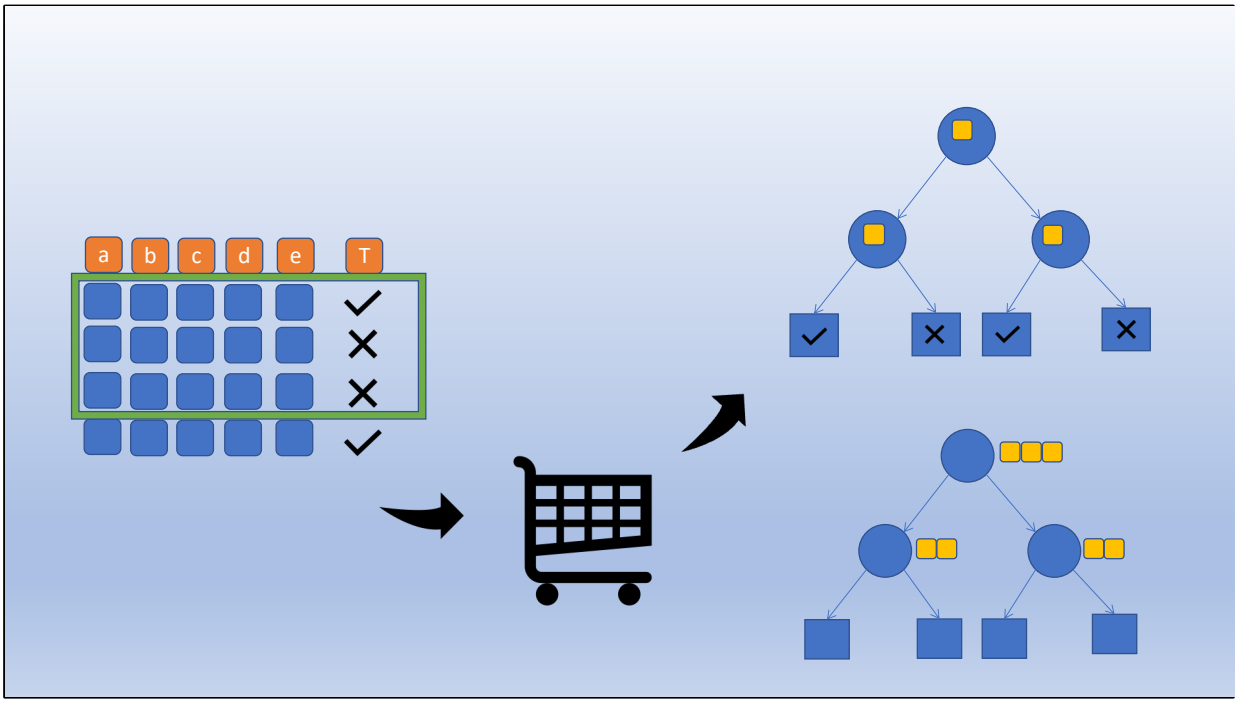
For each split check, decide which branch to take in order to maximize the violation.



