A Simulated Annealing Approach to Designing Optimal Decision Trees for Classification, Prescriptive, and Survival Analysis

by

Suleeporn Sujichantararat

S.B., Mahidol University (2010)
S.M., Imperial College London (2016)
S.M., Massachusetts Institute of Technology (2020)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2024

Authored by:	Suleeporn Sujichantararat Department of Electrical Engineering and Computer Science January 24, 2024
Certified by:	Dimitris Bertsimas Boeing Professor of Operations Research, Thesis Supervisor
Accepted by:	Leslie A. Kolodziejski Professor of Electrical Engineering and Computer Science Chair, Department Committee on Graduate Students

A Simulated Annealing Approach to Designing Optimal Decision Trees for Classification, Prescriptive, and Survival Analysis

by

Suleeporn Sujichantararat

Submitted to the Department of Electrical Engineering and Computer Science on January 24, 2024 in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

ABSTRACT

A binary decision tree is a highly interpretable machine learning model, as humans can easily understand how a prediction is made by answering a series of binary questions. Earlier work has provided a powerful framework for constructing optimal decision trees by utilizing multiple random warm starts and local search to iteratively improve each warm start until a locally optimal decision tree is found. However, local search does not guarantee global optimality if the number of random warm starts does not exceed the number of local minima. Hence, we propose to incorporate simulated annealing into decision tree construction, as some worse transformations could lead to a better final model. We focus on three problem domains: classification, prescriptive and survival analysis to produce Optimal Classification Trees with Simulated Annealing (OCT-SA), Optimal Policy Tree with Simulated Annealing (OPT-SA), and Optimal Survival Tree with Simulated Annealing (OST-SA). This approach further improves on OCT, OPT, and OST by probabilistically allowing a tree to move to a tree with a worse objective value. A challenge in designing OCT-SA, OPT-SA, and OST-SA is to define an appropriate simulated annealing cooling schedule that leads to a global optimal solution in practical runtime. We evaluate OCT-SA, OPT-SA, and OST-SA on widely-used benchmarking real-world datasets. The evaluation metrics are out-of-sample performances over unseen test datasets: Gini impurity scores for classification, mean rewards scores for prescriptive, and local full likelihood score for survival analysis. The results demonstrate that our simulated annealing framework benefits the decision trees construction process.

Thesis supervisor: Dimitris Bertsimas
Title: Boeing Professor of Operations Research

Acknowledgments

First and foremost, I would like to thank my PhD supervisor, Prof. Dimitris Bertsimas, for his endless support and for granting me an opportunity to become a PhD student under his supervision. The reasons for my interest in working with him were my research interest in the healthcare domain, and his compelling stories about himself and how he views the world on his personal web page. He asked me: "Can you tell me why I should accept you as my PhD student, given that your academic background is not very strong?" I responded to him with a firm voice and determined eyes: "I never give up. Once I set my goal, I will do whatever I can to achieve that goal." He responded: "Good! You have a strong perseverance, and this is a characteristic of a successful person. I will open the door for you." The door of opportunity in proving myself was opened in January 2020 when I first started working with Prof. Bertsimas. He is definitely a tough and demanding supervisor, but also very supportive. In my view, he is like a strict father who is hard on his daughter, i.e., me, as he wants his daughter to succeed. Then, during the MIT Independent Activities Period (IAP) and spring semester in 2022, it was a time I almost gave up on this PhD journey. He noticed that there was something wrong with me mentally, so he let me pause my research activities to have a time to rest, and reflect, before making any critical decision which could not be undone. Then in summer 2022, I started to have a physical health problem, and it continued until fall 2022. This problem definitely delayed my graduation, which was first planned to be in spring 2023. He provided me with very generous support, at a level that I felt he treats me like his family member. Prof. Bertsimas, I sincerely thank you from the bottom of my

heart. Without your support, I would not have gone this far. I will keep improving myself to make you proud, and not regret granting me an opportunity to be your PhD student.

Second, I would like to thank Prof. Devavrat Shah and Prof. Martin Wainwright, my thesis committee. Thank you so much for your time in providing guidance to help improve the quality of my thesis. The opportunity to practice and comments I obtained during our thesis committee meeting on August 10, 2023 were incredibly helpful. I have revised my experiments, writing, and presentation accordingly. Most importantly, thank you so much for allowing me to schedule my thesis defense, and going through my thesis report. I will follow your advice and address any feedback you have provided in order to be a better researcher. In addition, I would like to thank Prof. Shah, as well as Prof. Marzyeh Ghassemi, who also served on my EECS Research Qualifying Examination (RQE) committee. Since I had a physical health problem in fall 2022, the RQE got delayed and both of them were so accommodating in scheduling a new exam time in the spring semester on March 8, 2023. Their RQE recommendations were also crucial in shaping subsequent versions of my research experiments, and determining how I can better communicate my work to the scientific communities.

Third, I would like to thank Prof. Charles Leiserson, my academic lifesaver at MIT. At the beginning of the fall semester in 2019, I almost had to withdraw from MIT as I had no funding to pay my tuition fee, and also no PhD supervisor. Fortunately, Prof. Leiserson accepted me as a TA for his 6.172 Performance Engineering of Software Systems class. In order to help me fulfill another requirement from the EECS department in continuing as a PhD student, Prof. Leiserson was also very sympathetic in accepting me as his PhD student. However, after I had a chance to spend a decent amount of time doing research during the Thanksgiving break that year, I unexpectedly figured out that this was not the research area I was passionate about. On the day I had to tell Prof. Leiserson that I would like to switch my research group, I was so worried and nervous. Prof. Leiserson told me: "No hard feelings.," and still proposed to give a hand whenever I needed help from him. Then, he handed me a book titled *Give and Take: Why Helping Others Drives Our Success* to demonstrate his philosophy in helping people, and to stop me from worrying.

Fourth, I would like to thank the 6.172 Performance Engineering of Software Systems course staff for invaluable experiences in how to be a good TA, especially for one of the toughest undergraduate courses at MIT. Typically, each MIT course has 12 units but the 6.172 course has 18 units, which means the workload for students as well as TAs is 50% higher than the typical courses. Dr. Tao B. (TB) Schardl, our co-instructor of the 6.172 course, was certainly also my role model in how much he cares about students. I also would like to give a special thank you to Grace Yin, one of the graduate TAs of the 6.172 course, for being such a supportive colleague.

Fifth, I would like to thank Dr. Jack Dunn, who has helped me so much technically since day 1 that I joined Prof. Bertsimas' research group. More importantly, my PhD thesis focuses on improving over the three types of decision trees from Interpretable AI software [51], which has been mainly developed by Jack since he was a PhD student at MIT Operations Research Center (ORC). Not only has Jack provided considerable support to me technically, he also provided genuine and sympathetic advice when I struggled to meet Prof. Bertsimas' expectations, and that advice has laid very firm ground in how to best prepare for each research meeting, and how to best communicate my work as well as ideas to Prof. Bertsimas.

Sixth, I would like to thank Georgios Antonios Margonis, MD, PhD, my first medical collaborator whom I worked with since the first research project with Prof. Bertsimas. Dr. Antonis is not only a skillful surgeon, but also very knowledgeable in the field of machine learning. Moreover, he is a great mentor. For all these reasons, we always have fruitful conversations on our healthcare projects, both over Zoom video calls and tons of emails. Prof. Bertsimas told us once that he stopped looking at our email correspondence that he was cc'ed, as he could not really catch up with our speed. Surprisingly, after collaborating online for more than 3 years, Dr. Antonis has been appointed to work in person with

Prof. Bertsimas, and his office is also in the MIT Sloan School of Management building. Our longest in-person conversation lasted for 4 hours, and it was a life changing four-hour conversation.

Seventh, I would like to express deep gratitude to MIT's Writing and Communication Center (WCC), and MIT Teaching and Learning Laboratory (TLL), which have significantly contributed to the writing quality of this thesis report, as well as my thesis presentation. Last summer, i.e., summer 2023, I was so desperate to improve my writing skill. Prof. Bertsimas commented on my first paper draft: "Yui! This is a B-level paper! I will not sign a thesis with this level of writing!" After spending approximately a month at WCC in fall 2023 with Pamela Siska, Robert (Bob) Irwin, and Elizabeth (Betsy) Fox, I met my PhD supervisor on October 2. Prof. Bertsimas said: "I think this paper is good enough to be submitted to the journal." I saw on the first page of my paper that he wrote "Good Job!" I was so overwhelmed and extremely happy, as I have never received this level of admiration from him before. Thalia Rubio is a WCC staff member who can always give positive feedback, no matter how poor the quality of the work is. I also met Christopher (Chris) Featherman, who is so great at reviewing presentations. I could also pass along his effective advice to my students in the 6.UAR Seminar in Undergraduate Advanced Research course. In addition, I joined TLL microteaching workshop in Fall 2023 that was moderated by Benjamin (Ben) Hansberry. The microteaching amazingly boosted my confidence in giving presentations.

Eighth, I would like to thank Prof. Leslie Kolodziejski, and Janet Fisher from MIT EECS Graduate Office. My PhD journey is definitely like a roller coaster. However, whenever I look around, Leslie and Janet are always by my side, and willing to provide the support I need. There were several times that they could withdraw me from being an EECS PhD student, given that I did not have satisfactory PhD progress. Fortunately, they believed in me and led me continue. Leslie holds very high positions in EECS, but she is really down to earth. Unfortunately, I also heard a sad news from Leslie that Janet would retire soon. Near the end of November 2023, I met Janet in person, and told her how my PhD journey had been. It is great to know from Janet and Leslie that I have become one of the success cases they are proud of. In addition, I would like to thank Alicia Duarte for thorough revision of my thesis report cover, as well as taking care of my thesis report submission.

Ninth, I would like to thank Melanie Parker, ex-Executive Director of MIT Career Advising and Professional Development (CAPD), and Blanche Staton, ex-Senior Associate Dean for Graduate Education. I met Melanie in September 2018 during CAPD appointment with the aim to consult about my job search. Melanie figured out that my ultimate goal was to continue in the MIT PhD program, but since I faced some difficulties, I also prepared for an alternative to finish with a master degree from MIT and start working. She connected me with Blanche, as she believed Blanche would definitely be able to give me helpful advice to continue my PhD status. Blanche was definitely helpful. She emphasized to me that please do not leave MIT before seeing her again.

Tenth, I would like to thank 6.UAR Seminar in Undergraduate Advanced Research course staff: Prof. Dina Katabi, Rebecca Thorndike-Breeze, Kristen Starkowski, Thomas Pickering, Sarah Bates, Dorothy Curtis, Rachida Kernis, and William (Bill) Tilden. Prof. Katabi is the main course instructor of 6.UAR, and also my EECS academic advisor. One of the main sources of my PhD funding has come from being a TA of 6.UAR, where I have TA-ed this course for 6 semesters. Rebecca is the main coordinator of the MIT's Writing, Rhetoric, and Professional Communication program (WRAP) instructors in this class. I met Rebecca during the first semester I TA-ed 6.UAR in spring 2019, and I had always noticed significant great improvement to the course curriculum whenever I returned as a TA. Kristen, Thomas, and Sarah were the other 3 WRAP instructors I had a chance to work closely with in this course. I was fortunate to co-grade students' homework with all of them, as they were very dedicated with the aim to help students learn. Dorothy, Rachida, and Bill are definitely amazing logistics team of the course. I am thrilled to know how much they care about students, and would always do their best to accommodate the needs of both students and other course staff.

Eleventh, I would like to thank Thai professionals, and Early Career Teachers' Network (ECTN) at Institute for Operations Research and the Management Sciences (INFORMS). I dare to ask them any questions about Operation Research, or INFORMS, including any potentially stupid questions. I told in the ECTN kick-off meeting that I realized about my dream job a bit late, and this is probably because I think I am not good enough to be a professor. Prof. Angelika Leskovskaya responded to my self-doubt by saying "Nobody should ever think that they are not good enough to do anything". Her words deeply empower me, and strengthen my spirit to fight for my dream. I also would like to give a special thank to my ECTN mentor, Prof. Jessica Leung, for being such a dedicated mentor and ECTN Co-Chairs, Prof. Nazli Turken and Prof. Gabriela (Gaby) Gongora-Svartzman, for their hard work in making ECTN 2023 realizable.

Twelfth, I would like to thank ORC alumni: Agni Orfanoudaki, Yuchen Wang, and Holly Wiberg. They help me better understand what my PhD supervisor looks for in each weekly meeting, as well as what might make him confuse in my prior meetings, and how I should fix them. Holly eventually became the first person at ORC I think about whenever I had a problem.

Thirteenth, I would like to thank my ORC TAs: Léonard Boussioux, Vassilis Digalakis, Cynthia Zheng, Zhen Lin, Adam Cheol Woo Kim, Kimberly Villalobos Carballo, and Theodoros Koukouvinos. They are my TAs for the 3 courses taught by my PhD supervisor: 15.095 Machine Learning Under a Modern Optimization Lens, 6.251 Introduction to Mathematical Programming, and 15.094 Robust Modeling, Optimization, and Computation. All of them were very helpful in helping me to reach the goal I aim for.

Fourteenth, I would like to thank my ORC friends: Yu Ma, Irra Na, Benjamin Boucher, Maura Hegarty, and Angelos Koulouras. They are among not so many ORC students whom I felt comfortable talking with. Yu was one of the very first friends I got to know at ORC. Irra had just graduated a few semesters before my thesis defense. Benjamin was my classmate for all the 3 courses I took at ORC. I worked with Maura and Angelos to apply my PhD

thesis on their healthcare datasets.

Fifteenth, I would like to thank Seehanah Tang, and Carol Gao. Seehanah had been my mentee since she was a high school student in summer 2020, i.e., a summer during a Covid outbreak. With her liveliness, and enthusiasm to learn, mentoring her was one of my favorite activities during the time of isolation. Seehanah also asked me once whether I would like to be a professor or not, as she thought I am well suited for this profession. Her words greatly sparked my joy, and strengthened my confidence that I can be a high-quality teacher. Carol is a friend of Seehanah, and we had a chance to cultivate our friendship when she started working as a predoctoral researcher at MIT. She also lives just a 5-minute walk from my dormitory, so there were many times I stop by to have meals, and fun as well as serious conversations with her.

Sixteenth, I would like to thank my ex-lab mates at the Model-Based Embedded and Robotics Systems (MERS) group: Yuening Zhang, Marlyse Reeves, Nick Pascucci, Sungkweon Hong, Meng Feng, Steve Levine, and Andrew Wang. Yuening was my first MIT friend. Marlyse and Nick joined MERS at the same time as me, i.e. at the beginning of fall 2017. Shortly after that, Sungkweon and Meng joined MERS near the end of fall 2017. Steve and Andrew are senior lab mates who joined MERS several years before me. All of them contribute to my memory at MERS, both sweet and bitter ones. They are definitely true friends who always gave me support, especially in the critical moments of desperation. Sungkweon is the one I met most frequently, as we became close friends since the first semester we met.

Seventeenth, I would like to thank my MIT EECS friends: Paul Zhang, Ting-An Lin, Prafull Sharma, and Martin Nisser. They all started at MIT at the same time as me. I knew Paul since the MIT orientation, and we became close friends. Paul, Sungkweon (my ex-lab mate), and I became a gang of three, and we often hung out together when time permitted. Ting-An only spent 5 years to finish her PhD, so I frequently asked her many administrative questions. Prafull was my classmate for a few courses at MIT. As we both aimed to get an A to fulfill EECS Technical Qualifying Evaluation (TQE), we frequently helped each other whenever things were too difficult. Martin is also a great peer whom I have met from time to time at MIT. In addition, I would like to thank my 6.336 Introduction to Modeling and Simulation teammates: Jason Zhang and Georgia Thomas. All three of us were like a dream team.

Eighteenth, I would like to thank my 15.390 new enterprises classmates: Michelle Ewy, Shao Lan Wang, and Anna Greenthal. When things became too tough for me, they all did not blame me in deciding to drop the class, but instead gave me spirit to fight and provided much life-changing advice. We are all female students with the same aim to build a start-up to empower women. It was amazing how we perfectly blended together as we all came from different age groups and education levels: PhD graduate and retired, PhD student, master student, and undergraduate student.

Nineteenth, I would like to thank the administrative assistants of my PhD supervisor, and my thesis committee: Stephanie Tran, Shyla Putrevu, and Katie O'Reilly. Without their help, the scheduling of my thesis committee meeting, as well as my thesis defense, would not have been possible.

Twentieth , I would like to thank my sailing crew, especially Astera Tang and Weerachai (Junior) Neeranartvong who were my classmates for the MIT Intermediate Sailing Physical Education (PE) class, and contributed a lot to my very first steps in sailing. I loved sailing so much, as it was great fun to control a sailboat along the Charles River, while enjoying great company with my passengers. I aimed to sail 100 times before graduating from MIT. However, reaching 52 times was not bad at all. My ex-roommate, Ogochukwu (Ogo) Belo-Osagie, also boarded my sailboat once. We both have grown up so much since the first time we met: I have just earned the title "Dr. Yui" while Ogo recently became a Chief Financial Officer (CFO) of Shell Energy in Nigeria!

Twenty-first, I would like to thank my professors from Imperial College London where I did my master's degree before joining MIT: Prof. Michael Huth, Prof. Philippa Gardner, and Prof. Fariba Sadri. Prof. Huth was my master's thesis supervisor, and also wrote a letter of recommendation for me when I applied to the MIT EECS PhD program. Prof. Gardner was an instructor of my Separation Logic class. Prof. Sadri was the head of the MSc program. All 3 of them helped me so much when I had academic difficulties while studying at Imperial College London. Without their kindness, I would not have been able to finish from Imperial, and get admitted to MIT. In addition, I would like to thank my Imperial classmates: Hadeel Al-Negheimish, Bob Webb, Michael Arcangeles, and Paweł Gomoluch. We still kept in touch, and I had a warm reunion when I transited in the UK in 2019 on my flight from Thailand to the US.

Twenty-second, I would like to thank my undergraduate professors from Mahidol University in Thailand: Prof. Supachai Tangwongsan and Prof. Vasaka Visoottiviseth. Both of them have provided me tremendous support since I was an undergraduate student under their supervision. They also contributed significantly to numerous letters of recommendation that I used to apply to graduate schools, and scholarships. More importantly, they were lights that lit my paths to move forward, whenever I was so desperate in figuring out the next step in my academic journey.

Twenty-third, I would like to thank my Thai friends: Penporn (Kaew) Koanantakool and Aniwat (Toey) Tiralap. I almost did not apply to MIT, as I felt the reply over email from my prospective MIT supervisor was not very strongly positive. However, Kaew convinced me that I should give it a try, and it turned out only MIT accepted me to their PhD program out of about 5 universities that I applied for. Toey was my first Thai friend at MIT, whom Kaew introduced to me. He helped me so much in settling down at MIT.

Twenty-fourth, I would like to thank all the Thai students and Thai communities whom I got a chance to know at some point during my 6.5 years at MIT since September 2017 for making my PhD journey full of joy, and not lonely. Speaking in my mother tongue language with people who truly understood my background, and the origin I came from, was unarguably a great relief.

Twenty-fifth, I would like to thank my Facebook friends who always gave valuable advice and feedback throughout my PhD journey. I love writing and I use my Facebook page as my personal diary. There are tremendous amount of support flowing as emoji reactions, comments, and personal messages. Even during many critical periods of my PhD journey, much helpful and cheerful advice has effectively brought me back into my path to graduation again. Many thanks to anyone who attended my thesis defenses, especially the ones stayed in Thailand as 2pm in Boston was 2am in Thailand, the time that you should be in bed.

Twenty-sixth, I would like to thank my cousin, Apinya (Toei) Bloodgood Harper, and her family: Matt, Trisa, and Jak Bloodgood Harper. It was a great relief to know that I have relatives living in the US. During the first time that we had a phone call, we kept talking for almost an hour. I had a chance to meet them all in person for the first time in the US in October 2023, when I joined the INFORMS annual meeting in Pheonix. They drove 7 hours from where they lived in Colorado to congratulate my significant milestone of my PhD study. I felt so grateful for what they did for me, and it was also the first time I met my niece, Trisa, and my nephew, Jak. They were so cute. Moreover, P' Toei and P' Matt were always the first persons who gave me Christmas gifts for the past 2 years.

Last but not least, I would like to thank my family for their tireless support. Everyone in my family, except me, is a medical professional: my dad is a pediatrician; my mom is a nurse; and my two sisters are dermatologists. I always told other people that I mutated from them. Now, I have realized that healthcare is possibly in my blood. After escaping from the medical world to study computer science, I eventually followed my intrinsic passion and decided to do my PhD research in healthcare. It was a bit challenging to explain to my family about why I would like to pursue a PhD, as none of them did a PhD. The most important thing for them is possibly to support me on the path I select. Dad, P' Jeans, and Yam, I am so excited that I will soon has the title "Doctor" like you have. Let me write everyone's name down here, as this is definitely a master piece in my academic journey: Supachai Sujijantararat, MD (my dad); Nattaya Sujichanthararat (my mom); Nitiporn Sujijuntararut, MD (my elder sister); and Atchareeya Sujichanthararat, MD (my younger sister).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Optimal Decision Trees with Local Search

Interpretability is a key factor for users to decide whether or not to adopt a particular machine learning model. It is unlikely for humans to trust a black-box model as there is no rationale to explain why it makes such predictions. A binary decision tree is one of the most interpretable machine learning models because it replicates how a person usually solves a problem by asking a set of binary questions. Each binary question is to ask whether or not a particular feature has a value less than a cutoff. If the answer is yes, the user navigates to the left subtree and reaches another binary question at another branch node. If the answer is no, the navigation will be to the right subtree. Interpretable AI [51] has provided state-of-the-art binary decision tree construction framework in classification, prescriptive, and survival analysis including Optimal Classification Tree (OCT) by Bertsimas and Dunn [10, 11], Optimal Policy Tree (OPT) by Amram et al. [4], and Optimal Survival Tree (OST) by Bertsimas et al. [16]. Even though OCT, OPT, and OST outperform other popular decision tree algorithms, they use local search which does not guarantee global optimality.

1.1.1 Optimal Classification Trees (OCT)

In order to overcome the greedy nature of CART [69], Bertsimas and Dunn [10] developed OCT by utilizing local search [54]. CART is grown using a top-down approach [32]. It starts with a single node denoted as a root node and each split is then selected as a feature and cutoff value that yield the best objective value. For classification problems, typical objective values include misclassification, Gini impurity, and entropy score [49, 91]. This greedy approach in CART construction can result in a suboptimal tree, because an early bad split which CART does not select could lead to a better subsequent split further down in the tree. By utilizing local search, OCT manages to resolve this issue in CART caused by the top-down approach.