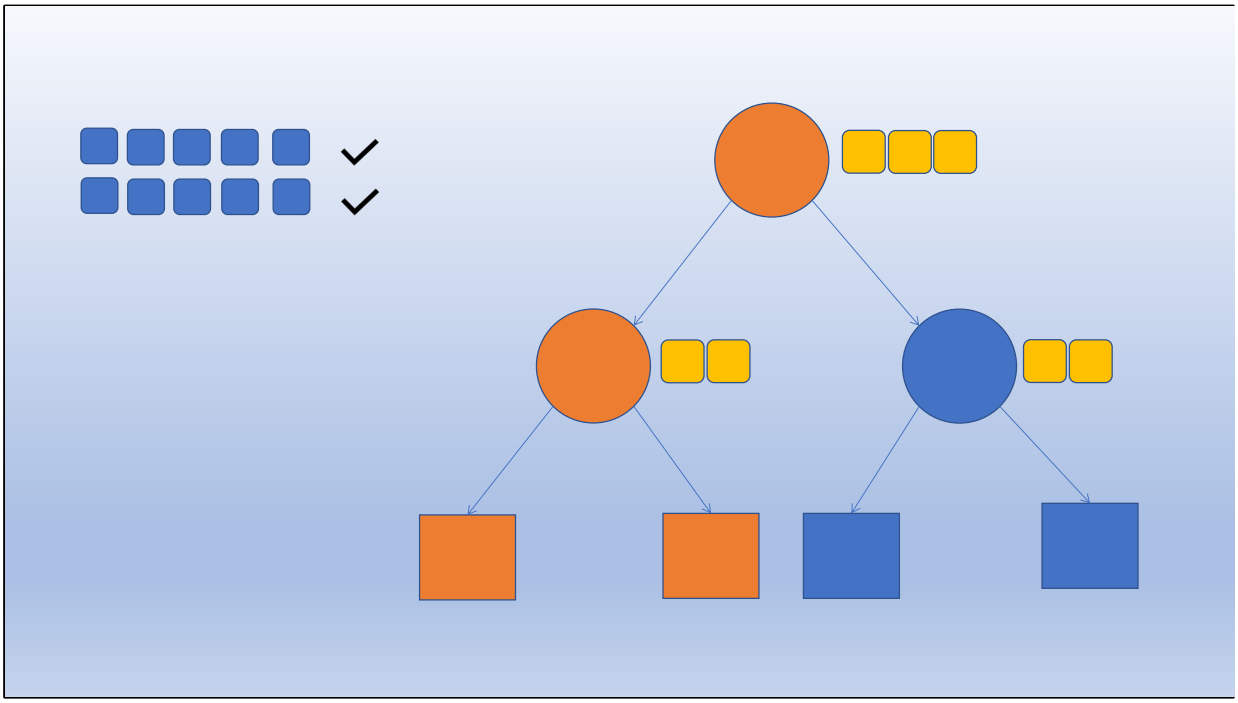
The answer is YES. We can actually generate datapoints for which the corresponding constraints are violated by LP relaxation. Note that I only need to generate the features of the rows and not the target. I don't care if the generated data point is classified correctly because I am not updating the objective.

I only need to figure out which branch this new data row will follow on each split check. And those are my binary variables.
This is a SAT model, but we can also linearize it. I

kept it this way and used a CP-SAT solver to solve this problem.



Finally, the biggest bottleneck of column generation is: generating columns. We avoid doing this as much as possible by putting more promising columns in the beginning. These columns come from the problem definition stage where we generated the candidate splits using CART, we can also take the entire paths and put them in the master problem. FIRAT only added paths from the last complete execution of CART. We do it for some more.

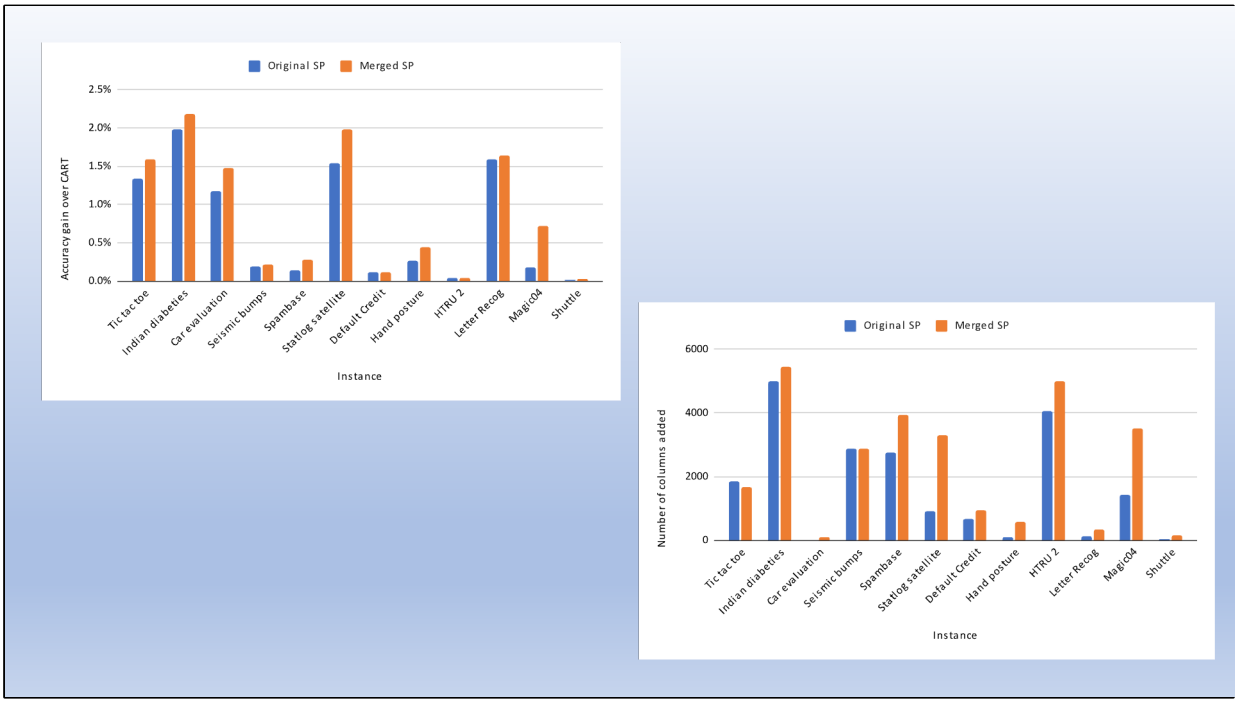


And now that our problem is restricted to a set of candidate splits, we can eliminate rows as well. If there is a pair of rows that take the same branch on all candidate split checks, we can remove one of them. We don't even need them to agree on all candidate split checks. We just need them to agree on candidate split checks of the nodes they can reach.

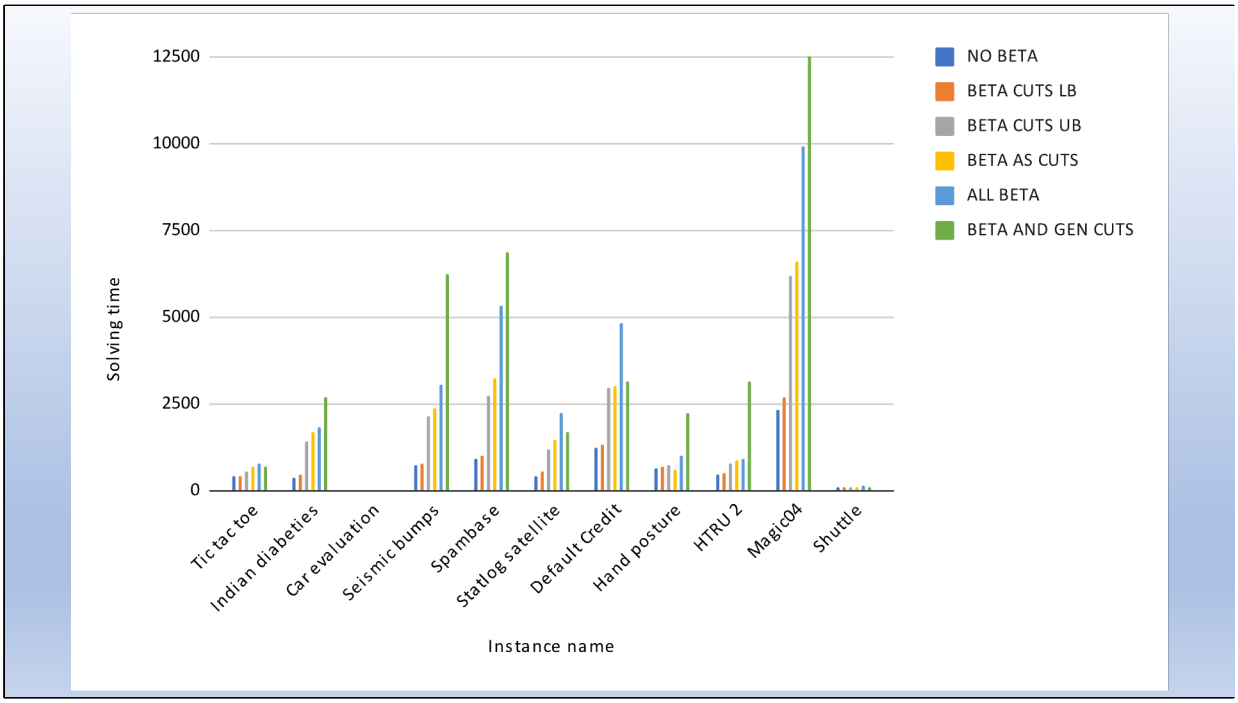
Dataset	Number of rows	Number of features	Number of classes
Small			
Tic tac toe	958	18	2
Indian diabetes	768	8	2
Car evaluation	1728	5	4
Seismic bumps	2584	18	2
Spambase	4601	57	2
Statlog satellite	4435	36	6
Large			
Default Credit	30000	23	2
Hand posture	78095	33	5
HTRU 2	17898	8	2
Letter Recognition	20000	16	26
Magic04	19020	10	2
Shuttle	43500	9	7