With the aim to allow an early bad split which may lead to a better split in a classification tree, OCT starts with multiple random warm start trees [5]. To construct each warm start in OCT, each best split is determined from the random $\sqrt{p}$ features where p is a total number of features in a dataset. As illustrated by Bertsimas and Dunn [11], $\sqrt{p}$ features is the optimal number of features found from their empirical experiments to determine a split of each random warm start.

After initializing a classification tree with a random warm start, local search optimizes over all nodes in random order in each iteration until no improvement can be made. There are three options to optimize each node: updating the split by searching through all available features and cutoffs, replacing with the left subtree, and replacing with the right subtree. The option with the best objective value is selected. In order to optimize a leaf node, it could also spawn its left child and right child, which causes the tree to grow with more nodes. On the other hand, replacing a node to be optimized with its left or right subtree is similar to pruning a tree. If any iteration fails to further optimize any node in the tree or two consecutive iterations have the same objective value, the algorithm terminates for that warm start. In order to avoid any local minima, OCT has multiple random warm starts (100 trees by default) and claims that a global optimal solution could be found if the number of random warm starts exceeds the number of local minima. The final tree with the best objective values among all warm starts is the OCT.

As demonstrated in Bertsimas and Dunn [11], OCT successfully improves over CART on mean out-of-sample accuracy across all 61 real-world datasets from the UCI machine learning repository. From maximum depth 1 to 10, OCT outperforms CART at all depths with approximately 1% to 2% improvement and the improvement is proven to be statistically significant. In addition, OCT at only maximum depth 4 could achieve performance at a similar level as CART at depth 10, which emphasizes the better ability of OCT to capture significant splits for classification trees.

Gini impurity score is one type of widely-used objective values for a classification tree [73]. At each leaf node j, we calculate the Gini impurity score $gini_j$ as shown in Equation (1.1) where K is the total number of classes and $prob(i)$ is a probability that a sample belongs to class i:

$$gini_j = \sum_{i=1}^{K} prob(i)(1 - prob(i)). \tag{1.1}$$

Then we calculate the weighted Gini impurity score at each leaf node in Equation (1.2) by weighting the score over the number of samples in each leaf node. The $|node_j|$ is the number of samples at the leaf node j and n is the number of samples in the entire classification tree:

$$gini_{j,weighted} = \frac{|node_j|}{n} gini_j. \tag{1.2}$$

Finally, we sum across all weighted Gini impurity scores at each leaf node and adjust with the complexity of the tree to get a Gini impurity score of the entire classification tree. The calculation is as shown in Equation (1.3) where L is the number of all leaf nodes, γ is complexity penalty as determined by hyperparameter tuning, and B is complexity of the tree equals the number of branching nodes:

$$gini_{tree} = \sum_{j=1}^{L} gini_{j,weighted} + \gamma B. \tag{1.3}$$

1.1.2 Optimal Policy Trees (OPT)

OPT construction resembles that of OCT, so local search also limits OPT from reaching a global optimal solution. Similar to the OCT framework, OPT begins with random warm start solutions (100 random warm starts by default). At each iteration, OPT optimizes every node in a random order until no improvement can be made for two consecutive iterations. There are three options to optimize each node: updating a parallel split, replacing a split node with its left subtree, or replacing a split node with its right subtree. OPT selects the option with the best objective value.

One of the main differences between OPT and OCT is objective value calculation. OCT uses either Gini impurity, misclassification, or entropy score as its objective function, which are the typical evaluations for classification problems. At each leaf node of OCT, OCT predicts a class which gives either the optimal Gini impurity, misclassification, or entropy score. In contrast, OPT optimizes over rewards or outcomes which are specific to each problem. At each leaf node of OPT, OPT predicts a treatment which gives the optimal reward.

If not all treatment options were prescribed as observed treatments for each data point, a counterfactual model is constructed to predict counterfactual outcomes [68]. OPT provides Random Forest [18] and XGBoost [21] as options for counterfactual models. These counterfactual models can utilize either the direct method or the doubly robust method [8] from Interpretable AI [51] software to estimate rewards as shown in Equations (1.4) and (1.5):

$$\Gamma_i(t) = f_t(x_i). \tag{1.4}$$

$$\Gamma_i(t) = f_t(x_i) + I\{T_i = t\} \cdot \frac{1}{p(x_i, t)}(y_i - f_t(x_i)), \tag{1.5}$$

where

- $\Gamma_i(t)$ denotes the estimated rewards of the data point i under treatment t
- $f_t(x_i)$ denotes the model for each treatment t to predict y as a function of x
- x_i denotes the vector of observed features of the data point i
- $I\{T_i = t\}$ denotes the indicator function (1 if $T_i = t$ is true and 0 otherwise)
- T_i denotes the observed treatment of the data point i
- $p(x, t)$ denotes the model to predict propensity scores [7] as the probabilities that a sample with features x is assigned treatment t
- y_i denotes the observed outcome of the data point i (corresponding to treatment T_i)

The propensity score is what makes the doubly robust method different from the direct method. As you compare Equation (1.4) with Equation (1.5), the term involving propensity only presents in Equation (1.5) which corresponds to the doubly robust method. The doubly robust method utilizes propensity scores to remove bias from treatment assignments so that the treatment assignments resemble that of the randomized control trials [86].

1.1.3 Optimal Survival Trees (OST)

OST construction resembles that of OCT so local search also limits OST from reaching a global optimal solution. Similar to OCT framework, OST begins with random warm start solutions (100 random warm starts by default). At each iteration, OST optimizes every node in a random order until no improvement can be made for two consecutive iterations. There are three options to optimize each node: updating a parallel split, replacing a split node with its left subtree, or replacing a split node with its right subtree where OST selects an option resulting in the best objective value.

One of the main differences between OST and OCT is objective value calculation. OCT uses either Gini impurity, misclassification or entropy score as its objective function which are the typical evaluations for classification problems. At each leaf node of OCT, OCT makes a prediction of class that gives the optimal Gini impurity, misclassification or entropy score. In contrast, OST optimizes over a local full likelihood score which involves cumulative hazard function with the Nelson-Aalen estimator [1, 71, 72]. At each leaf node of OST, it predicts a survival time which gives the best local full likelihood score. The curves in each OST node are the Kaplan-Meier curves [55] which estimate the survival distribution.

We calculate the survival distribution of each observed data point i as a function of the baseline cumulative hazard function $\Lambda(t_i)$ and an adjustment to the baseline cumulative hazard function θ_i of each data point i as shown in Equation (1.6). Then we measure the log-likelihood of this survival function against the data as shown in Equation (1.7).

$$P(S_i \leq t) = 1 - e^{\theta_i \Lambda(t)}, \tag{1.6}$$

$$error = \sum_{i=1}^{n} w_i(\Lambda(t_i)\theta_i - \delta_i[log(\Lambda(t_i)) + log(\theta_i) + 1]), \tag{1.7}$$

where

- S_i denotes survival time of the data point i.
- w_i denotes sample weight assigned to the data points i.
- t_i denotes the last observation of the data point i.
- $\delta_i \in \{0, 1\}$ denotes the indicator of the last observation of the data point i whether it was a death ($\delta_i = 1$) or a censoring ($\delta_i = 0$).

- $\Lambda(t)$ denotes the cumulative hazard function of the training data calculated with the Nelson-Aalen estimator.
- θ_i denotes the fitted hazard coefficient for the data point i.

1.2 Optimal Decision Trees with Simulated Annealing

In optimization, the simulated annealing process is similar to the reformulation of the initial warm start solution until reaching the global optimal solution at the final lowest temperature [58]. The initial heating breaks the structure of the initial warm start solution so that its components can move anywhere. Then, at the cooling step, the temperature starts decreasing and the solution starts to form its gross structure. The objective value of the optimization problem is similar to the energy of the metallic annealing that we aim to minimize. The probability to accept a worse solution is higher at high temperature and gradually decreases as the temperature decreases. This is similar to the metallic atoms having higher degrees of freedom in their movements at a higher temperature. Simulated annealing is one type of heuristic approach [13]. Generally, a heuristic approach can find a good quality suboptimal solution rapidly even though its runtime is not necessarily polynomial. Different problem types also require different heating and cooling procedures and this is called an annealing schedule or a cooling schedule.

The main difference of each simulated annealing cooling schedule is how the temperature decreases. Some of the common annealing schedules include geometric, logarithmic, and linear annealing schedules [74]. By defining the initial temperature as $temp_0$, the temperature at each iteration k of each annealing schedule can be defined as Equations (1.8), (1.9), and (1.10) respectively:

$$temp_k = \alpha^k \cdot temp_0, \tag{1.8}$$

$$temp_k = \frac{temp_0}{1 + \alpha \cdot \log(1 + k)}, \tag{1.9}$$

$$temp_k = temp_0 - \alpha \cdot k. \tag{1.10}$$

The typical range of values for α which determines the rate of temperature decay is different for each type of cooling schedule.

1.2.1 Optimal Classification Trees with Simulated Annealing (OCT-SA)

By replacing local search in OCT with simulated annealing, we obtain OCT-SA. Unlike local search that always transforms a current subtree into a neighoring subtree with a better objective value, simulated annealing probabilistically allows a worse transformation. We calculate the probability of such transformations based on the current temperature as determined by the simulated annealing cooling schedule, and the difference in energy, i.e., the objective value, between the current and the neighboring subtree. The appearance of OCT-SA still resembles that of OCT where they are both binary decision trees that predict classes of each data point at leaf nodes. A branching node splits on a particular feature and cutoff value. Each split divides data into 2 subtrees: the left subtree containing data points with feature values less than the cutoff, and the right subtree containing data points with feature values more than or equal to the cutoff. At each leaf node, OCT-SA makes a prediction on a class of data points in that leaf node that results in the best objective value, i.e., Gini impurity score for classfication analysis.

1.2.2 Optimal Policy Trees with Simulated Annealing (OPT-SA)

In order to construct OPT-SA, we change local search in OPT to simulated annealing. Similar to OCT-SA, simulated annealing in OPT-SA permits a bad transformation from a current state to a neighboring state. The main difference between OCT-SA and OPT-SA is the objective value where OCT-SA uses the Gini impurity score while OPT-SA uses the mean rewards score. At each leaf node, OCT-SA makes a class prediction while OPT-SA makes a prescription prediction, i.e., an optimal treatment resulting in the best mean rewards score or the outcomes. In case not all treatment options of interest were prescribed to every subject in the observed dataset, we need to construct counterfactual models to predict counterfactual outcomes. The mechanism we use to construct counterfactual models in OPT-SA stay unchanged from that of OPT. In order to utilize the same simulated annealing framework as that of OCT-SA, we need to scale mean rewards, whose magnitudes vary depending on types of datasets, to be in the same range as the Gini impurity score in OCT-SA, i.e., from 0 to 1. The display of OPT-SA is still the same as that of OPT where both of them are binary decision trees that predict an optimal treatment at each leaf node. However, simulated annealing can drive the tree transformation for OPT and OPT-SA to have a different final structure like a different number of nodes, or a different set of selected features and their cutoff values.

1.2.3 Optimal Survival Trees with Simulated Annealing (OST-SA)

OST-SA improves on OST by swapping local search with simulated annealing. Like that of OCT-SA and OPT-SA, simulated annealing can improve objective values of OST-SA by granting some suboptimal tranformations. In OST-SA, the process that we use to estimate a cumulative hazard function with the Nelson-Aalen estimator for each data point and approximate the survival distribution by fitting a Kaplan-Meier curve at each leaf node stays the same as that of OST. Similar to OPT-SA, OST-SA needs to scale the objective value, i.e., the local full likelihood score, to be in the range from 0 to 1 like the objective value of OCT-SA. The presentation of OST-SA is still the same as OST where they both represent binary decision trees that predict survival time, or time to event of interest, of data points at each leaf node.

1.3 Main Contributions

We make three main contributions in advancing the development of interpretable machine learning models for classification, prescriptive, and survival analysis. We implement our optimal decision trees with a simulated annealing framework, then evaluate it on widely-used benchmarking real-world datasets, and showcase our applications in the medical domain.

1.3.1 Optimal Decision Trees with Simulated Annealing

We develop an optimal decision trees framework by replacing local search with simulated annealing. This framework is highly generalizable and can be applied to all the three analysis domains: classification, prescriptive, and survival analysis. For classification analysis, we can alter OCT into OCT-SA. In terms of prescriptive analysis, we can adjust OPT into OPT-SA using the same simulated annealing framework with some additional arguments for reward estimations. Regarding survival analysis, this simulated annealing framework is also applicable to the change from OST into OST-SA by incorporating some additional inputs for hazard coefficients.

1.3.2 Evaluation on Widely-Used Benchmarking Real-World Datasets

We evaluate our simulated annealing optimal decision trees on widely-used benchmarking real-world datasets. The evaluations demonstrate our success in achieving better out-of-sample performance over other popular decision tree models with a greedy approach. First, we evaluate OCT-SA on 61 real-world datasets from the UCI machine learning repository [6]. Second, we evaluate OPT-SA on 10 real-world datasets from both the UCI machine learning repository and Kaggle. Third, we evaluate OST-SA on 10 real-world datasets from SurvSet, an open-source time-to-event dataset repository [29].

1.3.3 Case Studies on Real-World Medical Datasets

In order to showcase our optimal decision trees with simulated annealing in the medical domain, we apply our models on all three type of analyses (classification, prescriptive, and survival) over real-world medical datasets. First, we apply OCT-SA in the sarcoma dataset. Second, we apply OPT-SA in the Gastrointestinal Stromal Tumor (GIST) dataset. Third, we apply OST-SA in the sarcoma dataset, the same dataset as that of OCT-SA.

1.4 Structure of the Thesis

This thesis is organized as follows.

Chapter 2: We demonstrate six main adjustments on Interpretable AI [51] optimal decision trees framework when replacing local search with simulated annealing: random warm start construction, objective value calculation, geometric cooling schedule, transformation of a tree to a neighbor state, hyperparameter tuning, and terminating condition.

Chapter 3: We point out limitations of OCT and how we incorporate simulated annealing in OCT-SA construction to further improve on OCT. We evaluate OCT-SA on 61 real-world datasets from the UCI machine learning data repositories.

Chapter 4: We show the weak points of OPT and explain how to improve OPT by swapping local search with a simulated annealing framework in order to build OPT-SA. We evaluate OPT-SA on 10 real-world datasets from the UCI machine learning data repositories and Kaggle.

Chapter 5: We indicate drawbacks of OST and exhibit the method to utilize simulated annealing in place of local search to obtain OST-SA which outperforms OST. We evaluate OST-SA on 10 real-world datasets from SurvSet, an open-source time-to-event dataset repository.

Chapter 6: We showcase our simulated annealing optimal decision trees on real-world medical datasets: OCT-SA on the sarcoma dataset, OPT-SA on the Gastrointestinal Stromal

Tumor (GIST) dataset, and OST-SA on the same sarcoma dataset as that of the OCT-SA.

Chapter 7: We offer our concluding remarks on how simulated annealing can benefit the decision tree construction process and improve optimality for classification, prescriptive, and survival analysis.

Chapter 2

Optimal Decision Trees with Simulated Annealing

There are five main adjustments we have made on the Interpretable AI [51] optimal decision trees (OT) framework, which utilizes local search, in order to apply simulated annealing and obtain optimal decision trees with simulated annealing (OT-SA): random warm start construction, geometric cooling schedule, transformation of a tree to a neighbor state, hyperparameter tuning, and terminating condition.

2.1 Random Warm Start Construction

The only difference between random warm start construction of OT-SA than that of the OT framework is how we determine a split at the root node. In OT-SA, we evenly assign each feature as the first split at root node for all warm starts. For example, let a dataset have $p = 5$ features. With 100 warm starts, there will be p or 5 different first split features where each of them is assigned to $\frac{100}{p} = \frac{100}{5} = 20$ warm starts. For the same first split feature, the best cutoff is also the same as it is determined from the same set of all data points at the root node. The reason that we generate warm starts in this manner is because the split at root node can have a significant influence on the tree structure in subsequent depths. When we first tried generating warm starts with the same procedure as that of OT, we found that our warm starts of some datasets lack variety and it was likely because the split at root node always belongs to a small subset of best features. We still construct the splits at subsequent depths using a similar procedure as that of OT where each best split is derived from a subset of random $\sqrt{p}$ features where p equals the number of all features.

We also made use of multiple warm starts in order to distribute the search over the entire search space like that of OT. Since the search over each warm start is independent, we can perform multiple searches in parallel in order to reduce runtime. In addition to these 100 random warm starts, we also used an OT model (either OCT, OPT, or OST), a model a widely-used Python package (CART, or sksurv survival tree [80]) and a single root node as 3 additional warm starts, i.e., 103 warm starts in total.

2.2 Geometric Cooling Schedule

Among the three popular simulated annealing cooling schedules: geometric, logarithmic, and linear, we found from our empirical experiments that the geometric cooling schedule fits best with the OT-SA construction. With the geometric cooling schedule, the temperature drops fast at the beginning of the schedule. This is appropriate for the OT-SA algorithm that starts with a random warm start at the beginning because it does not need that many iterations to reformulate the tree into a random structure at high temperature. Then the temperature drops at a slower and slower rate. This matches well the case when more selective transformation of the tree is preferred near the end of the cooling schedule. Regarding the logarithm cooling schedule, it requires much longer runtime than the geometric cooling schedule as it takes many more iterations to reach the final temperature and this is a significant downside. In terms of the linear cooling schedule, the temperature linearly drops and this results in the algorithm spending too many iterations at the high temperature. Consequently, the initial tree may be transformed into a totally random tree while the random warm start tree already has some nice decision power that OT-SA should maintain. In addition, spending too few iterations at the low temperature also prevents the tree from completely enhancing its tree structure.

The mechanism of simulated annealing algorithm is driven by the geometric cooling schedule, the difference in energy, and the acceptance probability as shown in Equations (2.1), (2.2), and (2.3):

$$temp_{k+1} = \alpha \cdot temp_k, \tag{2.1}$$

$$\Delta E = |score_{current} - score_{neighbor}|, \tag{2.2}$$

$$prob_k = e^{\frac{-\Delta E}{temp_k}}. \tag{2.3}$$

where

- *temp* denotes temperature
- k denotes iteration number
- α denotes temperature decay rate
- ΔE denotes difference in energy between a current and a neighbor state
- *score* denotes objective value score
- *prob* denotes probability to transform a tree from a current state to a neighbor state
- e denotes the Euler's number equals 2.7183

From our empirical experiments, we found $\alpha = 0.95$ is an appropriate temperature decay rate for the real-world datasets we used to evaluate OT-SA.

2.3 Transformation of a Tree to a Neighbor State

OT-SA allows a tree to move to a worse neighbor state with some probability as shown in Equation (2.3). If a neighbor state has a lower objective value score than that of the current state, OT-SA always transforms the tree into a better neighbor state. Otherwise, the probability to accept a worse neighbor state depends on the current temperature as shown in Equation (2.1) and how much the objective value score of the neighbor state is worse than that of the current state as shown in Equation (2.2). OT-SA will then compare this probability with a random number in the range (0.1, 1). If the probability is more than or equals to the random number, this worse neighbor state is accepted. Otherwise, the tree stays unchanged.

2.4 Hyperparameter Tuning

When applying simulated annealing, each dataset may require different Markov chain length l or number of nodes to optimize per iteration, neighbor search radius r or boundary of solution search space, and complexity penalty γ. This is due to differences in complexity of each problem and how far an optimal solution is from a random warm start. The more complex and deeper tree depths tend to require a longer Markov chain length and search radius. In OT, all nodes are optimized per iteration which is equivalent to Markov chain length l equals the total number of nodes in the tree. However, we found that this mechanism of OT does not work well in OT-SA. The reason that OT-SA does not optimize all nodes at the same temperature, as in OT, because optimizing a few nodes can significantly change a tree structure, so OT-SA should use a new threshold for acceptance probability of the next temperature level when optimizing the subsequent random nodes. In terms of neighbor search radius r, OT-SA always selects the best neighbor and this is equivalent to search radius $r = 1$ which is against the mechanism of simulated annealing where some worse transformations are possible under some probability.

To find the best hyperparameters of each dataset, we perform grid search over six combinations of hyperparameters with Markov chain length l equals 1, 2 or 3 and search radius r equals 3 or 4. For each combination of Markov chain length l and neighbor search radius r, we run model training with 100 random warm starts (i.e., excluding OT, other Python decision tree, and a single root node) over a training dataset and perform the batch cost complexity pruning algorithm [30] which outputs an optimal complexity penalty γ that gives the best validation score for each dataset. Then we use these tuned hyperparameters to rerun model training over 103 warm starts (i.e., including OT, other Python decision tree, and a single root node in addition to the 100 random warm starts) with a combined training and validation dataset. The final tree with the best score over the combination of training and validation dataset is the OT-SA. From our empirical experiments over real-world datasets we use to evaluate OT-SA, we find that Markov chain length $l = 2$ and search radius $r = 3$ perform best by average.

Another difference in hyperparameter tuning is how OT and OT-SA determine a final depth of a tree. In OT construction, the best depth is figured out during hyperparameter tuning where the best depth gives a tree with the best validation score. Then the tuned hyperparameters are used by OT algorithm to train the trees over the combination of train and validation dataset with no depth constraint but the tree is finally pruned to the tuned best depth. On the other hand, OT-SA does not find the best depth during hyperparameter tuning but always constraints a tree initialization and transformation to not exceed the specified maximum depth during training so OT-SA does not need to conduct post-pruning like OT. We decide to use this approach in order for OT-SA algorithm to have a more thorough search over search space of tree transformations at a particular depth.

2.5 Terminating Condition

OT-SA stops when it meets either one of these two conditions. First, the algorithm reaches the end of the cooling schedule. This is to avoid overfitting and to limit the runtime. Second, the algorithm cannot optimize any node at the same temperature. For the lower temperature, OT-SA reduces the probability to accept worse states, so it is less likely that it will optimize any node. OT-SA also keeps track of the tree with the best objective value which is not necessarily the tree obtained from the last iteration of simulated annealing. This is to handle the case when transformations at lower temperatures make a tree diverge from global optimality and the best tree is found at the higher temperature.

Chapter 3

Optimal Classification Trees with Simulated Annealing (OCT-SA)

To improve on predictive performance of CART (Classification And Regression Trees), Bertsimas and Dunn [10] developed Optimal Classification Trees (OCT). OCT utilizes multiple random warm starts and applies local search to iteratively improve each warm start until a locally optimal classification tree is found. While OCT improves on CART, local search does not guarantee global optimality. Hence, we propose a framework called Optimal Classification Trees with Simulated Annealing (OCT-SA). OCT-SA utilizes simulated annealing to further improve OCT by probabilistically allowing a tree to move to a tree with a worse objective value. This mechanism is beneficial as some worse transformations could lead to a better final model. A challenge in designing OCT-SA is to define an appropriate simulated annealing cooling schedule that leads to a global optimal solution in practical runtime. We report computational results on 61 real-world datasets from the UCI machine learning repository. We train OCT-SA, OCT and CART over the maximum depths ranging from 2 to 7. OCT-SA successfully improves over OCT for all maximum depths. This improvement is statistically significant with p-values ≤ 0.1 at all maximum depths except depth 2. The improvement is still statistically significant with p-values ≤ 0.05 at maximum depth 3, 4 and 7. Furthermore, OCT-SA provides significant improvement over CART at all maximum depths. At the maximum depth 7, OCT-SA improves the average out-of-sample Gini impurity score on the 61 datasets by approximately 0.7% (0.2573 vs 0.2643) compared to OCT and 2.13% (0.2573 vs 0.2786) compared to CART. Although improvement was not found for all 61 datasets, OCT-SA achieves better out-of-sample Gini impurity score on 43 out of 61 datasets compared to OCT and also 43 out of 61 datasets compared to CART.

3.1 Introduction

Breiman et al. [17] developed Classification and Regression Trees (CART), a classical classification model which is a very interpretable model [19]. CART replicates how a person usually solves a problem by asking a set of binary questions. Each binary question asks whether or not a particular feature has a value less than a cutoff. If the answer is yes, the user navigates to the left subtree and reaches another binary question at another branch node. If the answer is no, the navigation will be to the right subtree. However, predictive performance of CART is not optimal due to its greedy approach where it always selects the best split. To elaborate, some early bad splits may lead to better subsequent splits of decision trees. Similarly, the other widely used classical classification models like C4.5 [82] and ID3 [81] also fail to reach optimality due to their greedy approach. The usage of the greedy approach is one way to construct a binary classification tree in practical time, since to construct an optimal binary decision tree is an $\mathcal{NP}$-hard problem [61]. Even though some ensemble tree models like Random Forests [18] and XGBoost [21] improve over CART on predictive performance, they are not interpretable and are usually viewed as black-box models. With the aim to construct an interpretable model with higher predictive performance in tractable time, Bertsimas and Dunn [10, 11] developed Optimal Classification Tree or OCT.

OCT achieved a better accuracy than that of CART on 61 datasets from the UCI machine learning repository [6]. Bertsimas and Dunn initially formulated OCT construction as a

Mixed Integer Optimization (MIO) problem but later adjusted their approach into local search for scaling purposes. Since OCT utilizes local search which always drives the tree transformation in the direction where objective value is improved, this approach could result in a suboptimal solution. The authors argued that by using a large enough number of random warm starts, the algorithm could effectively escape the majority of local minima. However, we typically do not know an exact number of local minima of each optimization problem. Hence, we propose a framework utilizing simulated annealing to construct optimal classification trees called Optimal Classification Trees with Simulated Annealing (OCT-SA). Given the same number of random warm starts, OCT-SA has a potential to better escape local minima than OCT.

Simulated annealing is an optimization technique which helps a model to reach a global optimal solution with some probability [see 12, 58]. Unlike local search, simulated annealing allows a tree to go to neighboring states with worse objective values. Under an appropriate cooling schedule, simulated annealing is guaranteed to find an optimal solution asymptotically. The objective of this paper is to see whether simulated annealing helps improve optimality or not. To fulfill this objective, we develop OCT-SA which makes use of a geometric cooling schedule. We also utilize multiple random warm starts similar to OCT while also adding CART, OCT, and a single root node as three additional warm starts. To adjust OCT-SA to fit a particular classification problem, we tune three hyperparameters: Markov chain length, search radius, and complexity penalty.

We make two main contributions in this chapter. First, OCT-SA is the first application of a simulated annealing algorithm on classification trees with parallel splits of sizable problems that guarantees to provide an optimal solution under an appropriate cooling schedule. Even though Bucy and Diesposti [20] and Lutsko and Kuijpers [65] made initial attempts to apply simulated annealing to classification trees, they only succeeded on small problems. Second, we report computational results on the same 61 datasets from Bertsimas and Dunn [11] that improve upon the out-of-sample performance of OCT while maintaining interpretability of each model.

This paper is organized as follows. In Section 3.2, we demonstrate how to incorporate simulated annealing into OCT construction. Then in Section 3.3, we evaluate OCT-SA on 61 real-world datasets on Gini impurity score and compare the performance of OCT-SA with OCT and CART. In Section 3.4, we summarize our findings.

3.2 Algorithms

OCT-SA further improves performance of OCT by replacing local search with simulated annealing. There are five algorithms involving in OCT-SA construction as shown in Figure 3.1. The entire flow of OCT-SA is described in Algorithm 1, which accepts five inputs: an initial classification tree T_1 as a random warm start tree; training data consisting of feature value X and outcome class y; geometric temperature decay rate $c < 1$ of a cooling schedule; Markov chain length l or number of nodes to optimize per iteration; and neighbor search radius r or boundary of solution search space, then outputs the optimal classification tree as T_{best}, which is an optimal transformation of the initial classification tree T_1 with simulated annealing. Algorithm 1 calls Algorithm 2 to rank and select a tree transformation from candidate trees. Algorithm 2 determines the quality of each tree transformation by gradually picking a better candidate tree as temperature in the geometric cooling schedule decreases. Algorithm 3 generates these candidate trees. There are three options to generate candidate trees by updating the current subtree root node: by replacing with the left subtree, by replacing with the right subtree, and by updating a parallel split. In order to update a parallel split, Algorithm 3 calls Algorithm 4 to generate options to transform the current subtree by updating the branching at the subtree root node with different features and cutoff values. Finally, Algorithm 1 calls Algorithm 5 to determine a probability to transform the current subtree into the neighbor subtree based on their Gini impurity scores and the current temperature in the annealing schedule.

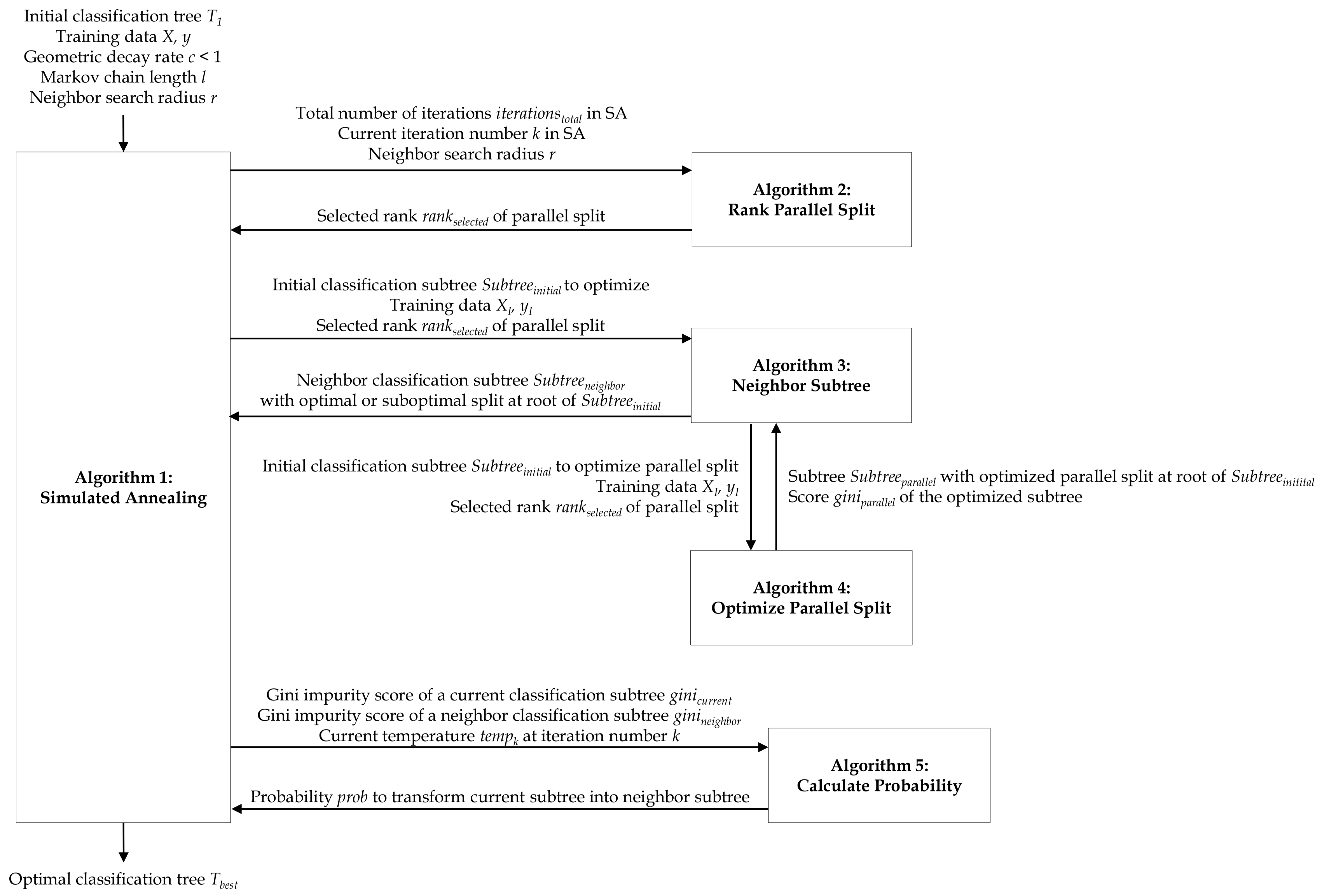
Initial classification tree T_1
Training data X, y
Geometric decay rate $c < 1$
Markov chain length l
Neighbor search radius r
Algorithm 1: Simulated Annealing
Total number of iterations $iterations_{total}$ in SA
Current iteration number k in SA
Neighbor search radius r
Algorithm 2: Rank Parallel Split
Selected rank $rank_{selected}$ of parallel split
Initial classification subtree $Subtree_{initial}$ to optimize
Training data X_I, y_I
Selected rank $rank_{selected}$ of parallel split
Algorithm 3: Neighbor Subtree
Neighbor classification subtree $Subtree_{neighbor}$ with optimal or suboptimal split at root of $Subtree_{initial}$
Initial classification subtree $Subtree_{initial}$ to optimize parallel split
Training data X_I, y_I
Selected rank $rank_{selected}$ of parallel split
Subtree $Subtree_{parallel}$ with optimized parallel split at root of $Subtree_{initital}$
Score $gini_{parallel}$ of the optimized subtree
Algorithm 4: Optimize Parallel Split
Gini impurity score of a current classification subtree $gini_{current}$
Gini impurity score of a neighbor classification subtree $gini_{neighbor}$
Current temperature $temp_k$ at iteration number k
Algorithm 5: Calculate Probability
Probability $prob$ to transform current subtree into neighbor subtree
Optimal classification tree T_{best}

Figure 3.1: OCT-SA flowchart of Algorithm 1 to 5 demonstrating inputs, outputs, and sequences of execution.

3.2.1 Overall Architecture of OCT-SA in Algorithm 1: Simulated Annealing

In Algorithm 1, we iteratively optimize the initial tree structure T_1 until it reaches the final tree structure T_{best}. The number of iterations to optimize the tree equals $iterations_{total}$, which is the number of iterations required for the initial temperature starting at 1 to be decreased with geometric decay rate $c < 1$ until the final temperature goes below 0.01. At each iteration k, we transform the tree into an intermediate tree T_k by optimizing over Markov chain length l nodes. Among all possible moves or neighboring trees, the tree does not necessarily move to the tree with the best objective value but possibly to the next subsequent best trees within the neighbor search radius r.

We implemented OCT-SA in Python version 3.8.15 and constructed OCT warm start solutions using an OCT package of Interpretable AI software [51] version 3.2.0 in Julia version 1.9.0. After that, we exported those warm start solutions into text files and imported them into OCT-SA. We ran OCT-SA on a Windows laptop and also on MIT Supercloud [84]. We construct the CART warm start solutions with scikit-learn Python package [78].

3.2.2 Algorithm 2: Rank Parallel Split

Algorithm 2 ranks and selects a tree transformation from candidate trees. The algorithm accepts three inputs: the total number of iterations $iterations_{total}$ that OCT-SA will optimize a tree; a current iteration number k; and a neighbor search radius r (which is determined from hyperparameter tuning); and outputs a select rank (based on objective value) of a neighbor subtree that OCT-SA will transform the current tree into. Instead of selecting the candidate tree with the best objective value, as in OCT, OCT-SA selects the next subsequent best tree from neighboring trees based on current iteration number k and neighbor search radius r. In case of neighbor search radius $r = 3$, the third best tree is selected for the first iteration. As the iteration number increases, the next best tree is selected, i.e., the second

Algorithm 1 SimulatedAnnealing

```
    Input: Initial classification tree $T_1$; Training data $X, y$;
           Geometric decay rate $c < 1$; Markov chain length $l$; Neighbor search radius $r$
    Output: Optimal classification tree $T_{best}$
 1: $temp_0 \leftarrow 1$                                         ▷ $temp_0$ is initial temperature
 2: $iterations_{total} \leftarrow k_{min} - 1$ where $temp_0 \cdot c^k < 0.01$
 3: $k \leftarrow 1$                                              ▷ k is iteration number
 4: $T_{best} \leftarrow T_1$                                     ▷ $T_{best}$ keeps the best tree
 5: repeat
 6:     $temp_k \leftarrow temp_0 \cdot c^k$           ▷ Temperature based on geometric cooling schedule
 7:     $rank_k \leftarrow$ RankParallelSplit($iterations_{total}, k, r$)
 8:     $nodes_{optimized} \leftarrow \emptyset$       ▷ Keep track of optimized nodes at current temperature
        ▷ Get a set of all current split nodes or leaf nodes in $T_k$ eligible to be split further
 9:     $nodes_{unoptimized} \leftarrow$ SplitNodes($T_k, X, y$)
10:     repeat
11:         $nodes_{unoptimized} \leftarrow nodes_{unoptimized} - nodes_{optimized}$
12:         $node_{random} \leftarrow$ RandomNode($nodes_{unoptimized}$)        ▷ Pick one random node
13:         $nodes_{unoptimized} \leftarrow nodes_{unoptimized} - \{node_{random}\}$
14:         $nodes_{optimized} \leftarrow nodes_{optimized} \cup \{node_{random}\}$

            ▷ Find subtree of the random node and calculate score
15:         $Subtree_{random} \leftarrow$ Subtree of $T_k$ where $node_{random}$ is root
16:         $I \leftarrow \{i\colon x_i$ is assigned to a leaf contained in $Subtree_{random}\}$
17:         $gini_{current} \leftarrow$ GiniImpurity($Subtree_{random}, X_I, y_I$)
            ▷ Find a neighbor of the random subtree and calculate score
18:         $NeighborSubtree_{random} \leftarrow$ NeighborSubtree($Subtree_{random}, X_I, y_I, rank_k$)
19:         $gini_{neighbor} \leftarrow$ GiniImpurity($NeighborSubtree_{random}, X_I, y_I$)

            ▷ Determine probability to transform the current subtree to its neighbor state
20:         $prob \leftarrow$ CalculateProbability($gini_{current}, gini_{neighbor}, temp_k$)
21:         if $prob \geq$ RandomNumber($0.1, 1$) then
22:             $T_k \leftarrow$ Replace $Subtree_{random}$ in $T_k$ with $NeighborSubtree_{random}$
23:             if GiniImpurity($T_k, X, y$) < GiniImpurity($T_{best}, X, y$) then
24:                 $T_{best} \leftarrow T_k$          ▷ Update the best tree if Gini impurity score is lower
25:             end if
26:         end if
27:     until $nodes_{unoptimized} = \emptyset$
28:     if $|nodes_{optimized}| = 0$ then        ▷ No node can be optimized at current temperature
        ▷ Move to the last iteration of the current temperature band
29:         $k \leftarrow k_{max}$ where RankParallelSplit($iterations_{total}, k, r$) returns $rank_k$
30:     end if
31:     $k \leftarrow k + 1$
32: until $k = iterations_{total}$
33: return $T_{best}$
```

Algorithm 2 RankParallelSplit

```
Input: Total number of iterations $iterations_{total}$ in simulated annealing;
       Current iteration number $k$ in simulated annealing;
       Neighbor search radius $r$
Output: Selected rank $rank_{selected}$ of parallel split to be used in Algorithm ??

1: $rank_{selected} \leftarrow 1$                                  ▷ 1 is the best rank
2: $rank_{set} \leftarrow \{r, r-1, r-2, ..., 1\}$    ▷ Set of all ranks in neighbor search radius $r$
3: for $i$ in $rank_{set}$ do
4:     if $k < \frac{iterations_{total}}{i}$ then
5:         $rank_{selected} \leftarrow i$
6:         break
7:     end if
8: end for
9: return $rank_{selected}$
```

best. Eventually, OCT-SA selects the first best tree near the end of the cooling schedule. OCT-SA selects the worse candidates at early temperature bands in order to form a gross structure of an optimal tree. This could lead to better states at subsequent temperature bands. In addition, the candidate ranking could help prevent overshooting. Randomly picking a candidate from the search neighborhood may make the algorithm diverge away from the optimal solution.

The implementation of candidate state ranking is outlined in Algorithm 2. Instead of randomly picking a state from our searched neighborhood, we select a candidate state based on their Gini impurity score ranking. We define neighbor search radius r where we select the r^{th} best state transition at the highest temperature while we select the 1st rank or the best state with the lowest Gini impurity score at the lowest temperature. With search radius r, the temperatures are divided into r bands. If our simulated algorithm runs for S iterations in total, the end of each temperature band is at iteration number $\frac{S}{r}, \frac{S}{r-1}, \frac{S}{r-2}, ..., \frac{S}{2}, S$. To illustrate, assume OCT-SA runs for $S = 90$ iterations with search radius $r = 3$. We stratify the temperatures into 3 bands which cover these ranges of iterations and selected candidate rank: 1 - 30 (3rd rank), 31 - 45 (2nd rank), and 46 - 90 (1st rank). By selecting the 1st rank candidate at the last temperature band, the algorithm performs similarly to the local search heuristic. This is appropriate for the low temperature of simulated annealing where OCT-SA hardly accepts worse solutions.

3.2.3 Algorithm 3: Neighbor Subtree

Algorithm 3 NeighborSubtree

```
Input: Initial classification subtree $Subtree_{initial}$ to optimize;
       Training data $X_I, y_I$;
       Selected rank $rank_{selected}$ of parallel split
Output: Neighbor classification subtree $Subtree_{neighbor}$ with optimal or suboptimal split at root of $Subtree_{initial}$

    ▷ Optimize parallel split of initial subtree and calculate score
 1: $Subtree_{parallel}, gini_{parallel} \leftarrow$ OptimizeParallelSplit($Subtree_{initial}, X_I, y_I, rank_{selected}$)

    ▷ 1 is the worst Gini impurity score
 2: $gini_{lower} \leftarrow 1$
 3: $gini_{upper} \leftarrow 1$

    ▷ Replace initial subtree with its lower or upper subtree and calculate score
 4: if $Subtree_{initial}$ is non-leaf node then
 5:     $Subtree_{lower} \leftarrow$ Lower subtree of $Subtree_{initial}$
 6:     $gini_{lower} \leftarrow$ GiniImpurity($Subtree_{lower}, X_I, y_I$)

 7:     $Subtree_{upper} \leftarrow$ Upper subtree of $Subtree_{initial}$
 8:     $gini_{upper} \leftarrow$ GiniImpurity($Subtree_{upper}, X_I, y_I$)
 9: end if

    ▷ Determine neighbor subtree with the best Gini impurity score (a lower score is better)
10: if ($gini_{parallel} \leq gini_{lower}$) and ($gini_{parallel} \leq gini_{upper}$) then
11:     $Subtree_{neighbor} \leftarrow Subtree_{parallel}$
12: else
13:     if $gini_{lower} \leq gini_{upper}$ then
14:         $Subtree_{neighbor} \leftarrow Subtree_{lower}$
15:     else
16:         $Subtree_{neighbor} \leftarrow Subtree_{upper}$
17:     end if
18: end if
19: return $Subtree_{neighbor}$
```

The classification tree state transition is illustrated in Algorithm 3. The algorithm accepts three inputs: an initial classification subtree $Subtree_{initial}$ to optimize; training data X_I, y_I corresponding to subset of training data X, y presenting in the current subtree to optimize; and selected rank $rank_{selected}$ as determined by Algorithm 2. The algorithm outputs a neighbor classification subtree $Subtree_{neighbor}$ with optimal or suboptimal split at the root node of $Subtree_{initial}$ depending on how we update the subtree. The procedure to update the current subtree is deterministically defined into three options over the root node of the subtree: updating a parallel split by considering all possible cutoffs over all features and values; replacing a branch node with its left subtree; or replacing a branch node with its right subtree. The logic to update a tree state is very similar to that of OCT, except $Subtree_{parallel}$ is not necessarily the subtree obtained from the best split, but it could be the r^{th} best split where r ranges from 1 to 4.

3.2.4 Algorithm 4: Optimize Parallel Split

Algorithm 4 shows how a parallel split is optimized which is moderately similar to that of OCT. The algorithm accepts exactly the same three inputs as in Algorithm 3 including an initial classification subtree $Subtree_{initial}$ to optimize, training data X_I, y_I corresponding to subset of training data X, y presenting in the current subtree to optimize, and selected rank $rank_{selected}$ as determined by Algorithm 2. The algorithm outputs an optimized subtree $Subtree_{parallel}$ with an optimized parallel split at the root node of $Subtree_{initial}$ and the corresponding Gini impuriry score $gini_{parallel}$ of the optimized subtree.