So, let's see how all these worked out. We used the same dataset as them. It was open sourced. Except, we are not interested in tiny datasets that is less than 500 rows. So, we selected a total of 12 datasets (6 small (<10000) and 6 large (>10000). For each, we ran on 5 different train test splits. This is again same as FIRAT.

Other non-CG approaches fail at handling large datasets.

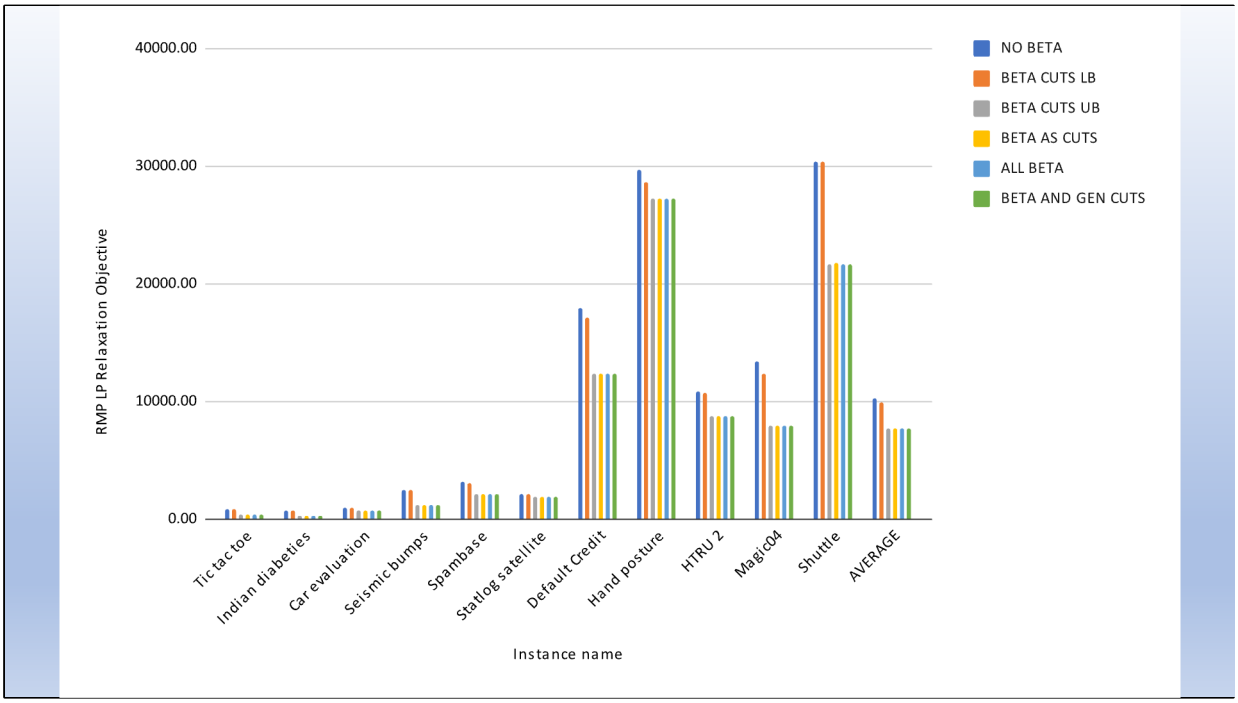


Merging subproblems is always better. The merged subproblem is faster and hence it results in more