To update a parallel split, Algorithm 4 iterates through all features and possible cutoff values of the subtree root node. We examine all possible cutoffs by going through all features, sorting all distinct feature values in ascending order and calculating each cutoff as a midpoint between 2 consecutive sorted feature values. The main difference from OCT is that OCT-SA does not always select the best parallel split, but the split is selected based on solution ranking determined by Algorithm 2. Hence, the algorithm needs to keep track of parallel split ranking. If multiple different splits on the same split node resulted in the same objective

Algorithm 4 OptimizeParallelSplit

Input: Initial classification subtree $Subtree_{initial}$ to optimize parallel split;
Training data X_I, y_I;
Selected rank $rank_{selected}$ of parallel split

Output: Subtree $Subtree_{parallel}$ with optimized parallel split at root of $Subtree_{initial}$;
Score $gini_{parallel}$ of the optimized subtree

```
1: $n' \leftarrow$ Number of samples in $X_I$
2: $p' \leftarrow$ Number of features in $X_I$
3: $candidates \leftarrow \emptyset$

   ▷ Iterate through all $p'$ features
4: for $v = 1, 2, 3, ..., p'$ do
5:     $featureValues \leftarrow \{X_{I_{uv}} : u = 1, 2, 3, ..., n'\}$
6:     $featureValues \leftarrow$ Sort $featureValues$ in ascending order
7:     $f \leftarrow$ Number of unique feature values in $featureValues$
8:     $branchValues \leftarrow \{\frac{1}{2}(featureValues_m + featureValues_{m+1}) : m = 1, 2, 3, ..., f-1\}$

       ▷ Iterate through all branch values of feature $v$
9:     for $m = 1, 2, 3, ..., f-1$ do
10:
           ▷ Update split at root node and calculate score
11:        $Subtree_{optimized} \leftarrow$ Split root of $Subtree_{initial}$ with feature $v$ and branch value $m$
12:        $gini_{optimized} \leftarrow \textsc{GiniImpurity}(Subtree_{optimized}, X_I, y_I)$

           ▷ Keep only one subtree for each unique score
13:        if no subtree in $candidates$ with score equals $gini_{optimized}$ then
14:            $candidates \leftarrow candidates \cup \{(Subtree_{optimized}, gini_{optimized})\}$
15:        end if
16:    end for
17: end for

    ▷ Pick final parallel optimized subtree based on score ranking
18: $candidates_{sorted} \leftarrow$ Sort $candidates$ by $gini_{optimized}$ score in ascending order
19: $Subtree_{parallel}, gini_{parallel} \leftarrow$ Pick subtree and score at $rank_{selected}$ from $candidates_{sorted}$
20: return $Subtree_{parallel}, gini_{parallel}$
```

value, the algorithm only kept the first split it found.

3.2.5 Algorithm 5: Calculate Probability

Algorithm 5 CalculateProbability

```
Input: Gini impurity score of a current classification subtree $gini_{current}$;
       Gini impurity score of a neighbor classification subtree $gini_{neighbor}$;
       Current temperature $temp_k$ at iteration number $k$
Output: Probability $prob$ to transform current subtree into neighbor subtree

1: $prob \leftarrow 0$        ▷ No subtree transformation if $gini_{neighbor} = gini_{current}$
2: if $gini_{neighbor} < gini_{current}$ then        ▷ A lower Gini impurity score is better
3:     $prob \leftarrow 1$        ▷ Always transform to neighbor subtree with better score
4: else
   ▷ Calculate probability from difference in energy and current temperature
5:     if $gini_{neighbor} > gini_{current}$ then
6:         $\Delta E \leftarrow |gini_{current} - gini_{neighbor}|$        ▷ Difference in energy
7:         $prob \leftarrow e^{\frac{-\Delta E}{temp_k}}$        ▷ Euler's number e = 2.7183
8:     end if
9: end if
```

Algorithm 5 calculates the probability that OCT-SA will transform a current subtree into a neighbor subtree. The algorithm accepts three inputs: the Gini impurity score of the current classification subtree $gini_{current}$; the Gini impurity score of the neighbor classification subtree $gini_{neighbor}$; and the current temperature $temp_k$ at iteration number k; and outputs a probability $prob$ to transform the current subtree into the neighbor subtree. If the current subtree has the Gini impurity score equal to that of the neighbor subtree, the transformation will not occur and the probability equals 0. If the neighbor subtree has a lower Gini impurity score (a lower score is better) than that of the current subtree, OCT-SA always allows the subtree transformation so the probability equals 1. Otherwise, we calculate the probability of subtree transformation from the difference in energy ΔE (i.e., the difference in Gini impurity score between the current and neighbor subtree) and the current temperature $temp_k$.

3.3 Results on Real-World Datasets and Discussion

3.3.1 Experimental Setup

In order to make a fair comparison with OCT in Bertsimas and Dunn [11], we also evaluate OCT-SA on the same 61 datasets from the UCI machine learning repository that researchers widely use as benchmark datasets for classification problems. These 61 datasets have variable sizes with the numbers of data points ranging from 47 to 245057; the number of features ranging from 2 to 100; and the number of classes ranging from 2 to 10. We convert all feature values into numeric values in order to be applicable to Algorithm 4.

Similar to Bertsimas and Dunn [11], we also split each dataset into train, validation, and test sets with the ratio of train:validation:test being 50:25:25. We used 10% of the total number of data points in each dataset as a minbucket to avoid overfitting and to ensure the resulting models are interpretable. The classification trees are trained with the maximum depths ranging from 2 to 7. The maximum depth 7 is the highest possible maximum depth given that the minbucket is 10% over the entire dataset and the combination of training and validation datasets which are used to construct a final OCT-SA account for 75% of the entire dataset.

We perform the hyperparameter tuning of OCT-SA only over complexity penalty γ while setting Markov chain length $l = 2$ and search radius $r = 3$ which is the combination of hyperparameters that performs best by average over the 61 datasets as found from our empirical experiments. The set of hyperparameters giving the best validation score is then used to retrain all 103 warm starts over a combination of train and validation set. A tree (out of 103 trees) with the highest score evaluated over the combination of train and validation set is selected as an OCT-SA of each dataset. The out-of-sample score of the OCT-SA is then evaluated on an unseen test set. We compare our models (indicated as OCT-SA) with CART and OCT on out-of-sample Gini impurity score. In order to prove that the results are statistically significant, we run the same experiments over five different train:validation:test splits of each dataset. We measure average performance across the five different data splits together with standard deviation and derive p-values.

3.3.2 OCT-SA vs OCT vs CART

As shown in Table 3.1, OCT-SA performs better than OCT and CART on average Gini impurity score at all maximum depths above 1. At the maximum depth 1, all models achieve almost the same performance with Gini impurity score at approximately 0.3715. Since the classification tree at the maximum depth 1 only contains 1 branching node which is the root node, it is very likely that each model selects the same feature and cutoff to branch on.

When comparing OCT-SA with CART on average Gini impurity score and standard deviation, the results are statistically significant with p-values below 0.05 from the maximum depth 2 to 7. This suggests that OCT-SA successfully improves over CART. On the other hand, when comparing OCT-SA with OCT, the results are statistically significant at only 3 maximum depths including depth 3, 4, and 7. This points out that OCT is a strong algorithm where OCT-SA can truly perform better only for a certain maximum depth constraints. The result of each dataset at maximum depth 7, which is the highest depth in our experiments is also included in Table 3.2 and Table 3.3, where OCT-SA beats OCT on 43 out of 61 datasets. The average Gini impurity scores of OCT-SA, OCT, and CART from Table 3.1 are plotted in Figure 3.2.

From Table 3.2 and Table 3.3, OCT-SA has the largest performance gap over OCT on the banknote authentication dataset. This dataset involves an authentication process for 1372 banknotes whether it is a genuine or a fake banknote. Each banknote has four features: variance, skewness, curtosis, and entropy which are data extracted from a banknote's image. Figure 3.3, 3.4, and 3.5 demonstrate differences in tree structure and performance among OCT-SA, OCT, and CART of the banknote authentication dataset (from one of the five different data splits) trained with maximum depth 7 and minbucket 10% over the entire

Maximum Depth	Mean out-of-sample Gini impurity score			Mean improvement of OCT-SA over		p-value	
	CART	OCT	OCT-SA	OCT	CART	OCT	CART
2	0.3093	0.2942	**0.2899**	**0.0043** ± 0.0025	0.0194 ± 0.0025	0.1291	0.0000
3	0.2857	0.2698	**0.2628**	**0.0069** ± 0.0023	0.0229 ± 0.0047	0.0109	0.0000
4	0.2802	0.2654	**0.2588**	**0.0066** ± 0.0038	0.0213 ± 0.0037	0.0210	0.0001
5	0.2786	0.2646	**0.2584**	**0.0062** ± 0.0032	0.0201 ± 0.0050	0.0543	0.0002
6	0.2786	0.2643	**0.2583**	**0.0060** ± 0.0047	0.0202 ± 0.0055	0.0920	0.0004
7	0.2786	0.2643	**0.2573**	**0.0070** ± 0.0046	0.0213 ± 0.0041	0.0358	0.0001

Table 3.1: Average out-of-sample Gini impurity score (a lower score is better) averaged across 61 datasets from the UCI machine learning repository trained with maximum depth 2 to 7 and minbucket 10% over entire dataset. Mean improvement and p-value are calculated between OCT-SA vs OCT and OCT-SA vs CART. A positive mean improvement indicates OCT-SA having better performance than OCT or CART.

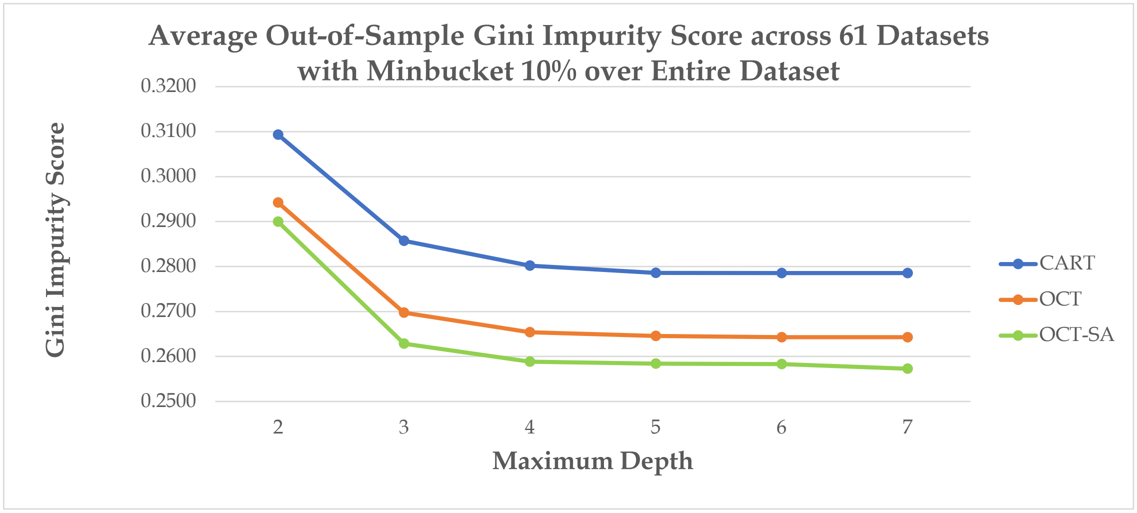

Figure 3.2: Average out-of-sample Gini impurity score (a lower score is better) averaged across 61 datasets from the UCI machine learning repository trained with maximum depth 2 to 7 and minbucket 10% over entire dataset.

dataset. In terms of performance, OCT-SA performs best by achieving out-of-sample Gini impurity score (a lower score is better) of 0.1090, followed by OCT with the score 0.1885 while CART performs worse with the score 0.1904. Both OCT-SA and CART have the same level of complexity with 5 splits. OCT has the simplest tree structure with only 2 splits but it can perform better than CART with 5 splits. This points out that OCT can identify meaningful splits better than CART. In terms of split features, all three models select variance and skewness where OCT only split on these two features, while both OCT-

No.	Dataset	n	p	K	Mean out-of-sample Gini impurity score			Mean improvement of OCT-SA over	
					CART	OCT	OCT-SA	OCT	CART
1	acute inflammations 1	120	6	2	0.1431	0.0000	0.0000	0.0000 ± 0.0000	0.1431 ± 0.0707
2	acute inflammations 2	120	6	2	0.0899	0.0087	0.0000	0.0087 ± 0.0144	0.0899 ± 0.0354
3	balance scale	625	4	3	0.4294	0.4142	0.3964	0.0178 ± 0.0318	0.0330 ± 0.0282
4	banknote authentication	1372	4	2	0.1857	0.2047	0.0990	0.1057 ± 0.0480	0.0867 ± 0.0275
5	blood transfusion	748	4	2	0.3187	0.3206	0.3119	0.0087 ± 0.0093	0.0068 ± 0.0095
6	breast cancer	277	9	2	0.3613	0.3723	0.3705	0.0018 ± 0.0204	-0.0091 ± 0.0240
7	breast cancer diagnostic	569	30	2	0.1115	0.1158	0.0822	0.0335 ± 0.0131	0.0293 ± 0.0169
8	breast cancer prognostic	194	32	2	0.3480	0.3560	0.3648	-0.0088 ± 0.0234	-0.0168 ± 0.0351
9	car evaluation	1728	6	4	0.2517	0.2517	0.2521	-0.0004 ± 0.0009	-0.0004 ± 0.0009
10	chess king rook vs king pawn	3196	36	2	0.2295	0.1265	0.1221	0.0044 ± 0.0098	0.1074 ± 0.0605
11	climate model crashes	540	18	2	0.1324	0.1055	0.1051	0.0004 ± 0.0048	0.0272 ± 0.0214
12	congressional voting records	232	16	2	0.0522	0.0538	0.0530	0.0007 ± 0.0017	-0.0009 ± 0.0024
13	connectionist bench sonar	208	60	2	0.3215	0.3355	0.3107	0.0248 ± 0.0761	0.0108 ± 0.0478
14	contraceptive method choice	1473	9	3	0.5880	0.5789	0.5734	0.0055 ± 0.0078	0.0147 ± 0.0181
15	credit approval	653	15	2	0.2068	0.2009	0.1962	0.0047 ± 0.0113	0.0106 ± 0.0112
16	dermatology	358	34	6	0.2114	0.2112	0.2083	0.0029 ± 0.0170	0.0030 ± 0.0191
17	echocardiogram	62	7	2	0.3168	0.2909	0.3222	-0.0313 ± 0.1101	-0.0055 ± 0.0218
18	ecoli	336	7	8	0.3050	0.2934	0.2866	0.0068 ± 0.0141	0.0184 ± 0.0182
19	fertility	100	9	2	0.2059	0.2153	0.2111	0.0042 ± 0.0122	-0.0052 ± 0.0158
20	haberman survival	306	3	2	0.3389	0.3493	0.3355	0.0138 ± 0.0097	0.0034 ± 0.0131
21	hayes roth	132	4	3	0.5816	0.5883	0.5866	0.0017 ± 0.0190	-0.0050 ± 0.0353
22	heart disease cleveland	297	13	5	0.5222	0.5193	0.5448	-0.0256 ± 0.0520	-0.0226 ± 0.0372
23	hepatitis	80	19	2	0.1626	0.1704	0.1609	0.0094 ± 0.0174	0.0016 ± 0.0413
24	hill valley with noise	606	100	2	0.5076	0.5053	0.5046	0.0007 ± 0.0033	0.0030 ± 0.0046
25	hill valley without noise	606	100	2	0.5031	0.5017	0.5000	0.0017 ± 0.0046	0.0031 ± 0.0049
26	image segmentation	210	19	7	0.2576	0.2347	0.2355	-0.0008 ± 0.0202	0.0221 ± 0.0287
27	indian liver patient	579	10	2	0.3688	0.3641	0.3678	-0.0036 ± 0.0089	0.0010 ± 0.0147
28	ionosphere	351	34	2	0.1512	0.1610	0.1573	0.0037 ± 0.0111	-0.0061 ± 0.0236
29	iris	150	4	3	0.0899	0.0999	0.0839	0.0159 ± 0.0139	0.0060 ± 0.0124
30	magic gamma telescope	19020	10	2	0.3118	0.3012	0.2945	0.0067 ± 0.0036	0.0173 ± 0.0059

Table 3.2: Out-of-sample Gini impurity score (a lower score is better) on the first 30 datasets (out of 61 datasets) from the UCI machine learning repository trained with maximum depth 7 and minbucket 10% over entire dataset where n, p and K denote the number of data points, the number of features, and the number of classes of each dataset respectively. A positive mean improvement indicates OCT-SA having better performance than OCT or CART.

SA and CART select a different additional feature: curtosis for OCT-SA and entropy for CART. OCT-SA selects the additional feature which can significantly improve the overall performance of OCT-SA over OCT by 7.95% (0.1090 vs 0.1885).

Figures 3.6 and 3.7 demonstrate how performance and runtime of OCT-SA, OCT, and CART of the banknote authentication dataset get affected as the maximum depth increases. We conduct the model training on one of the five data splits and it is the same data split used to train OCT-SA, OCT and CART in Figure 3.3, 3.4, and 3.5. In terms of performance measured as out-of-sample Gini impurity score (a lower score is better), OCT-SA performs better than both OCT and CART at all maximum depths while the performance starts

No.	Dataset	n	p	K	Mean out-of-sample Gini impurity score			Mean improvement of OCT-SA over	
					CART	OCT	OCT-SA	OCT	CART
31	mammographic masses	830	5	2	0.2455	0.2487	0.2431	0.0056 ± 0.0089	0.0025 ± 0.0055
32	monks problems 1	124	6	2	0.2735	0.2230	0.2196	0.0034 ± 0.0217	0.0538 ± 0.0461
33	monks problems 2	169	6	2	0.4507	0.4510	0.4430	0.0080 ± 0.0392	0.0077 ± 0.0435
34	monks problems 3	122	6	2	0.1108	0.1220	0.1149	0.0071 ± 0.0090	-0.0041 ± 0.0115
35	mushroom	5644	22	2	0.0878	0.0735	0.0761	-0.0026 ± 0.0116	0.0117 ± 0.0070
36	optical recognition	3823	64	10	0.6606	0.5932	0.5965	-0.0033 ± 0.0079	0.0641 ± 0.0205
37	ozone level detection eight	1847	72	2	0.1159	0.1154	0.1147	0.0007 ± 0.0034	0.0011 ± 0.0070
38	ozone level detection one	1848	72	2	0.0579	0.0569	0.0532	0.0038 ± 0.0026	0.0047 ± 0.0013
39	parkinsons	195	22	2	0.2303	0.1434	0.1162	0.0272 ± 0.0331	0.1141 ± 0.0413
40	pen based recognition	7494	16	10	0.5828	0.5706	0.5536	0.0170 ± 0.0306	0.0292 ± 0.0110
41	planning relax	182	12	2	0.4189	0.4095	0.4091	0.0004 ± 0.0053	0.0098 ± 0.0158
42	qsar biodegradation	1055	41	2	0.3114	0.2691	0.2764	-0.0073 ± 0.0150	0.0350 ± 0.0113
43	seeds	210	7	3	0.1327	0.1434	0.1155	0.0279 ± 0.0218	0.0172 ± 0.0202
44	seismic bumps	2584	18	2	0.1139	0.1147	0.1150	-0.0003 ± 0.0019	-0.0011 ± 0.0012
45	skin segmentation	245057	3	2	0.1602	0.0987	0.1026	-0.0039 ± 0.0014	0.0576 ± 0.0008
46	soybean small	47	35	4	0.0700	0.0600	0.0600	0.0000 ± 0.1225	0.0100 ± 0.0945
47	spambase	4601	57	2	0.2535	0.1773	0.1689	0.0084 ± 0.0034	0.0846 ± 0.0012
48	spect heart	80	22	2	0.3813	0.3939	0.3890	0.0049 ± 0.0143	-0.0077 ± 0.0161
49	spectf heart	80	44	2	0.2787	0.3493	0.3342	0.0151 ± 0.0378	-0.0556 ± 0.1000
50	statlog german credit	1000	20	2	0.3649	0.3687	0.3638	0.0049 ± 0.0100	0.0011 ± 0.0051
51	statlog landsat	4435	36	6	0.3967	0.3550	0.3494	0.0056 ± 0.0158	0.0473 ± 0.0082
52	teaching assistant	151	5	3	0.5992	0.5968	0.6084	-0.0117 ± 0.0284	-0.0093 ± 0.0297
53	thoraric surgery	470	16	2	0.2489	0.2528	0.2557	-0.0030 ± 0.0087	-0.0068 ± 0.0086
54	thyroid disease ann	3772	21	3	0.0866	0.0866	0.0866	0.0000 ± 0.0000	0.0000 ± 0.0000
55	thyroid disease new	215	5	3	0.1348	0.1200	0.1140	0.0060 ± 0.0305	0.0208 ± 0.0245
56	tic tac toe endgame	958	9	2	0.3645	0.3680	0.3680	0.0000 ± 0.0001	-0.0035 ± 0.0063
57	wall following robot 2	5456	2	4	0.0631	0.0631	0.0631	0.0000 ± 0.0000	0.0000 ± 0.0000
58	wall following robot 24	5456	24	4	0.2357	0.1408	0.1406	0.0002 ± 0.0033	0.0951 ± 0.0068
59	wine	178	13	3	0.1669	0.1219	0.0981	0.0238 ± 0.0579	0.0688 ± 0.0546
60	yeast	1484	8	10	0.6342	0.6111	0.6035	0.0076 ± 0.0028	0.0307 ± 0.0065
61	zoo	101	16	7	0.1560	0.1728	0.1484	0.0244 ± 0.0170	0.0075 ± 0.0109

Table 3.3: Out-of-sample Gini impurity score (a lower score is better) on the last 31 datasets (out of 61 datasets) from the UCI machine learning repository trained with maximum depth 7 and minbucket 10% over entire dataset where n, p and K denote the number of data points, the number of features, and the number of classes of each dataset respectively. A positive mean improvement indicates OCT-SA having better performance than OCT or CART.

to stabilize at maximum depth 5, which indicates that no deeper split can improve model performance. On the other hand, OCT performs worse than CART at maximum depth 2 but starts to perform better at maximum depth 3 where both OCT and CART stabilize. This points out that OCT-SA can better utilize a larger search space and find a better model at a deeper maximum depth. In terms of runtime, CART runs significantly fast within only 1 second across all maximum depths. OCT-SA requires relatively the same amount of time to generate a model across all maximum depths with approximately 20 seconds. Since OCT is trained over 100 random warm starts while CART has no warm start and starts with a single root node, this is a reason why OCT requires longer runtime than CART. Regarding

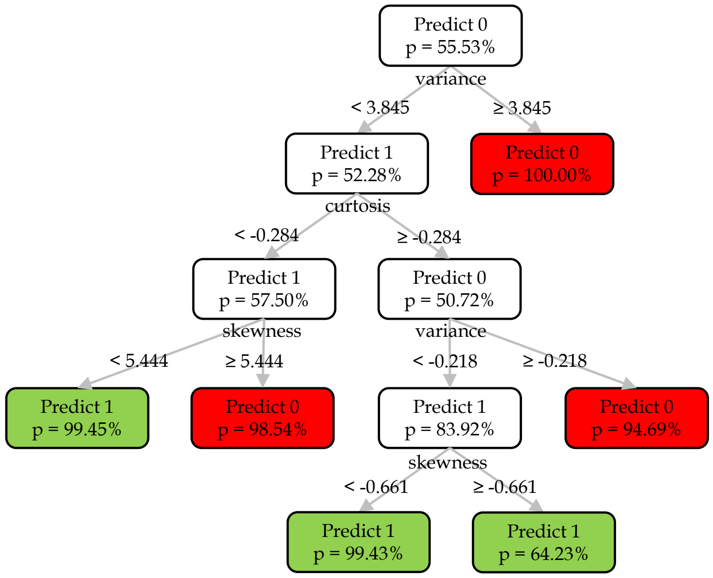

Figure 3.3: OCT-SA of banknote authentication dataset trained with maximum depth 7 and minbucket 10% over entire dataset. The OCT-SA model has 5 splits and achieves out-of-sample Gini impurity score (a lower score is better) of 0.1090.

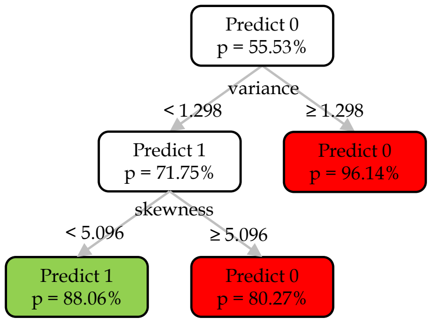

Figure 3.4: OCT of banknote authentication dataset trained with maximum depth 7 and minbucket 10% over entire dataset. The OCT model has 2 splits and achieves out-of-sample Gini impurity score (a lower score is better) of 0.1885.

OCT-SA, the runtime increases as the maximum depth increases starting from 240 seconds at maximum depth 2 and reaching approximately 330 seconds at maximum depth 3 and above. The longer runtime of OCT-SA as compared to OCT is due to a larger number of iterations required by OCT-SA to optimize a model. As specified by the geometric cooling schedule, OCT-SA keeps running until the temperature drops below a predefined threshold where optimization at each temperature level is equivalent to an iteration of local search in

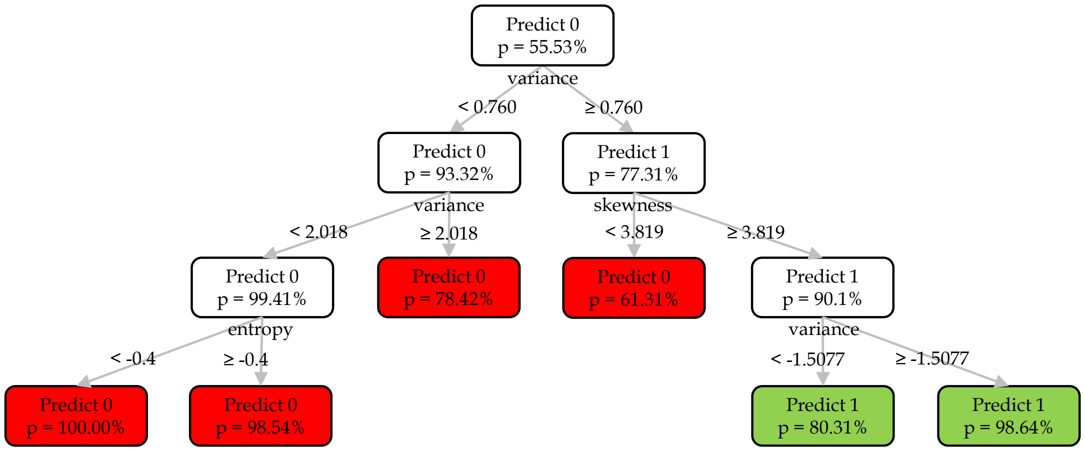

Figure 3.5: CART of banknote authentication dataset trained with maximum depth 7 and minbucket 10% over entire dataset. The CART model has 5 splits and achieves out-of-sample Gini impurity score (a lower score is better) of 0.1904.

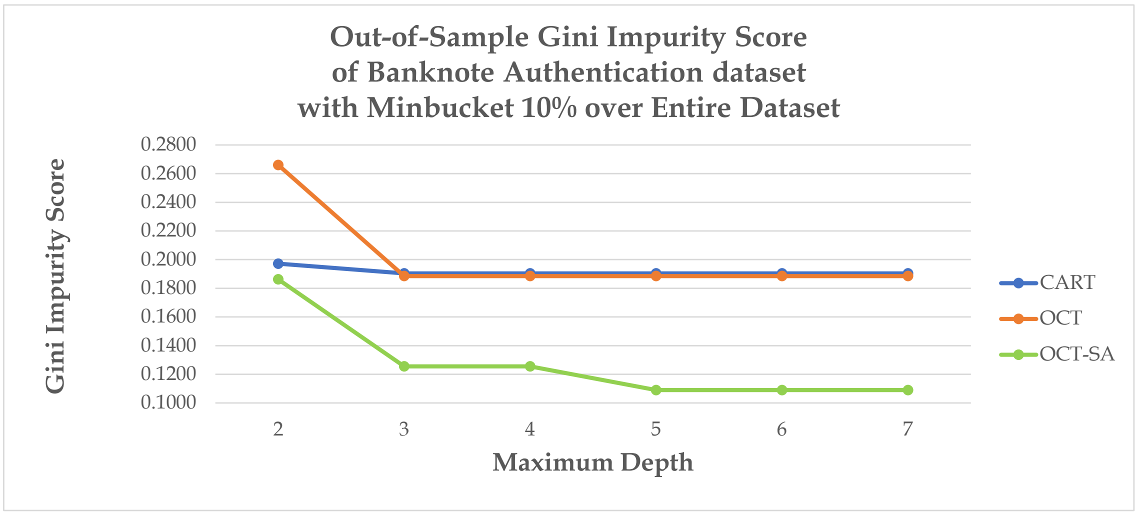

Figure 3.6: Out-of-sample Gini impurity score (a lower score is better) on banknote authentication dataset trained with maximum depth 2 to 7 and minbucket 10% over entire dataset.

OCT. On the other hand, OCT stops whenever no further improvement can be made from 2 consecutive iterations of local search.

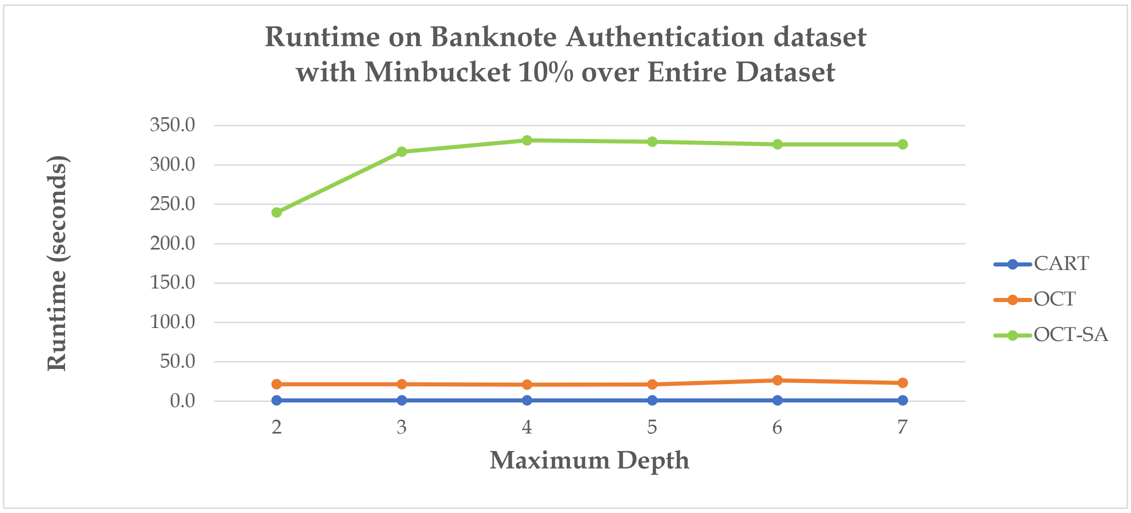

Figure 3.7: Runtime (in seconds) of banknote authentication dataset trained with maximum depth 2 to 7 and minbucket 10% over entire dataset.

3.4 Conclusion

Our research suggests that simulated annealing is beneficial to the classification tree growing process. OCT-SA successfully improves upon OCT by achieving better average out-of-sample Gini impurity score for all maximum depths but it is only statistically significant to maximum depth 3, 4 and 7.

Chapter 4

Optimal Policy Trees with Simulated Annealing (OPT-SA)

To improve on predictive performance of Optimal Prescriptive Trees, Amram et al. [4] developed Optimal Policy Trees (OPT). Even though OPT successfully improves on Optimal Prescriptive Trees by separating rewards estimation from the objective function, OPT still uses local search like that of Optimal Classification Trees (OCT), which does not guarantee global optimality. Since Optimal Classification Trees with Simulated Annealing (OCT-SA) successfully improves on OCT, in this chapter we apply simulated annealing in the process of building OPT and obtain Optimal Policy Trees with Simulated Annealing (OPT-SA). We report computational results on 10 real-world datasets from the UCI machine learning repository and Interpretable AI [51]. By comparing mean out-of-sample of mean rewards, we find that OPT-SA outperforms OPT on 3 out of 10 datasets and outperforms Classification And Regression Trees (CART) on 4 out of 10 datasets.

4.1 Introduction

Bertsimas et al. [14] developed Optimal Prescriptive Trees to prescribe an optimal treatment or a personalized treatment which leads to an optimal outcome for a particular subgroup of subjects. The authors define treatments as anything that can be controlled or changed in a particular problem with the expectation to achieve better outcomes. For example, in order to improve the probability that patients survive, we may alter the dosage of medicines. Another example is to increase the probability that customers will buy strawberries where department stores adjust prices. Even though Optimal Prescriptive Trees has proven to effectively prescribe optimal treatments in certain scenarios, there are cases where Optimal Prescriptive Trees fail to provide realistic results. This failure is due to the design of the objective function, where Optimal Prescriptive Trees combines both rewards estimation and treatment assignment within the same objective function. This design may lead to poor rewards estimation with highly efficient treatment assignment and vice versa. In order to fix this weak point of Optimal Prescriptive Trees, Amram et al. [4] developed Optimal Policy Trees (OPT) by separating the rewards estimation process so as to be independent from treatment assignment when constructing the decision trees. This approach allows reward estimators to independently achieve the optimal performance without compromising with the qualities of treatment assignment and vice versa. However, OPT still applies local search to iteratively find the next best tree transformation. Consequently, this mechanism could lead to local minimas as some worse transformations may lead to better final models. Since Optimal Classification Trees with Simulated Annealing (OCT-SA) was proven to successfully improve on Optimal Classification Trees (OCT), in this paper we apply simulated annealing in OPT and obtain Optimal Policy Trees with Simulated Annealing (OPT-SA).

Simulated Annealing is an optimization method that guarantees to provide a global optimal solution asymptotically under an appropriate cooling schedule [see 12, 58]. While local search always transforms a tree to a neighboring tree with a better objective value, simulated annealing allows some bad transformations which could lead to a better final tree. A cooling schedule is a main controller that drives a tree transformation depending a current temperature and a difference in objective values between the current tree and a neighboring tree. At high temperatures, the probability to accept worse transformations is high. As the temperatures decrease, the chance to accept bad transformations also keeps decreasing. Simulated annealing terminates when it reaches the predefined lowest temperature or no tree transformation happens for two consecutive temperatures. In OPT-SA, we are interested in figuring out alternative treatments which may lead to the better outcomes than the observed outcomes. However, in the observed datasets, not all possible treatment options exist.

In this study, we use counterfactual outcome estimators to estimate those possible outcomes. For example, by following a medical guideline, a physician typically prescribes a specific dosage for a certain type of patient and it is classified as an observed treatment. However, it could be possible that a medical guideline may fail to recommend an optimal dosage, a dosage which can provide the best outcome for that patient. In order to predict what would have happened to that type of patient if a physician had prescribed a different dosages, we can use counterfactual outcome estimators to estimate those counterfactual outcomes. Some examples of counterfactual outcome estimators include Random Forest and XGBoost models. These counterfactual models make predictions on all treatment options that we are interested to consider. Generally, these treatment options are specific to each dataset and are in a subset of observed treatments or treatments that were actually prescribed as recorded in the dataset. We will refer to these counterfactual outcomes as rewards. These predicted rewards are then used to build an optimal policy tree to decide the optimal policy or the best treatment option for each type of subject.

The main differences between OPT-SA and OCT-SA are the objective values where OCT-SA uses the Gini impurity scores while OPT-SA uses the mean rewards, which are specific to each problem. In addition, OCT-SA tries to minimize Gini impurity scores while OPT-SA may want to minimize or maximize the outcomes. For example, in the red wine quality problem from the UCI machine learning data repository [6], OPT-SA aims to predict the optimal pH value to maximize the red wine quality. Another example is to predict the optimal house age in order to minimize the house price of unit area, as buyers are interested in buying the cheapest house.

There are two main contributions we have made in this paper. First, OPT-SA is the first application of the simulated annealing algorithm on policy trees that guarantees to provide optimal solution under an appropriate cooling schedule. Second, we report computing results on the same public grocery pricing dataset from Amram et al. [4] and 9 additional real world datasets from the UCI machine learning repository. These results demonstrate the improvement of OPT-SA over OPT on 3 datasets on out-of-sample performance while maintaining interpretability.

This paper is organized as follows. In Section 4.2, we demonstrate how to apply simulated annealing in OPT construction. The demonstration emphasizes the difference between OPT-SA and OCT-SA. In Section 4.3, we evaluate OPT-SA on the same grocery pricing dataset from Amram et al. [4] and 9 additional real-world datasets from the UCI machine learning repository. In Section 4.4, we make our concluding remarks.

4.2 Algorithms

Since the OCT-SA framework performed better than OCT on classification problems of 61 real-world datasets from the UCI machine learning data repository, we also apply the similar simulated annealing framework to construct OPT-SA. There are five algorithms involved in OPT-SA construction as shown in Figure 4.1, which is similar to that of OCT-SA. The entire flow of OPT-SA is described in Algorithm 6, which accepts seven inputs: an initial policy tree T_1 as a random warm start tree; training data consisting of feature value X; estimated rewards rw; geometric temperature decay rate $c < 1$ of a cooling schedule; Markov chain length l or number of nodes to optimize per iteration; neighbor search radius r or boundary of solution search space; and scaling factor β to adjust difference in mean rewards to be in the range from 0 to 1. Then Algorithm 6 outputs the optimal policy tree as T_{best}, which is an optimal transformation of the initial policy tree T_1 with simulated annealing. Algorithm 6 calls Algorithm 7 to rank and select a tree transformation from candidate trees. Algorithm

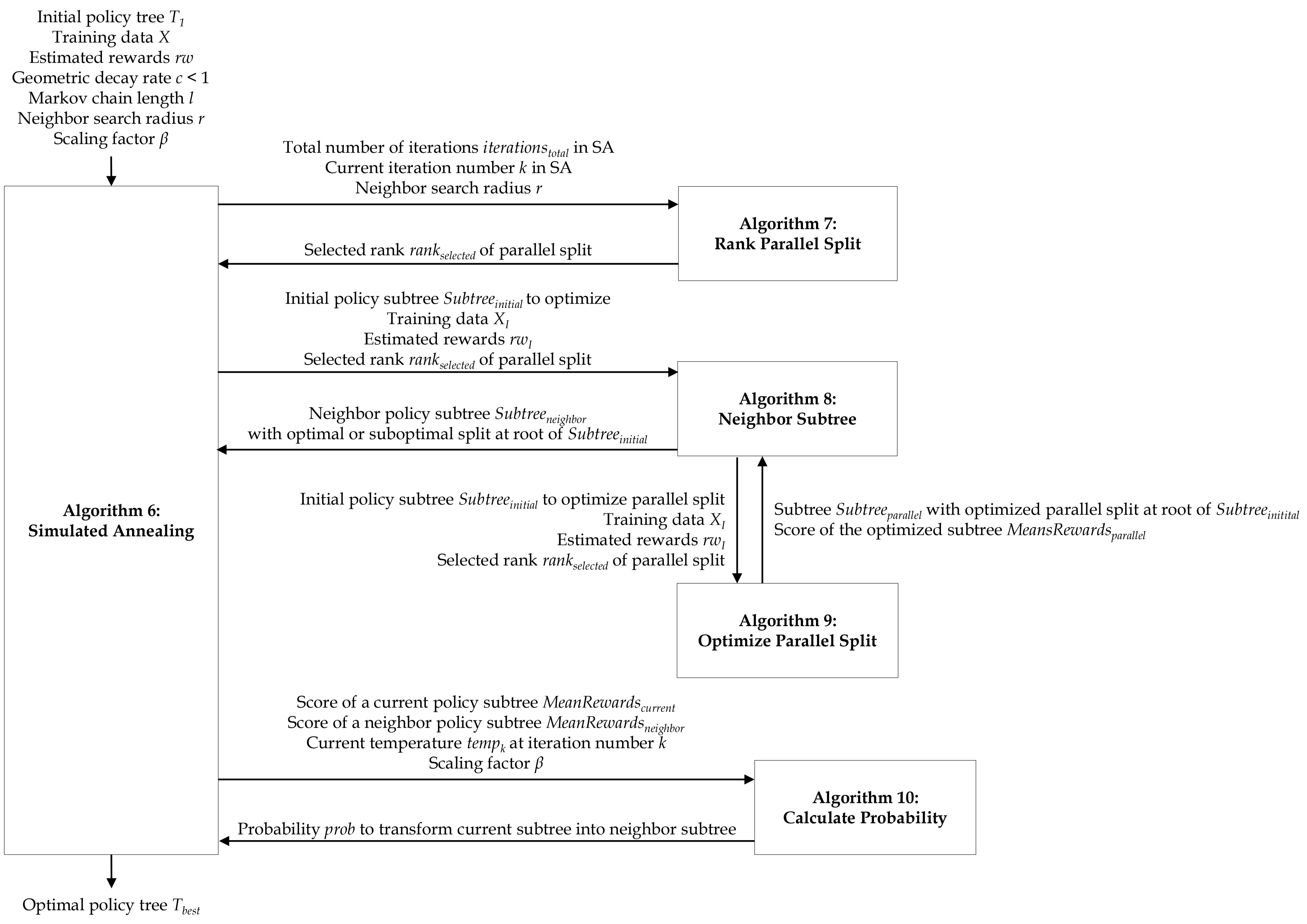
Initial policy tree T_1
Training data X
Estimated rewards rw
Geometric decay rate $c < 1$
Markov chain length l
Neighbor search radius r
Scaling factor β
Algorithm 6: Simulated Annealing
Optimal policy tree T_{best}
Total number of iterations $iterations_{total}$ in SA
Current iteration number k in SA
Neighbor search radius r
Algorithm 7: Rank Parallel Split
Selected rank $rank_{selected}$ of parallel split
Initial policy subtree $Subtree_{initial}$ to optimize
Training data X_l
Estimated rewards rw_l
Selected rank $rank_{selected}$ of parallel split
Algorithm 8: Neighbor Subtree
Neighbor policy subtree $Subtree_{neighbor}$
with optimal or suboptimal split at root of $Subtree_{initial}$
Initial policy subtree $Subtree_{initial}$ to optimize parallel split
Training data X_l
Estimated rewards rw_l
Selected rank $rank_{selected}$ of parallel split
Subtree $Subtree_{parallel}$ with optimized parallel split at root of $Subtree_{initital}$
Score of the optimized subtree $MeansRewards_{parallel}$
Algorithm 9: Optimize Parallel Split
Score of a current policy subtree $MeanRewards_{current}$
Score of a neighbor policy subtree $MeanRewards_{neighbor}$
Current temperature $temp_k$ at iteration number k
Scaling factor β
Algorithm 10: Calculate Probability
Probability $prob$ to transform current subtree into neighbor subtree

Figure 4.1: OPT-SA flowchart of Algorithm 1 to 5 demonstrating inputs, outputs, and sequences of execution.

7 determines the quality of each tree transformation by gradually picking a better candidate tree as the temperature in the geometric cooling schedule decreases. Algorithm 8 generates these candidate trees. There are three options to generate candidate trees by updating the current subtree root node: by replacing with the left subtree, by replacing with the right subtree, or by updating a parallel split. In order to update a parallel split, Algorithm 8 calls Algorithm 9 to generate options to transform the current subtree by updating the branching at the subtree root node with different features and cutoff values. Finally, Algorithm 6 calls Algorithm 10 to determine a probability to transform the current subtree into the neighboring subtree based on their mean reward scores and the current temperature in the annealing schedule.

The mechanism of the simulated annealing algorithm is driven by the geometric cooling schedule, the difference in energy, and the acceptance probability as shown in Equations (4.1), (4.2), and (4.3):

$$temp_{k+1} = \alpha \cdot temp_k, \tag{4.1}$$

$$\Delta E = |MeanRewards_{current} - MeanRewards_{neighbor}|, \tag{4.2}$$

$$prob_k = e^{\frac{-\Delta E}{\beta \cdot temp_k}}. \tag{4.3}$$

where

- $temp$ denotes temperature
- k denotes iteration number
- α denotes temperature decay rate
- ΔE denotes difference in energy between a current and a neighbor state
- $MeanRewards$ denotes mean rewards score
- $prob$ denotes probability to transform a tree from a current state to a neighbor state

- e denotes the Euler's number equals 2.7183

- β denotes a scaling factor to adjust ΔE to be in a range from 0 to 1

From our empirical experiments, we found $\alpha = 0.95$ is an appropriate temperature decay rate for the 10 real-world datasets we used to evaluate OPT-SA.

The main difference between OCT-SA and OPT-SA is the objective function, where OCT-SA uses a Gini impurity score, while OPT-SA uses a mean rewards score, which is specific to each problem. In OCT-SA, we always aim to minimize Gini impurity scores. On the other hand, OPT-SA may want to minimize or maximize mean rewards score depending on the policy type. To simply the implementation of OPT-SA, we convert all maximization problems into minimization problems. In addition, each policy dataset has a different magnitude for its mean rewards and the scaling factor β is applied to adjust a mean rewards score to be in a similar range from 0 to 1, like the Gini impurity score.

We applied the doubly robust method to estimate rewards for almost all datasets where propensity score is also taken into account. We use the direct method for the grocery pricing dataset because an acceptable propensity score estimator cannot be trained with the available observed dataset, possibly due to the lack of correlation between price (the treatment) and the other features as mentioned in Interpretable AI [51] documentation. In order to avoid overfitting, we apply penalty term γ by multiplying it with the complexity parameter cp where cp equals the sum of the number of branching nodes.