column generation iterations in the given time limit. This results in higher gains over CART compared to the original SP.

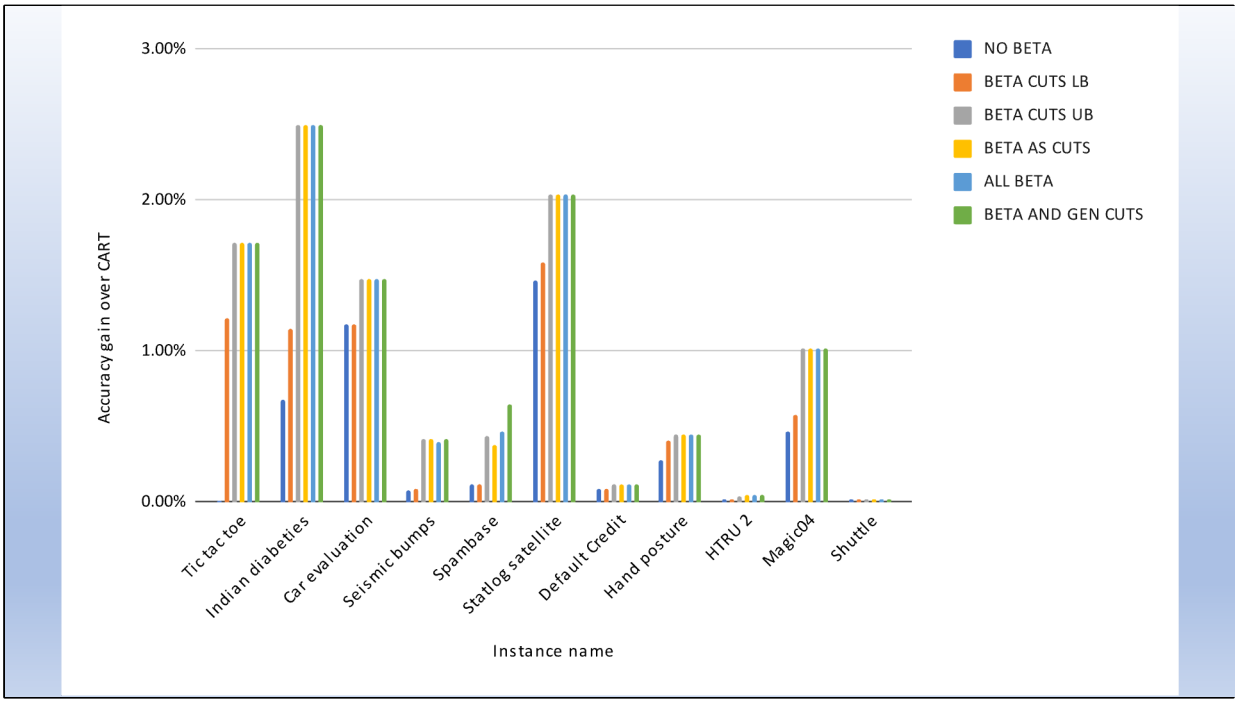


We tried 6 ways of using Beta constraints. The first one is not using them at all. Next three variants use them as cuts (with lower bound, upper bound, and equality constraints). The next variant uses all given beta constraints (used by Firat). The last variant uses beta constraints as cuts and generates additional cuts (using SAT model). We measure the solving time.

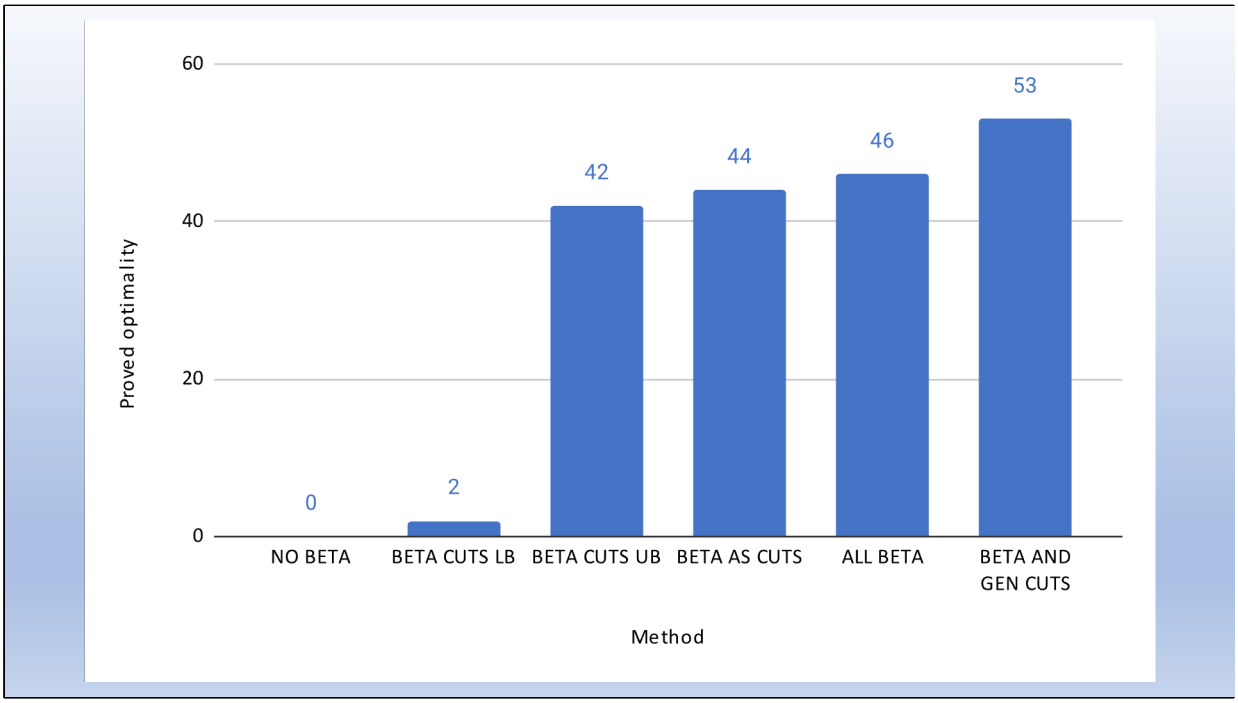
The first two variants are the fastest. But we will see in the next slides that they provide weaker LP relaxation and result in lower gains over CART. The next two variants are the best in terms of solving time. Using all given beta constraints is always slower. Finally, generating extra cuts takes the longest because of the solving time for SAT model and the added column generation iterations because of added cuts.