4.2.1 Overall Architecture of OPT-SA in Algorithm 6: Simulated Annealing

In Algorithm 6, we iteratively optimize the initial policy tree structure T_1 until it reaches the final policy tree structure T_{best}. The cooling schedule determines the number of iterations $iterations_{total}$ to optimize the tree where the temperature starts at 1 and gradually decreases geometrically with the decay rate $c < 1$ until reaching the final temperature. The algorithm

Algorithm 6 SimulatedAnnealing

```
Input: Initial policy tree $T_1$; Training data $X$; Estimated rewards $rw$;
       Geometric decay rate $c < 1$; Markov chain length $l$; Neighbor search radius $r$;
       Scaling factor $\beta$
Output: Optimal policy tree $T_{best}$
 1: $temp_0 \leftarrow 1$                                      ▷ $temp_0$ is initial temperature
 2: $iterations_{total} \leftarrow k_{min} - 1$ where $temp_0 \cdot c^k < 0.01$
 3: $k \leftarrow 1$                                           ▷ k is iteration number
 4: $T_{best} \leftarrow T_1$                                  ▷ $T_{best}$ keeps the best tree
 5: repeat
 6:     $temp_k \leftarrow temp_0 \cdot c^k$         ▷ Temperature based on geometric cooling schedule
 7:     $rank_k \leftarrow$ RankParallelSplit($iterations_{total}, k, r$)
 8:     $nodes_{optimized} \leftarrow \emptyset$     ▷ Keep track of optimized nodes at current temperature
        ▷ Get a set of all current split nodes or leaf nodes in $T_k$ eligible to be split further
 9:     $nodes_{unoptimized} \leftarrow$ SplitNodes($T_k, X, rw$)
10:     repeat
11:         $nodes_{unoptimized} \leftarrow nodes_{unoptimized} - nodes_{optimized}$
12:         $node_{random} \leftarrow$ RandomNode($nodes_{unoptimized}$)      ▷ Pick one random node
13:         $nodes_{unoptimized} \leftarrow nodes_{unoptimized} - \{node_{random}\}$
14:         $nodes_{optimized} \leftarrow nodes_{optimized} \cup \{node_{random}\}$

            ▷ Find subtree of the random node and calculate score
15:         $Subtree_{random} \leftarrow$ Subtree of $T_k$ where $node_{random}$ is root
16:         $I \leftarrow \{i\colon x_i$ is assigned to a leaf contained in $Subtree_{random}\}$
17:         $MeanRewards_{current} \leftarrow$ MeanRewards($Subtree_{random}, X_I, rw_I$)
            ▷ Find a neighbor of the random subtree and calculate score
18:         $NeighborSubtree_{random} \leftarrow$ NeighborSubtree($Subtree_{random}, X_I, rw_I, rank_k$)
19:         $MeanRewards_{neighbor} \leftarrow$ MeanRewards($NeighborSubtree_{random}, X_I, rw_I$)

            ▷ Determine probability to transform the current subtree to its neighbor state
20:         $prob \leftarrow$ CalculateProbability($MeanRewards_{current}, MeanRewards_{neighbor}, temp_k, \beta$)
21:         if $prob \geq$ RandomNumber($0.1, 1$) then
22:             $T_k \leftarrow$ Replace $Subtree_{random}$ in $T_k$ with $NeighborSubtree_{random}$
23:             if MeanRewards($T_k, X, rw$) < MeanRewards($T_{best}, X, rw$) then
24:                 $T_{best} \leftarrow T_k$          ▷ Update the best tree if Gini impurity score is lower
25:             end if
26:         end if
27:     until $nodes_{unoptimized} = \emptyset$
28:     if $|nodes_{optimized}| = 0$ then       ▷ No node can be optimized at current temperature
        ▷ Move to the last iteration of the current temperature band
29:         $k \leftarrow k_{max}$ where RankParallelSplit($iterations_{total}, k, r$) returns $rank_k$
30:     end if
31:     $k \leftarrow k + 1$
32: until $k = iterations_{total}$
33: return $T_{best}$
```

stops when the temperature is below 0.01. OPT-SA denotes each iteration with a different temperature level as an iteration k where OPT-SA derives T_k by optimizing over Markov chain length l nodes. At higher temperatures, OPT-SA has a higher probability of accepting worse transformations or neighboring trees within the neighbor search radius r.

As with OCT-SA, we implemented OPT-SA in Python version 3.8.15. Since OCT-SA includes OCT as a warm start solution, we also include OPT as a warm start solution for OPT-SA. We constructed OPT warm start solutions using an OPT package of Interpretable AI software [51] version 3.2.0 in Julia version 1.9.0. Then we exported those OPT models into text files and imported them into OPT-SA as warm start solutions. We ran OPT-SA on a Windows laptop and also on MIT Supercloud [84]. In addition, we constructed the CART warm start solutions with scikit-learn Python package [78]. We used a regression tree from the CART module as one of the warm start solutions for OPT-SA. In contrast, we use a classification tree from the CART module as one of the warm solutions for OCT-SA.

4.2.2 Algorithm 7: Rank Parallel Split

Algorithm 7 RankParallelSplit

```
    Input: Total number of iterations $iterations_{total}$ in simulated annealing;
           Current iteration number $k$ in simulated annealing;
           Neighbor search radius $r$
    Output: Selected rank $rank_{selected}$ of parallel split to be used in Algorithm ??

1: $rank_{selected} \leftarrow 1$                              ▷ 1 is the best rank
2: $rank_{set} \leftarrow \{r, r-1, r-2, ..., 1\}$        ▷ Set of all ranks in neighbor search radius $r$
3: for $i$ in $rank_{set}$ do
4:     if $k < \frac{iterations_{total}}{i}$ then
5:         $rank_{selected} \leftarrow i$
6:         break
7:     end if
8: end for
9: return $rank_{selected}$
```

Algorithm 7 in OPT-SA performs exactly the same task as Algorithm 2 in OCT-SA where it assigns ranks to candidate trees and selects a specific rank based on a current simulated annealing temperature. Algorithm 6 passes three inputs into Algorithm 7: the total number of iterations $iterations_{total}$ that OPT-SA will transform a tree; a current iteration number k; and a neighbor search radius r whose optimal value OPT-SA determines from hyperparameter tuning. Then Algorithm 7 outputs a select rank of candidate subtrees into which OPT-SA will convert the current subtree. Unlike OPT, OPT-SA may not select the best tree transformation in terms of objective values. For example, when the neighbor search radius $r = 3$, OPT-SA selects the tree with the third-best objective value at the first iteration. Then as the temperature decreases and reaches a certain cutoff, OPT-SA will start to select the second-best tree for a tree transformation. Finally, OPT-SA will select the best tree when a temperature is low enough. This candidate state ranking mechanism helps OPT-SA to converge faster than allowing random candidate tree transformations, as a traditional simulated annealing algorithm usually does, while also preventing overshooting.

We outline the implementation of candidate state ranking in Algorithm 7. For the search radius r, we divide the entire cooling schedule into r temperature bands: the 1^{st}, the 2^{nd}, ..., and the r^{th} temperature band ranging from the highest to the lowest temperature. If $iterations_{total} = S$, then OPT-SA defines the last iteration number of each of the r temperature bands as $\frac{S}{r}, \frac{S}{r-1}, \frac{S}{r-2}, ..., \frac{S}{2}, S$. We design this temperature band and iteration number mapping based on empirical experiments of the 10 datasets we used to evaluate our OPT-SA algorithm. This design is also similar to that of OCT-SA. In terms of the candidate state selection for each temperature band, OPT-SA selects the r^{th} best state transition for the 1^{st} temperature band, then selects the $(r-1)^{th}$ best state transition for the 2^{nd} temperature band, and finally selects the 1^{st} best state transition for the r^{th} temperature band, i.e., the final temperature band. To give an example, let the total number of iterations $iterations_{total} = 90$ and the search radius $r = 3$. We then divide this cooling schedule into 3 temperature bands: the 1^{st} temperature band for the iteration number 1 to 30, the 2^{nd} temperature band for the iteration number 31 to 45, and the 3^{rd} temperature band for the iteration number 46 to 90 where OPT-SA selects the 3^{rd}, the 2^{nd} and the 1^{st} best state transition respectively.

4.2.3 Algorithm 8: Neighbor Subtree

Algorithm 8 illustrates how OPT-SA transforms the policy tree. OPT-SA passes four inputs into Algorithm 8: an initial policy subtree $Subtree_{initial}$ to optimize; training data X_I which is the subset of training data X in the current subtree that OPT-SA optimizes; estimated rewards rw; and selected rank $rank_{selected}$ which is derived by Algorithm 7. Algorithm 8 outputs a policy subtree $Subtree_{neighbor}$ which is a subtree in the neighboring area of the $Subtree_{initial}$. Similar to OPT, there are three options to update the current subtree of OPT-SA: updating the parallel split, pruning the left subtree, or pruning the right subtree. The policy subtree $Subtree_{neighbor}$ may be either an optimal or suboptimal optimization at the root node of the subtree $Subtree_{initial}$, as determined by the selected rank $rank_{selected}$ where $rank_{selected} = r$ corresponds to the r^{th} best neighboring subtree when updating a parallel split.

4.2.4 Algorithm 9: Optimize Parallel Split

Being almost the same as that of OPT, Algorithm 9 optimizes a parallel split by iterating through all features and cutoff values but does not always update the parallel split with the option that results in the best objective value. OPT-SA passes four inputs to Algorithm 9. These four inputs to Algorithm 9 are exactly the same as the four inputs to Algorithm 8: an initial policy subtree $Subtree_{initial}$ to optimize; training data X_I, which is the subset of training data X in the current subtree that OPT-SA optimizes; estimated rewards rw_I; and selected rank $rank_{selected}$, which is derived by Algorithm 7. Assuming $rank_{selected} = r$, Algorithm 9 outputs the r^{th} best optimized subtree $Subtree_{parallel}$ by updating the root node of the subtree $Subtree_{initial}$ and the mean rewards score $MeanRewards_{parallel}$ of this optimized subtree.

In order for Algorithm 9 to optimize a parallel split by iterating through all possible cutoff

Algorithm 8 NeighborSubtree

Input: Initial policy subtree $Subtree_{initial}$ to optimize;
Training data X_I;
Estimated rewards rw_I;
Selected rank $rank_{selected}$ of parallel split

Output: Neighbor classification subtree $Subtree_{neighbor}$ with optimal or suboptimal split at root of $Subtree_{initial}$

```
    ▷ Optimize parallel split of initial subtree and calculate score
 1: $Subtree_{parallel}, MeanRewards_{parallel} \leftarrow$ OptimizeParallelSplit($Subtree_{initial}, X_I, rw_I, rank_{selected}$)

    ▷ $\infty$ is the worst mean reward score
 2: $MeanRewards_{lower} \leftarrow \infty$
 3: $MeanRewards_{upper} \leftarrow \infty$

    ▷ Replace initial subtree with its lower or upper subtree and calculate score
 4: if $Subtree_{initial}$ is non-leaf node then
 5:     $Subtree_{lower} \leftarrow$ Lower subtree of $Subtree_{initial}$
 6:     $MeanRewards_{lower} \leftarrow$ MeanRewards($Subtree_{lower}, X_I, rw_I$)

 7:     $Subtree_{upper} \leftarrow$ Upper subtree of $Subtree_{initial}$
 8:     $MeanRewards_{upper} \leftarrow$ MeanRewards($Subtree_{upper}, X_I, rw_I$)
 9: end if

    ▷ Determine neighbor subtree with the mean rewards score (a lower score is better)
10: if ($MeanRewards_{parallel} \leq MeanRewards_{lower}$) and ($MeanRewards_{parallel} \leq$
    $MeanRewards_{upper}$) then
11:     $Subtree_{neighbor} \leftarrow Subtree_{parallel}$
12: else
13:     if $MeanRewards_{lower} \leq MeanRewards_{upper}$ then
14:         $Subtree_{neighbor} \leftarrow Subtree_{lower}$
15:     else
16:         $Subtree_{neighbor} \leftarrow Subtree_{upper}$
17:     end if
18: end if
19: return $Subtree_{neighbor}$
```

Algorithm 9 OptimizeParallelSplit

```
Input: Initial policy subtree $Subtree_{initial}$ to optimize parallel split;
       Training data $X_I$;
       Estimated rewards $rw_I$;
       Selected rank $rank_{selected}$ of parallel split
Output: Subtree $Subtree_{parallel}$ with optimized parallel split at root of $Subtree_{initial}$
        Score $MeanRewards_{parallel}$ of the optimized subtree

 1: $n' \leftarrow$ Number of samples in $X_I$
 2: $p' \leftarrow$ Number of features in $X_I$
 3: $candidates \leftarrow \emptyset$

    ▷ Iterate through all $p'$ features
 4: for $v = 1, 2, 3, ..., p'$ do
 5:     $featureValues \leftarrow \{X_{I_{uv}} : u = 1, 2, 3, ..., n'\}$
 6:     $featureValues \leftarrow$ Sort $featureValues$ in ascending order
 7:     $f \leftarrow$ Number of unique feature values in $featureValues$
 8:     $branchValues \leftarrow \{\frac{1}{2}(featureValues_m + featureValues_{m+1}) : m = 1, 2, 3, ..., f-1\}$

        ▷ Iterate through all branch values of feature $v$
 9:     for $m = 1, 2, 3, ..., f-1$ do
10:
            ▷ Update split at root node and calculate score
11:         $Subtree_{optimized} \leftarrow$ Split root of $Subtree_{initial}$ with feature $v$ and branch value $m$
12:         $MeanRewards_{optimized} \leftarrow \textsc{MeanRewards}(Subtree_{optimized}, X_I, rw_I)$

            ▷ Keep only one subtree for each unique score
13:         if no subtree in $candidates$ with score equals $MeanRewards_{optimized}$ then
14:             $candidates \leftarrow candidates \cup \{(Subtree_{optimized}, MeanRewards_{optimized})\}$
15:         end if
16:     end for
17: end for

    ▷ Pick final parallel optimized subtree based on score ranking
18: $candidates_{sorted} \leftarrow$ Sort $candidates$ by $MeanRewards_{optimized}$ score from best to worst
19: $Subtree_{parallel}, MeanRewards_{parallel} \leftarrow$ Pick subtree and score at $rank_{selected}$ from
    $candidates_{sorted}$
20: return $Subtree_{parallel}, MeanRewards_{parallel}$
```

values of each feature, we place all unique feature values in ascending order and calculate midpoints between each pair of two consecutive feature values. Since OPT-SA does not always select the feature and cutoff value that gives the best objective value, the algorithm needs to keep track of the r best options to update a parallel split, where $rank_{selected} = r$. If there are multiple options to update a parallel split that results in the same objective value, the algorithm only keeps the first option.

4.2.5 Algorithm 10: Calculate Probability

Algorithm 10 CalculateProbability

```
Input: Score of a current policy subtree $Mean_{current}$;
       Score of a neighbor policy subtree $gini_{neighbor}$;
       Current temperature $temp_k$ at iteration number $k$;
       Scaling factor $\beta$
Output: Probability $prob$ to transform current subtree into neighbor subtree

1: $prob \leftarrow 0$  ▷ No subtree transformation if $MeanRewards_{neighbor} = MeanRewards_{current}$
2: if $MeanRewards_{neighbor} < MeanRewards_{current}$ then    ▷ A lower score is better
3:     $prob \leftarrow 1$    ▷ Always transform to neighbor subtree with better score
4: else
   ▷ Calculate probability from difference in energy and current temperature
5:     if $MeanRewards_{neighbor} > MeanRewards_{current}$ then
6:         $\Delta E \leftarrow |MeanRewards_{current} - MeanRewards_{neighbor}|$    ▷ Difference in energy
7:         $prob \leftarrow e^{\frac{-\Delta E}{\beta \cdot temp_k}}$    ▷ Euler's number e = 2.7183
8:     end if
9: end if
```

To drive probabilistic transformations to a worse neighboring state, Algorithm 10 calculates the probability of such transformations. OPT-SA passes four inputs into Algorithm 10: the mean rewards score of the current policy subtree $MeanRewards_{current}$; the mean rewards score of the neighbor policy subtree $MeanRewards_{neighbor}$; the current temperature $temp_k$ at the current iteration number k; and the scaling factor β to adjust the difference in energy ΔE to be in the range from 0 to 1. Algorithm 10 outputs a probability $prob$ to transform the current policy subtree whose mean rewards score equals $MeanRewards_{current}$ into the neighbor subtree whose mean rewards score equals $MeanRewards_{neighbor}$. We calculate the transformation probability from the difference in energy ΔE and the current temperature $temp_k$ as shown in Equation (4.3). On the other hand, if the mean rewards score of the neighbor policy subtree $MeanRewards_{neighbor}$ is better than that of the current subtree, i.e., $MeanRewards_{current}$, the probability to transform the current subtree to this better subtree equals 1, which means we always allow transformations to a better subtree. Whenever $MeanRewards_{neighbor} = MeanRewards_{current}$, we never make transformation between states with the same objective value, so the probability of transformation is set to 0.

4.3 Results on Real-World Datasets and Discussion

4.3.1 Experimental Setup

To compare OPT-SA with OPT in Amram et al. [4], we construct OPT-SA on the same grocery pricing dataset which is the only public real-world dataset used in that paper. In addition, we run the experiments on 9 real-world datasets from the UCI machine learning data repository. These datasets have the number of data points ranging from 301 to 97295; and the number of features ranging from 5 to 14. Their outcomes and treatments are either continuous or binary. To make all feature values feasible for Algorithm 9, we convert all feature values into numeric values.

We split each of the 10 datasets into train, validation, and test sets with the ratio of train:validation:test being 35:15:50. The reason that we allocate a large amount of data points to the test set is because OPT-SA needs to construct a separate rewards estimator for the training set and the test set in order to avoid information leakage. In terms of minbucket, we tried several minbucket values: 1%, 3%, 5%, and 10% of the total number of data samples. We found that 3% is the highest value of minbucket for most of the datasets before OPT throws the error of not having enough data samples for each treatment option. However, the minbucket of the grocery pricing dataset needs to be as low as 1% as there are

No.	Dataset	$Variable_{outcome}$	$Type_{outcome}$	$Variable_{treatment}$	$Type_{treatment}$	n	p
1	auto mpg	miles per gallon	continuous	acceleration	continuous	392	6
2	credit approval policy	approval status	binary	anonymized attribute	binary	653	14
3	grocery pricing	buying decision	binary	price	continuous	97295	7
4	wine quality red	quality	continuous	pH	continuous	694	10
5	wine quality white	quality	continuous	residual sugar	continuous	1275	10
6	used car	selling price	continuous	km driven	continuous	301	7
7	garments worker productivity	productivity	continuous	incentive	continuous	673	13
8	concrete compressive strength	concrete compressive strength	continuous	water	continuous	1030	7
9	real estate	house price of unit area	continuous	house age	continuous	414	5
10	qsar fish toxicity	LC50	continuous	MLOGP	continuous	908	5

Table 4.1: Variable and type of outcome and treatment, number of samples n, and number of features p of the 10 datasets from the UCI machine learning repository and Interpretable AI [51]

not that many data samples for some treatment options.

Regarding a type of rewards estimator, OPT-SA utilizes Random Forest and uses 100 trees in a forest of all datasets, except the grocery pricing dataset. Since the grocery pricing dataset is much larger than other datasets, OPT-SA utilizes Random Forest with 1000 trees in a forest. We train policy trees with maximum depths in the range from 2 to 5, as policy trees with the maximum depth 5 are still interpretable while being complex enough to capture meaningful splits. We reformulate the OPT-SA optimization problems to resemble those of OCT-SA. Hence, we can use similar hyperparameters where Markov chain length $l = 2$ and search radius $r = 3$. We need to tune only complexity penalty γ. As in OCT-SA, OPT-SA first tunes a set of hyperparameters over a validation dataset and retrains the model using the combination of validation and training datasets over 103 warm start solutions (i.e., including CART, OPT, and a single root node in addition to the 100 random warm starts).

Finally, the policy with the best objective value among all 103 optimized trees is the OPT-SA. We then evaluate out-of-sample performance of this OPT-SA model on the unseen test dataset. We perform the experiments on 5 different train-test data splits and average the results as well as getting standard deviations. The characteristics of each dataset are illustrated in Table 4.1 where we list the variable and type of both the outcome and treatment, the number of samples n, and the number of features p of each dataset.

					Mean out-of-sample of mean rewards			Mean improvement of OPT-SA over	
No.	Dataset	Policy	n	p	CART	OPT	OPT-SA	OPT	CART
1	auto mpg	maximization	392	6	23.3280	23.8520	**23.4420**	**-0.4100** ± 0.6435	**0.1140** ± 0.3706
2	credit approval policy	maximization	652	14	0.4272	0.4486	**0.4472**	**-0.0014** ± 0.0134	**0.0200** ± 0.0162
3	grocery pricing	maximization	92588	7	0.0556	0.0556	**0.0556**	**0.0000** ± 0.0000	**0.0000** ± 0.0000
4	wine quality red	maximization	694	10	5.4437	5.4506	**5.4419**	**-0.0087** ± 0.0162	**-0.0018** ± 0.0254
5	wine quality white	maximization	1276	10	6.0644	6.0703	**6.0805**	**0.0102** ± 0.0591	**0.0161** ± 0.0275
6	used car	maximization	300	7	5.2047	5.1220	**5.1383**	**0.0163** ± 0.1137	**-0.0664** ± 0.0948
7	garments worker productivity	maximization	672	12	0.7518	0.7547	**0.7504**	**-0.0043** ± 0.0024	**-0.0014** ± 0.0027
8	concrete compressive strength	maximization	1030	7	46.2322	46.2322	**45.8659**	**-0.3663** ± 0.3177	**-0.3663** ± 0.3177
9	real estate	minimization	414	5	35.9262	36.0852	**36.2738**	**-0.1886** ± 0.4945	**-0.3476** ± 0.8438
10	qsar fish toxicity	minimization	634	5	4.2368	4.2626	**4.2040**	**0.0586** ± 0.0632	**0.0328** ± 0.0398

Table 4.2: Out-of-sample mean rewards (a higher mean rewards is better for maximization policy and vice versa) on 10 datasets from UCI machine learning repository trained with maximum depth 5 and minbucket 3% (only grocery pricing uses minbucket 1%) over entire dataset where Policy, n and p denote the policy type (maximization or minimization), the number of data points and the number of features of each dataset, respectively. A positive mean improvement indicates OPT-SA having better performance than OPT or CART.

4.3.2 OPT-SA vs OPT vs CART

In Table 4.2, we calculate the average mean rewards scores across all 5 data splits of CART, OPT, and OPT-SA. We also calculate the mean improvement of OPT-SA over OPT as well as the mean improvement of OPT-SA over CART, where a positive mean improvement indicates that OPT-SA outperforms OPT or CART. As shown in Table 4.2, OPT-SA outperforms OPT on 3 datasets and also outperforms CART on 4 datasets

The QSAR fish toxicity dataset has the largest performance gap between OPT-SA and OPT as well as OPT-SA and CART. This dataset involves 6 features which are molecular descriptors of 908 chemicals and contains 634 data points. We can use these molecular descriptors to predict quantitative acute aquatic toxicity towards the fish called fathead minnow (Pimephales promelas). The structure of OPT-SA, OPT, and CART of the QSAR fish toxicity dataset are as shown in Figure 4.2, 4.3, and 4.4.

Figure 4.5 demonstrates the runtime of OPT-SA, OPT, and CART of the QSAR fish toxicity dataset when we train these models at different maximum depths ranging from 2 to 5 on one of the five data splits. We use the same data split that we use to train OPT-SA, OPT, and CART in the Figure 4.2, 4.3, and 4.4. Both CART and OPT spend approximately the same amount of time to train their models, i.e., around one minute. In contrast, the

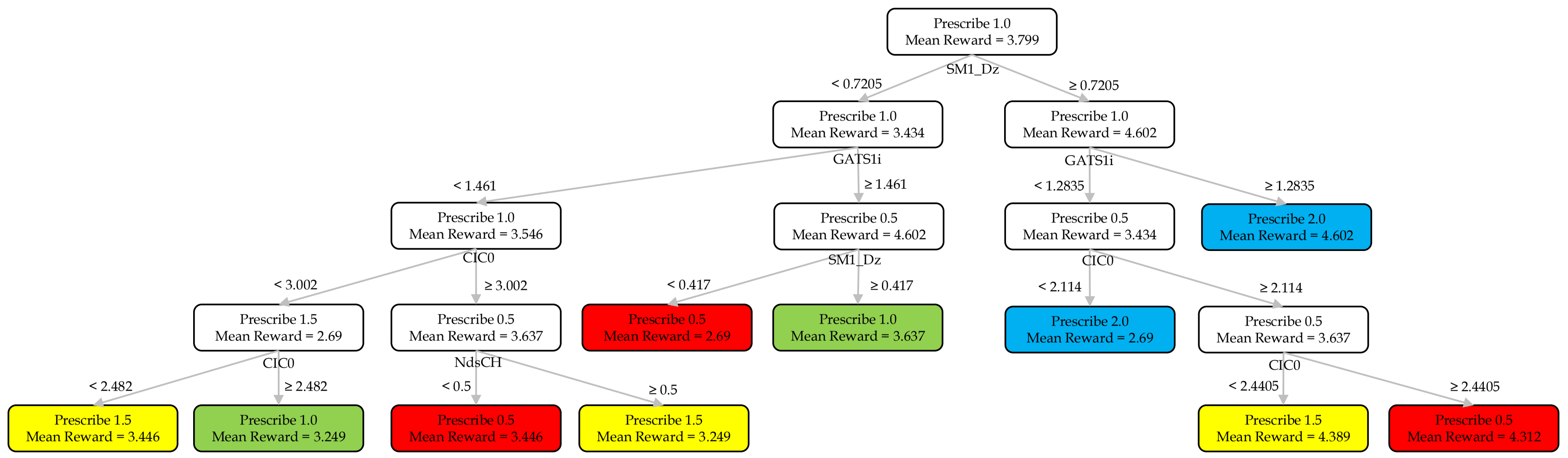

Figure 4.2: OPT-SA of QSAR fish toxicity dataset trained with maximum depth 5 and minbucket 3% over entire dataset. The OPT-SA model has 9 splits and achieves out-of-sample mean rewards score (a lower score is better) of 4.3270.

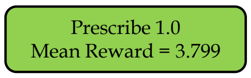

Figure 4.3: OPT of QSAR fish toxicity dataset trained with maximum depth 5 and minbucket 3% over entire dataset. The OCT model has only one split and achieves out-of-sample mean rewards score (a lower score is better) of 4.4770.

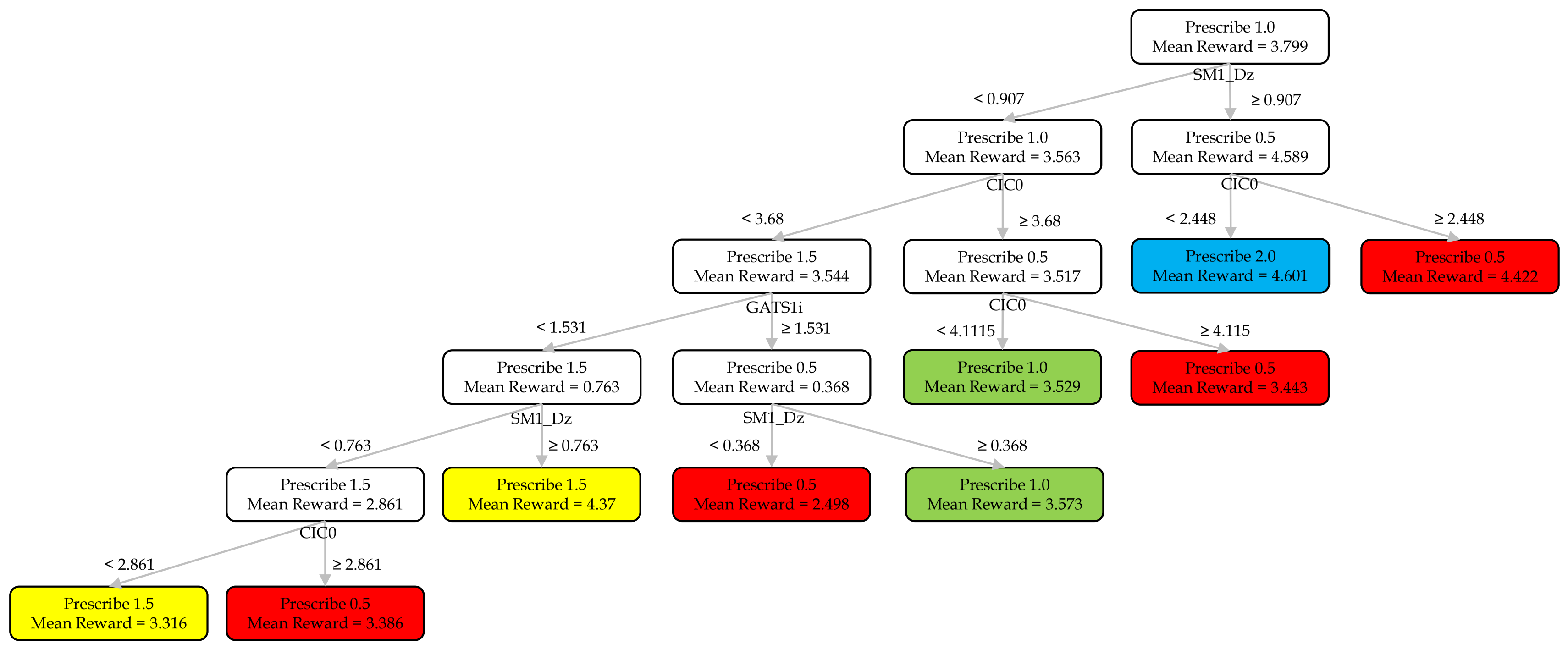

Figure 4.4: CART of QSAR fish toxicity dataset trained with maximum depth 5 and minbucket 3% over entire dataset. The CART model has 8 splits and achieves out-of-sample mean rewards score (a lower score is better) of 4.3620.

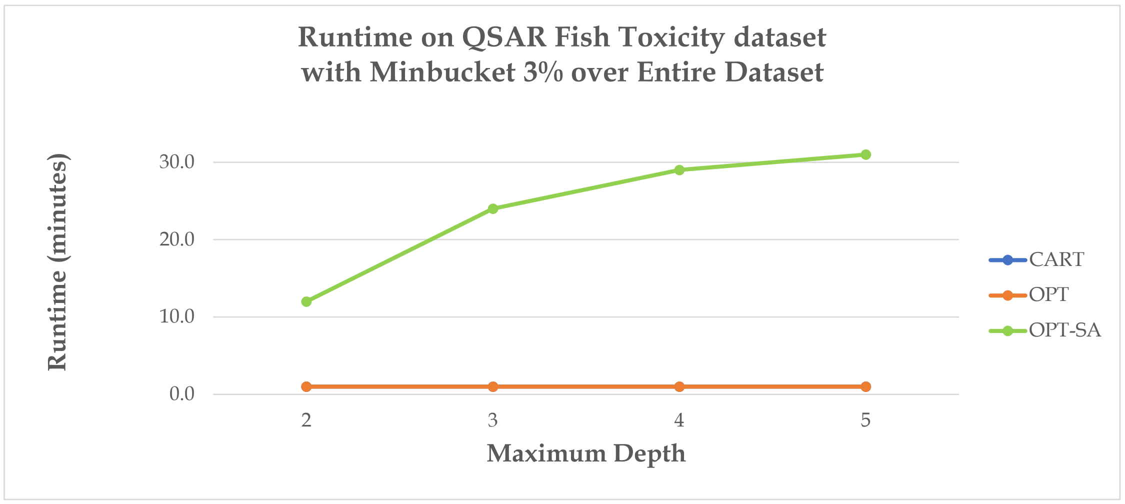

Figure 4.5: Runtime (in minutes) of QSAR fish toxicity dataset trained with maximum depth 2 to 5 and minbucket 3% over entire dataset.

runtime of OPT-SA is as high as 12 minutes at maximum depth 2, then rapidly increases to 24 minutes at maximum depth 3, while starting to stabilize at maximum depth 4 with 29 minutes and 31 minutes at maximum depth 5.

One of the main reasons underlying the long runtime of OPT-SA is the number of iterations that OPT-SA runs to optimize a tree. Unlike OPT that terminates whenever no improvement can be made for two consecutive rounds, the simulated annealing cooling schedule determines the number of iterations that OPT-SA runs. Even though OPT-SA may terminate early if no improvement can be made for two consecutive temperatures, it typically runs for more iterations and leads to optimization over more nodes than in OPT.

Another reason that OPT runs slower than CART is that we implemented OPT on Python programming language. Python is a widely-used and versatile programming language, which is appropriate to prototype our OPT-SA framework. However, the versatility of Python is a trade-off with speed. In order to significantly speed up model training, Julia and Cython, where OPT and CART are implemented on, can be good candidates.

4.4 Conclusion

Simulated annealing benefits decision tree construction and performs better than local search in reaching global optimality. The results were demonstrated on the 10 real-world datasets where OPT-SA achieved better out-of-sample mean rewards on 3 datasets compared to OPT and 4 datasets compared to CART. Even though performance gain was not the case for all 10 datasets, the results imply that there might be certain types of datasets that benefit more from our simulated annealing framework. We will conduct evaluations on additional datasets to prove or disprove this hypothesis.

Chapter 5

Optimal Survival Trees with Simulated Annealing (OST-SA)

To improve on predictive performance of publicly available survival tree algorithms, Bertsimas et al. [16] developed Optimal Survival Tree (OST). OST outperforms the existing survival tree algorithms, especially the well documented and user-friendly algorithms such as rpart [92] and ctree [50] in R packages [83]. However, OST utilizes local search like that of Optimal Classification Trees (OCT), which always picks the best solution locally, and may lead to a suboptimal global solution. Optimal Classification Trees with Simulated Annealing (OCT-SA) successfully improves on OCT by replacing local search with simulated annealing approach. Therefore, we also apply the same simulated annealing framework in this chapter to obtain Optimal Survival Tree with Simulated Annealing (OST-SA). We report computational results on 10 real-world datasets from SurvSet, an open-source time-to-event dataset repository. We train OST-SA and OST over the maximum depths ranging from 2 to 7. OST-SA successfully improves over OST for all maximum depths, and the results are statistically significant with p-values less than 0.05 at any maximum depths higher than 2. Although improvement was not found for all 10 datasets at the maximum depth 7, OST-SA achieves better out-of-sample local full likelihood scores on 8 out of 10 datasets compared to

OST.

5.1 Introduction

Optimal Survival Tree (OST) [16] involves censored data [40]. An example of application in this area is to predict survival time of patients where we have some censored data because we lose follow-up with some patients (so we do not know whether or not they are still alive) or some patients are still alive (so we do not know exactly when they will die) [59]. In order to predict survival time of censored data, we construct OST. The objective function of OST is a local full likelihood score used by LeBlanc and Crowley [62] which involves a cumulative hazard function with the Nelson-Aalen estimator. The curves in each OST node are the Kaplan-Meier curves which estimate the survival distribution. Evaluation of OST on a wide variety of both synthetic and real-world datasets shows improvement in accuracy when comparing with other existing survival tree-based methods like rpart and ctree in R, especially on large datasets.

Survival analysis involves the study of censored data, i.e., data with unknown time to event of interest [40]. The events of interest vary among different problem domains. In the medical domain, an event of interest could be a disease onset, a patient death, or a patient reaction to a treatment. In the reliability domain, an event of interest could be a hardware failure, or a software failure [57]. In the economics domain, an event of interest could be a massive unemployment, inflation, or bankruptcy [26, 37, 38]. In the sociology domain, an event of interest could be a war, a protest, or a crime [2, 3, 79]. The medical domain is one of the domains where survival analysis is heavily used [22, 41, 44, 63, 88]. To predict patients' survival time, we consider patients' death as events of interest which are binary events.

A survival distribution function is a widely-used technique in survival analysis with a binary event. First, we use the Nelson–Aalen estimator to generate a cumulative harzard function. In the case that a death is an event of interest, the Nelson–Aalen estimator can estimate a risk of death. Second, we use the Kaplan-Meier curve to plot a survival distribution function. At a particular time point, the Kaplan-Meier curve demonstrates the number of patients who survive. In order to make the Kaplan-Meier curve more interpretable, we can use survival trees to group subjects with a similar survival distribution function together.

A limited number of survival tree algorithms are publicly available, well documented, and user-friendly. Two of these algorithms are rpart and ctree in R packages. Even though rpart and ctree perform well on small datasets, they fail to scale on large datasets. To address the issue of scalability, Bertsimas et al. [16] developed OST. However, OST still uses the same decision tree construction framework like that of Optimal Classification Trees (OCT) where local search does not guarantee global optimality. Since Optimal Classification Trees with Simulated Annealing (OCT-SA) has successully improved on OCT by replacing local search with simulated annealing approach, in this paper, we also utilize the same simulated annealing framework to improve on OST and develop Optimal Survival Tree with Simulated Annealing (OST-SA).

OST-SA utilizes a similar approach as that of OCT-SA. The main difference is an objective value: OCT-SA uses a Gini impurity score while OST-SA uses a local full likelihood score or how well the survival time prediction fits the calculated Kaplan-Meier curve. In other words, OST-SA aims to minimize the error between its prediction and that of the Kaplan-Meier curve.

We make two main contributions in this paper. First, OST-SA is the first application of a simulated annealing algorithm on survival trees that guarantees to provide an optimal solution under an appropriate cooling schedule. Ying et al. [99] applied simulated annealing in survival analysis but on a non-tree-based model like a median regression model. Second, we report computational results on 10 real-world datasets from SurvSet, an open-source time-to-event dataset repository. These results improve upon the out-of-sample performance of OST while maintaining interpretability of each model.

We organizes this paper as follows. In section 5.2, we demonstrate how to apply simulated annealing in OST construction. The demonstration emphasizes the difference between OCT-SA and OST-SA. In Section 5.3, we evaluate OST-SA on 10 real-world datasets from SurvSet. In section 5.4, we make our concluding remarks.

5.2 Algorithms

OST-SA replaces local search in OST with simulated annealing to improve the predictive performance. Figure 5.1 outlines how the five algorithms in OST-SA interact with each other. Starting at Algorithm 11, it directs the main flow of the OST-SA algorithm. Algorithm 11 obtains six inputs: an initial survival tree T_1 as a random warm start tree; training data consisting of feature value X and event outcome y; estimated hazard coefficients hc; geometric temperature decay rate $c < 1$ of a cooling schedule; Markov chain length l or number of nodes to optimize per iteration; and neighbor search radius r or boundary of solution search space. Then Algorithm 11 produces one output, i.e., the optimal survival tree T_{best} which is the best tree from the series of transformations of the initial survival tree T_1. In order to derive the optimal survival tree T_{best}, Algorithm 11 calls Algorithm 12 to select a rank of candidate solutions to transform the current tree into. As the temperature of the simulated annealing cooling schedule decreases, OST-SA selects a better rank of candidate trees and Algorithm 13 is responsible for generating those candidate trees. Similar to OST transformation, there are three ways to transform a subtree in OST-SA: updating a parallel split, removing a left subtree, or removing a right subtree. Algorithm 13 calls Algorithm 14 to generate a new parallel split by iterating through all features and all feature values. At the last step, Algorithm 11 calls Algorithm 15 to determine a probability to transform the current subtree into a neighbor subtree based on the difference in local full likelihood scores of these two subtrees and the current temperature in the cooling schedule.

The mechanism of the simulated annealing algorithm is driven by the geometric cooling

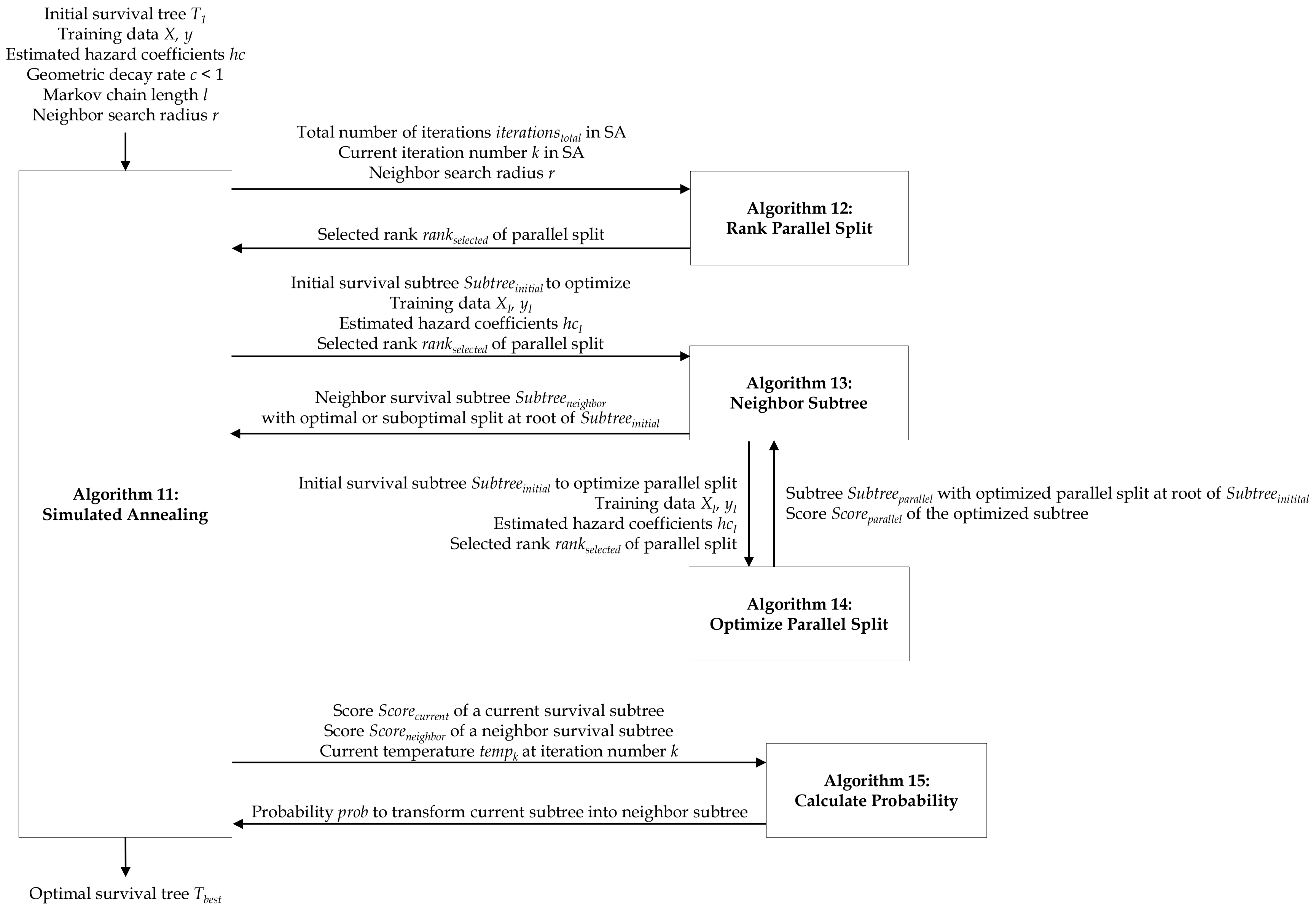
Initial survival tree T_1
Training data X, y
Estimated hazard coefficients hc
Geometric decay rate $c < 1$
Markov chain length l
Neighbor search radius r
Algorithm 11: Simulated Annealing
Optimal survival tree T_{best}
Total number of iterations $iterations_{total}$ in SA
Current iteration number k in SA
Neighbor search radius r
Algorithm 12: Rank Parallel Split
Selected rank $rank_{selected}$ of parallel split
Initial survival subtree $Subtree_{initial}$ to optimize
Training data X_I, y_I
Estimated hazard coefficients hc_I
Selected rank $rank_{selected}$ of parallel split
Algorithm 13: Neighbor Subtree
Neighbor survival subtree $Subtree_{neighbor}$
with optimal or suboptimal split at root of $Subtree_{initial}$
Initial survival subtree $Subtree_{initial}$ to optimize parallel split
Training data X_I, y_I
Estimated hazard coefficients hc_I
Selected rank $rank_{selected}$ of parallel split
Subtree $Subtree_{parallel}$ with optimized parallel split at root of $Subtree_{initital}$
Score $Score_{parallel}$ of the optimized subtree
Algorithm 14: Optimize Parallel Split
Score $Score_{current}$ of a current survival subtree
Score $Score_{neighbor}$ of a neighbor survival subtree
Current temperature $temp_k$ at iteration number k
Algorithm 15: Calculate Probability
Probability $prob$ to transform current subtree into neighbor subtree

Figure 5.1: OST-SA flowchart of Algorithm 1 to 5 demonstrating inputs, outputs, and sequences of execution.

schedule, the difference in energy between the current state and the neighboring state, and the acceptance probability as shown in Equations (5.1), (5.2), and (5.3):

$$temp_{k+1} = \alpha \cdot temp_k, \tag{5.1}$$

$$\Delta E = |Score_{current} - Score_{neighbor}|, \tag{5.2}$$

$$prob_k = e^{\frac{-\Delta E}{temp_k}}. \tag{5.3}$$

where

- $temp$ denotes temperature
- k denotes iteration number
- α denotes temperature decay rate
- ΔE denotes difference in energy between a current and a neighbor state
- $Score$ denotes local full likelihood score
- $prob$ denotes probability to transform a tree from a current state to a neighbor state
- e denotes the Euler's number equals 2.7183

From our empirical experiments, we found $\alpha = 0.95$ is an appropriate temperature decay rate for the 10 real-world datasets we used to evaluate OST-SA.