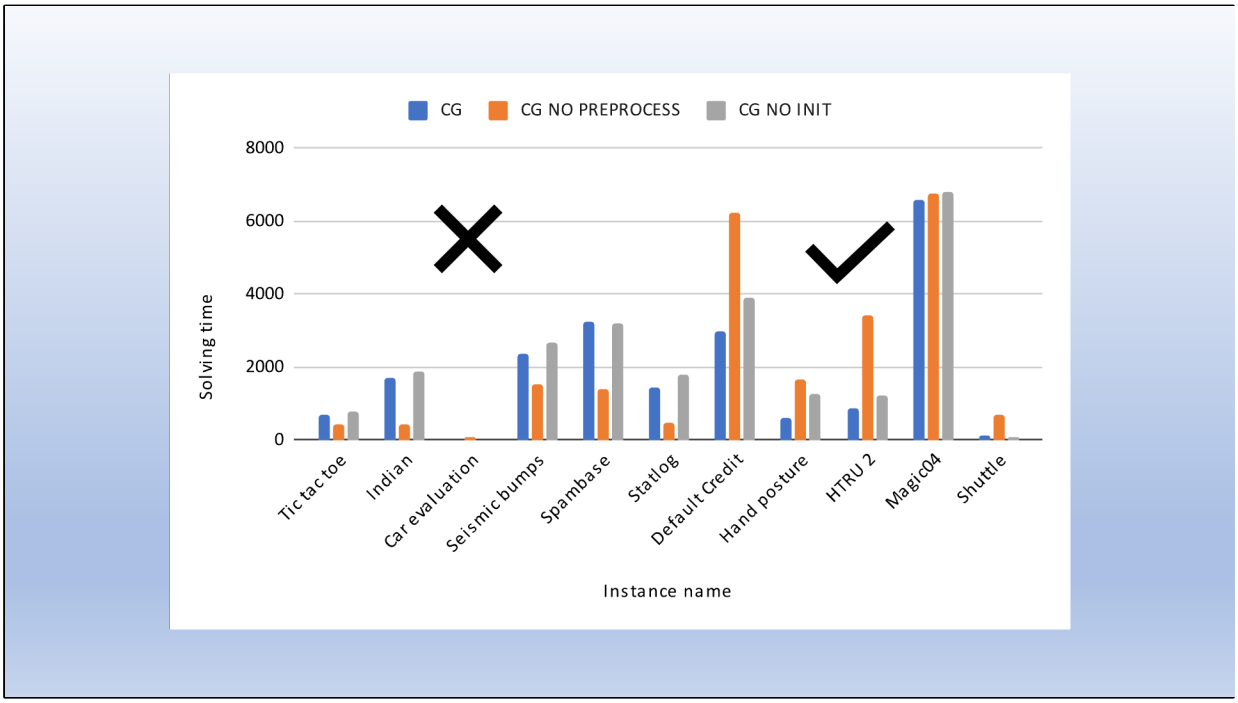
As discussed, the first two variants provide worse LP relaxation. The other four have minor differences but not noticeable in the graph.



And this translates into gains over CART. Lower for the first two variants. The other 4 are similar. So, because of their lower solving time, the BETA CUTS UB and BETA AS CUTS are best overall.



The generated cuts help with ”proving” optimality. This is done by having good LP relaxation at root and finding the integer solution with the same objective. BETA AND GEN CUTS could prove optimality for 53 out of 55 instances. This shows that the branch and price is not really needed (won’t result in much gains) for this approach.



Finally, effects of preprocessing and extra initialization. 3 variants. One where everything is on. One where preprocessing is off. And one where extra initialization is off.

It is always better to use extra initialization.

For preprocessing, the story is more complex. On smaller instances, preprocessing is harmful but works very well on the larger instances.

		Firat CG approach			Updated CG approach		
Instance	Depth	CART accuracy	CG accuracy	Accuracy gain	CART accuracy	CG accuracy	Accuracy gain
Tic tac toe	2	71.2%	71.8%	**0.6%**	71.3%	71.8%	0.5%
	3	75.4%	76.7%	1.3%	75.4%	77.4%	**1.9%**
	4	84.4%	85.4%	1.0%	84.5%	86.2%	**1.8%**
Indian diabeties	2	77.3%	78.8%	**1.5%**	77.3%	78.8%	**1.5%**
	3	78.9%	81.2%	2.3%	78.9%	81.3%	**2.4%**
	4	82.9%	84.2%	1.3%	82.9%	85.3%	**2.4%**
Car evaluation	2	76.9%	76.9%	**0.0%**	76.9%	76.9%	**0.0%**
	3	79.0%	79.8%	0.8%	79.0%	80.2%	**1.2%**
	4	84.2%	85.2%	1.0%	84.2%	85.7%	**1.5%**
Seismic bumps	2	93.1%	93.3%	0.2%	93.1%	93.4%	**0.3%**
	3	93.4%	93.7%	**0.3%**	93.4%	93.7%	**0.3%**
	4	93.9%	94.2%	0.3%	93.9%	94.3%	**0.4%**
Spambase	2	86.0%	87.1%	1.1%	85.9%	87.2%	**1.3%**
	3	89.6%	90.3%	**0.7%**	89.6%	90.3%	0.6%
	4	91.6%	91.6%	0.0%	91.6%	92.0%	**0.4%**
Statlog satellite	2	63.2%	64.0%	0.8%	63.2%	64.3%	**1.1%**
	3	78.7%	79.5%	0.8%	78.7%	80.0%	**1.3%**
	4	81.6%	82.9%	1.3%	81.6%	83.6%	**2.0%**

Comparisions with old CG approach. 600s time limit. Clearly the new approach gains more over CART compared to the original approach.

		Firat CG approach			Updated CG approach		
Instance	Depth	Firat CART accuracy	Firat CG accuracy	Firat Accuracy gain	CART accuracy	CG accuracy	Accuracy gain
Default Credit	2	82.3%	82.3%	**0.0%**	81.9%	81.9%	**0.0%**
	3	82.3%	82.3%	**0.0%**	82.0%	82.0%	**0.0%**
	4	82.3%	82.3%	**0.0%**	81.9%	82.0%	**0.0%**
Hand posture	2	56.4%	56.4%	**0.0%**	56.6%	56.6%	**0.0%**
	3	62.5%	62.8%	0.3%	62.6%	63.0%	**0.4%**
	4	69.0%	69.1%	0.1%	69.4%	69.7%	**0.3%**
HTRU 2	2	97.8%	97.8%	**0.0%**	97.6%	97.6%	**0.0%**
	3	97.9%	97.9%	**0.0%**	97.7%	97.7%	**0.0%**
	4	98.0%	98.0%	**0.0%**	97.8%	97.7%	-0.1%
Letter Recog	2	12.5%	12.7%	0.2%	12.3%	12.7%	**0.4%**
	3	17.7%	18.6%	0.9%	17.6%	19.6%	**2.0%**
	4	24.8%	27.0%	2.2%	24.5%	27.8%	**3.3%**
Magic04	2	78.4%	79.1%	**0.7%**	79.0%	79.7%	**0.7%**
	3	79.1%	80.1%	**1.0%**	79.1%	79.9%	0.8%
	4	81.5%	81.5%	0.0%	81.6%	82.4%	**0.8%**
Shuttle	2	93.7%	93.7%	**0.0%**	93.9%	93.9%	**0.0%**
	3	99.6%	99.7%	**0.1%**	99.6%	99.7%	0.0%
	4	99.8%	99.8%	**0.0%**	99.8%	99.8%	**0.0%**

Results on larger datasets on testing sets. The training results are not available for Firat's approach. Since we don't optimize for generalization of performance, we see a lot of zeros in gain over cart for both approaches. But there are significant gains for example in letter recog.

Conclusion

Q & S

Thank you!

We improved master problem, subproblem, and preprocessing and initialization stage of the column generation approach. So, overall, we improved trees!