In addition to OCT-SA, we also applied the simulated annealing framework in Optimal Policy Tree (OPT) and obtained Optimal Policy Tree with Simulated Annealing (OPT-SA). OPT-SA successfully improved on OPT on certain types of datasets. OST-SA resembles OPT-SA in such a way that OST-SA needs to first calculate hazard coefficient hc while OPT-SA needs to first calculate rewards rw. However, it is simpler to estimate hazard coefficients hc than to estimate rewards rw. OST-SA uses the off-the-shelf Nelson-Aalen estimator to derive the cumulative hazard function and uses the Kaplan-Meier curves to estimate the survival distribution. There is no hyperparameter tuning involved in constructing the Nelson-Aalen estimator and the Kaplan-Meier curves. In contrast, OPT-SA uses counterfactual models like Random Forest to estimate rewards rw where hyperparameter tuning highly influences the quality of those counterfactual models. In addition, the local full likelihood scores, which are the objective values of OST-SA, have the same value range for all datasets. In contrast the mean rewards scores, which are the objective values of OPT-SA, have varied value ranges depending on the datasets. Accordingly, the procedure to scale the mean rewards score of each dataset to be in a similar range can significantly influence how well the objective values of OPT-SA fit with the design of the simulated annealing cooling schedule.

5.2.1 Overall Architecture of OST-SA in Algorithm 11: Simulated Annealing

Algorithm 11 gradually optimizes the initial survival tree structure T_1 and finally derives the best survival tree T_{best}. The cooling schedule starts with the highest temperature of 1 and geometrically decreases the temperature with the decay rate $c < 1$. It takes $iterations_{total}$ iterations for the temperature to fall below 0.01 and it is when the algorithm terminates. For each iteration number k, OST-SA transforms the survival tree over l nodes, i.e., Markov chain length, and obtains T_k or the survival tree at temperature $temp_k$. Following a typical simulated annealing mechanism, the probability to accept worse transformations is high at high temperatures, and vice versa.

As with OCT-SA and OPT-SA, we build OST-SA in Python version 3.8.15. Since OCT-SA and OPT-SA add OCT and OPT as warm start solutions, we also add OST as one of the warm start solutions for OST-SA. We use the OST package of Interpretable AI software [51] version 3.2.0 in Julia version 1.9.0 to make OST warm start solutions. In order to import those OST models into OST-SA as warm start solutions, we dumped those OST models into text files before importing them into OST-SA. We use a Windows laptop and also MIT Supercloud [84] to run OST-SA. Instead of using CART as a warm start solution

Algorithm 11 SimulatedAnnealing

```
    Input: Initial survival tree $T_1$; Training data $X, y$; Estimated hazard coefficients $hc$;
           Geometric decay rate $c < 1$; Markov chain length $l$; Neighbor search radius $r$
    Output: Optimal survival tree $T_{best}$
 1: $temp_0 \leftarrow 1$                                   ▷ $temp_0$ is initial temperature
 2: $iterations_{total} \leftarrow k_{min} - 1$ where $temp_0 \cdot c^k < 0.01$
 3: $k \leftarrow 1$                                        ▷ k is iteration number
 4: $T_{best} \leftarrow T_1$                               ▷ $T_{best}$ keeps the best tree
 5: repeat
 6:     $temp_k \leftarrow temp_0 \cdot c^k$                ▷ Temperature based on geometric cooling schedule
 7:     $rank_k \leftarrow$ RankParallelSplit($iterations_{total}, k, r$)
 8:     $nodes_{optimized} \leftarrow \emptyset$            ▷ Keep track of optimized nodes at current temperature
        ▷ Get a set of all current split nodes or leaf nodes in $T_k$ eligible to be split further
 9:     $nodes_{unoptimized} \leftarrow$ SplitNodes($T_k, X, y, rw$)
10:     repeat
11:         $nodes_{unoptimized} \leftarrow nodes_{unoptimized} - nodes_{optimized}$
12:         $node_{random} \leftarrow$ RandomNode($nodes_{unoptimized}$)        ▷ Pick one random node
13:         $nodes_{unoptimized} \leftarrow nodes_{unoptimized} - \{node_{random}\}$
14:         $nodes_{optimized} \leftarrow nodes_{optimized} \cup \{node_{random}\}$

            ▷ Find subtree of the random node and calculate score
15:         $Subtree_{random} \leftarrow$ Subtree of $T_k$ where $node_{random}$ is root
16:         $I \leftarrow \{i\colon x_i$ is assigned to a leaf contained in $Subtree_{random}\}$
17:         $Score_{current} \leftarrow$ Score($Subtree_{random}, X_I, y_I, hc_I$)
            ▷ Find a neighbor of the random subtree and calculate score
18:         $NeighborSubtree_{random} \leftarrow$ NeighborSubtree($Subtree_{random}, X_I, y_I, hc_I, rank_k$)
19:         $Score_{neighbor} \leftarrow$ Score($NeighborSubtree_{random}, X_I, y_I, hc_I$)

            ▷ Determine probability to transform the current subtree to its neighbor state
20:         $prob \leftarrow$ CalculateProbability($Score_{current}, Score_{neighbor}, temp_k$)
21:         if $prob \geq$ RandomNumber($0.1, 1$) then
22:             $T_k \leftarrow$ Replace $Subtree_{random}$ in $T_k$ with $NeighborSubtree_{random}$
23:             if Score($T_k, X, y, hc$) < Score($T_{best}, X, y, hc$) then
24:                 $T_{best} \leftarrow T_k$               ▷ Update the best tree if score of $T_k$ is lower
25:             end if
26:         end if
27:     until $nodes_{unoptimized} = \emptyset$
28:     if $|nodes_{optimized}| = 0$ then         ▷ No node can be optimized at current temperature
        ▷ Move to the last iteration of the current temperature band
29:         $k \leftarrow k_{max}$ where RankParallelSplit($iterations_{total}, k, r$) returns $rank_k$
30:     end if
31:     $k \leftarrow k + 1$
32: until $k = iterations_{total}$
33: return $T_{best}$
```

like that of OCT-SA and OPT-SA, we use sksurv, a survival tree from scikit-survival which is a Python module for survival analysis expanded from scikit-learn [80], as one of our warm start solutions.

5.2.2 Algorithm 12: Rank Parallel Split

Algorithm 12 RankParallelSplit

Input: Total number of iterations $iterations_{total}$ in simulated annealing;
Current iteration number k in simulated annealing;
Neighbor search radius r

Output: Selected rank $rank_{selected}$ of parallel split to be used in Algorithm **??**

```
1: $rank_{selected} \leftarrow 1$                          ▷ 1 is the best rank
2: $rank_{set} \leftarrow \{r, r-1, r-2, ..., 1\}$          ▷ Set of all ranks in neighbor search radius $r$
3: for $i$ in $rank_{set}$ do
4:     if $k < \frac{iterations_{total}}{i}$ then
5:         $rank_{selected} \leftarrow i$
6:         break
7:     end if
8: end for
9: return $rank_{selected}$
```

Algorithm 12 in OST-SA performs exactly the same task as Algorithm 2 in OCT-SA and in OPT-SA where it assigns ranks to candidate trees and selects a specific rank based on the current simulated annealing temperature. Algorithm 12 obtains three inputs from Algorithm 11: $iterations_{total}$ representing the total number of iterations that OST-SA runs on different temperatures; k representing the current iteration number; and r representing the neighbor search radius whose the optimal value OST-SA derives from hyperparameter tuning. Next, Algorithm 12 outputs a rank of candidate solutions into which OST-SA will transform the current subtree. Unlike local search in OST, OST-SA does not always transform the current subtree into a subtree with the best objective value. If the neighbor search radius $r = 3$, OST-SA will pick the third best candidate solution for the current subtree to transform into at the start of the annealing schedule. As temperatures decrease, OST-SA will select better candidate solution, i.e, the second-best solution and eventually the first-best solution. The benefit of this candidate state ranking compared to the random candidate state selection, which is in a typical simulated annealing, is a faster convergence rate to the global optimal solution. In addition, this procedure can prevent overshooting.

The detailed implementation step of the candidate state ranking is shown in Algorithm 12. If r is the search radius, Algorithm 12 then divides temperature of the entire simulated annealing cooling schedule into r temperature bands: the 1^{st}, the 2^{nd}, ..., and the r^{th} temperature band covering the highest to the lowest temperatures. OST-SA maps a iteration number to the end of each temperature band with a series $\frac{S}{r}, \frac{S}{r-1}, \frac{S}{r-2}, ..., \frac{S}{2}, S$ where $iterations_{total} = S$ and r is the number of temperature bands. We derive this mapping from the empirical experiments on the 10 datasets that we evaluate our OST-SA algoritm on and it is exactly the same mapping used in OCT-SA and OPT-SA. Assuming we have $iterations_{total} = 90$ and the search radius $r = 3$, we can divide the entire cooling schedule into 3 temperature bands: iteration 1 to 30 where OST-SA selects the 3^{rd} best solution, iteration 31 to 45 where OST-SA selects the 2^{nd} best solution, and iteration 46 to 90 where OST-SA selects the 1^{st} best solution.

5.2.3 Algorithm 13: Neighbor Subtree

Algorithm 13 selects a neighboring subtree into which the current subtree will transform. Algorithm 11 passes four inputs into Algorithm 13: an initial survival subtree $Subtree_{initial}$ that OST-SA will optimize; training data X_I, y_I or training data X, y in the current subtree that OST-SA will optimize; estimated hazard coefficients hc_I of the current subtree; and $rank_{selected}$ denoting the selected rank of candidate solution from Algorithm 12. After that, Algorithm 13 outputs a survival subtree $Subtree_{neighbor}$, i.e., a transformation of the initial survival subtree which may be an optimal or suboptimal transformation at the root node of $Subtree_{initial}$. As in OST, OST-SA can update the current subtree in three ways: updating the parallel split, pruning the left subtree, or pruning the right subtree. In terms of the update to the parallel split when $rank_{selected} = r$, the r^{th} best transformation will take

Algorithm 13 NeighborSubtree

Input: Initial survival subtree $Subtree_{initial}$ to optimize;
Training data X_I, y_I;
Estimated hazard coefficients hc_I;
Selected rank $rank_{selected}$ of parallel split

Output: Neighbor survival subtree $Subtree_{neighbor}$ with optimal or suboptimal split at root of $Subtree_{initial}$

```
    ▷ Optimize parallel split of initial subtree and calculate score
 1: $Subtree_{parallel}, Score_{parallel} \leftarrow$ OptimizeParallelSplit($Subtree_{initial}, X_I, y_I, hc_I,$
                                                  $rank_{selected}$)

    ▷ 1 is the worst local full likelihood score
 2: $Score_{lower} \leftarrow 1$
 3: $Score_{upper} \leftarrow 1$

    ▷ Replace initial subtree with its lower or upper subtree and calculate score
 4: if $Subtree_{initial}$ is non-leaf node then
 5:     $Subtree_{lower} \leftarrow$ Lower subtree of $Subtree_{initial}$
 6:     $Score_{lower} \leftarrow$ Score($Subtree_{lower}, X_I, y_I, hc_I$)

 7:     $Subtree_{upper} \leftarrow$ Upper subtree of $Subtree_{initial}$
 8:     $Score_{upper} \leftarrow$ Score($Subtree_{upper}, X_I, y_I, hc_I$)
 9: end if

    ▷ Determine neighbor subtree with the local full likelihood score (a lower score is better)
10: if ($Score_{parallel} \leq Score_{lower}$) and ($Score_{parallel} \leq Score_{upper}$) then
11:     $Subtree_{neighbor} \leftarrow Subtree_{parallel}$
12: else
13:     if $Score_{lower} \leq Score_{upper}$ then
14:         $Subtree_{neighbor} \leftarrow Subtree_{lower}$
15:     else
16:         $Subtree_{neighbor} \leftarrow Subtree_{upper}$
17:     end if
18: end if
19: return $Subtree_{neighbor}$
```

place.

5.2.4 Algorithm 14: Optimize Parallel Split

Algorithm 14 in OST-SA performs the same task as that of OCT-SA and OPT-SA where OST-SA updates a parallel splits by going through all features and all branching values, and selects the new parallel split that is the r^{th} best option where $rank_{selected} = r$. Similar to Algorithm 13, Algorithm 14 accepts four inputs: an initial survival subtree $Subtree_{initial}$ that OST-SA will optimize; training data X_I, y_I corresponding to the training data X, y in the current subtree that OST-SA will optimize; estimated hazard coefficients hc_I; and selected rank $rank_{selected}$ whose value determined by Algorithm 12. In the case of $rank_{selected} = r$, Algorithm 14 outputs the subtree $Subtree_{parallel}$, which is the transformation of the root node of the subtree $Subtree_{initial}$ with the r^{th} best parallel split, and the corresponding local full likelihood score $Score_{parallel}$ of the subtree $Subtree_{parallel}$.

To go over all possible branching values of each feature, Algorithm 14 rearranges all unique feature values in ascending order and find midpoints between two consecutive feature values. Algorithm 14 stores only a set of r best features and branching values. Whenever there are multiple parallel split options resulting in the same objective value, Algorithm 14 maintains only the first option and discard all subsequent options.

5.2.5 Algorithm 15: Calculate Probability

Since OST-SA allows a probabilistic transformation to a worse neighboring state, Algorithm 15 calculates such probability based on three inputs: the local full likelihood score $Score_{current}$ of the current survival subtree; the local full likelihood score $Score_{neighbor}$ of the neighbor survival subtree; and the current temperature $temp_k$ at the current iteration number k. Then Algorithm 15 outputs the probability $prob$ for OST-SA to transform the current subtree with the local full likelihood score of $Score_{current}$ into the worse neighboring state with the local full likelihood score of $Score_{neighbor}$. In order to calculate the probability of worse transformation,

Algorithm 14 OptimizeParallelSplit

```
Input: Initial survival subtree $Subtree_{initial}$ to optimize parallel split;
       Training data $X_I, y_I$;
       Estimated hazard coefficients $hc_I$;
       Selected rank $rank_{selected}$ of parallel split
Output: Subtree $Subtree_{parallel}$ with optimized parallel split at root of $Subtree_{initial}$
        Score $Score_{parallel}$ of the optimized subtree

1: $n' \leftarrow$ Number of samples in $X_I$
2: $p' \leftarrow$ Number of features in $X_I$
3: $candidates \leftarrow \emptyset$

   ▷ Iterate through all $p'$ features
4: for $v = 1, 2, 3, ..., p'$ do
5:     $featureValues \leftarrow \{X_{I_{uv}} : u = 1, 2, 3, ..., n'\}$
6:     $featureValues \leftarrow$ Sort $featureValues$ in ascending order
7:     $f \leftarrow$ Number of unique feature values in $featureValues$
8:     $branchValues \leftarrow \{\frac{1}{2}(featureValues_m + featureValues_{m+1}) : m = 1, 2, 3, ..., f - 1\}$

       ▷ Iterate through all branch values of feature $v$
9:     for $m = 1, 2, 3, ..., f - 1$ do
10:

           ▷ Update split at root node and calculate score
11:        $Subtree_{optimized} \leftarrow$ Split root of $Subtree_{initial}$ with feature $v$ and branch value $m$
12:        $Score_{optimized} \leftarrow \textsc{Score}(Subtree_{optimized}, X_I, y_I, hc_I)$

           ▷ Keep only one subtree for each unique score
13:        if no subtree in $candidates$ with score equals $Score_{optimized}$ then
14:            $candidates \leftarrow candidates \cup \{(Subtree_{optimized}, Score_{optimized})\}$
15:        end if
16:    end for
17: end for

   ▷ Pick final parallel optimized subtree based on score ranking
18: $candidates_{sorted} \leftarrow$ Sort $candidates$ by $Score_{optimized}$ score from best to worst
19: $Subtree_{parallel}, Score_{parallel} \leftarrow$ Pick subtree and score at $rank_{selected}$ from $candidates_{sorted}$
20: return $Subtree_{parallel}, Score_{parallel}$
```

Algorithm 15 CalculateProbability

Input: Score $Score_{current}$ of a current survival subtree;
Score $Score_{neighbor}$ of a neighbor survival subtree;
Current temperature $temp_k$ at iteration number k

Output: Probability $prob$ to transform current subtree into neighbor subtree

```
1: $prob \leftarrow 0$          ▷ No subtree transformation if $Score_{neighbor} = Score_{current}$
2: if $Score_{neighbor} < Score_{current}$ then          ▷ A lower score is better
3:     $prob \leftarrow 1$          ▷ Always transform to neighbor subtree with better score
4: else
       ▷ Calculate probability from difference in energy and current temperature
5:     if $Score_{neighbor} > Score_{current}$ then
6:         $\Delta E \leftarrow |Score_{current} - Score_{neighbor}|$          ▷ Difference in energy
7:         $prob \leftarrow e^{\frac{-\Delta E}{temp_k}}$          ▷ Euler's number e = 2.7183
8:     end if
9: end if
```

we first calculate the difference in energy ΔE as shown Equation (5.2), i.e., the absolute difference between the local full likelihood score $Score_{current}$ of the current subtree and the local full likelihood score $Score_{neighbor}$ of the neighbor subtree. Then we follow the calculation shown in Equation (5.3) to obtain the probability of worse transformation $prob$ from the difference in energy ΔE and the current temperature $temp_k$. In contrast, we always allow a better transformation from the current subtree into a neighboring state with a better objective value and we enforce this mechanism by setting the probability $prob$ of state transformation to 1 for this scenario. For the case that the local full likelihood score $Score_{current}$ of the current subtree equals the local full likelihood score $Score_{neighbor}$ of the neighboring subtree, we never make a transformation between states with the same objective values and we enforce this mechanism by setting the probability $prob$ of state transformation to 0.

5.3 Results on Real-World Datasets and Discussion

5.3.1 Experimental Setup

We evaluate OST-SA on 10 datasets from SurvSet, an open-source time-to-event dataset repository [29]. These datasets have number of data points in a range from 137 to 3371, and number of features in a range from 3 to 80. All datasets demonstrate binary events but the time to event may have different time units, e.g., days, months, years etc.

To separate data points of each dataset into train, validation, and test set, we allocate the ratio of train:validation:test being 50:25:25. In order to avoid overfitting and obtain interpretable models that can still capture meaningful information from each dataset, we set 10% over each entire dataset as minbucket. Since the combination of train and validation dataset accounts for 75% of the entire dataset, the highest survival tree is at maximum depth 7. For these reasons, we train the models with maximum depths ranging from 2 to 7.

We use the Nelson-Aalen estimator from the OST package of Interpretable AI software [51] to estimate a hazard coefficient for each data point. To make a fair performance comparison, we train sksurv, OST, and OST-SA using the same estimated hazard coefficients of each dataset. As mentioned in Section 5.2, sksurv is a survival tree from scikit-survival which is a Python module for survival analysis expanded from scikit-learn [80]. We scale the local full likelihood score to be in the range from 0 to 1, the same range as the objective value of OCT-SA. We obtain the scaling factor by calculating the ratio of the local full likelihood score of the final tree in comparison with a baseline tree. A baseline tree is a tree with only one node and it predicts the same survival time for all patient. Accordingly, we can apply the same combination of hyperparameters as that of OCT-SA in OST-SA, i.e., Markov chain length $l = 2$ and search radius $r = 3$, and only need to tune the complexity penalty γ.

Similar to that of OCT-SA and OPT-SA, after obtaining the best hyperparameter over the validation dataset, we retrain the model over the combination of the validation and the training dataset. Out of the 103 warm starts (sksurv, OST, a single node, and 100 random warm starts), we pick the tree with the best objective value as the OST-SA. Then we evaluate the OST-SA on the unseen test dataset and report out-of-sample performance in comparison to OST-SA and sksurv models. To ensure that our results are statistically significant, we run the same experiments over 5 different train:validation:test data splits on each dataset and calculate p-values, based on the average scores and the standard deviations.

5.3.2 OST-SA vs OST vs sksurv

As shown in Table 5.1, OST-SA outperforms OST at all maximum depths ranging from 2 to 7, and also outperforms sksurv at all maximum depths except depth 3. At each maximum depth, we average the local full likelihood scores over all the 10 datasets and 5 data splits. In terms of p-values, the mean improvements of OST-SA over OST are statistically significant with p-values less than 0.05 at any maximum depths higher than 2. This trend indicates strong improvements of OST-SA over OST when models are complicate enough to capture meaningful signal within each dataset. In contrast, the mean improvements of OST-SA over sksurv are not statistically significant and the magitude of improvements are much lower than the improvement of OST-SA over OST. This trend demonstrates that sksurv is a strong algorithm.

Figure 5.2 plots mean out-of-sample local full likelihood scores of sksurv, OST, and OST-SA from Table 5.1. Table 5.2 shows average results over 5 data splits of each dataset at maximum depth 7, i.e., the highest survival tree depth in our experiments. The results demonstrate that OST-SA outperforms OST on 8 out of 10 datasets, and also outperforms sksurv on 6 out of 10 datasets.

From Table 5.2, OST-SA of the unemployment dataset outperforms both the OST and sksurv model. The unemployment dataset is a four-month panel of revised Current Population Survery data regarding unemployment conducted by the Bureau of Labor Statistics in 1993 [85]. There are 450 data points in this dataset and each data sample has 5 features: race

Maximum Depth	Mean out-of-sample local full likelihood score			Mean improvement of OST-SA over		*p*-value	
	sksurv	OST	OST-SA	OST	sksurv	OST	sksurv
2	0.8876	0.9008	**0.8772**	**0.0236** $\pm$ 0.0137	0.0104 $\pm$ 0.0104	**0.0628**	0.2066
3	0.8633	0.9053	**0.8636**	**0.0416** $\pm$ 0.0285	-0.0004 $\pm$ 0.0101	**0.0329**	0.9723
4	0.8616	0.9042	**0.8564**	**0.0478** $\pm$ 0.0417	0.0051 $\pm$ 0.0179	**0.0258**	0.7034
5	0.8614	0.9041	**0.8587**	**0.0454** $\pm$ 0.0334	0.0027 $\pm$ 0.0104	**0.0269**	0.8162
6	0.8614	0.9048	**0.8608**	**0.0440** $\pm$ 0.0319	0.0006 $\pm$ 0.0131	**0.0271**	0.9558
7	0.8614	0.9048	**0.8609**	**0.0439** $\pm$ 0.0301	0.0005 $\pm$ 0.0106	**0.0264**	0.9611

Table 5.1: Average out-of-sample local full likelihood score (a lower score is better) averaged across 10 datasets from SurvSet, an open-source time-to-event dataset repository, trained with maximum depth 2 to 7 and minbucket 10% over entire dataset. Mean improvement and p-value are calculated between OST-SA vs OST and OST-SA vs sksurv. A positive mean improvement indicates OST-SA having better performance than OST or sksurv.

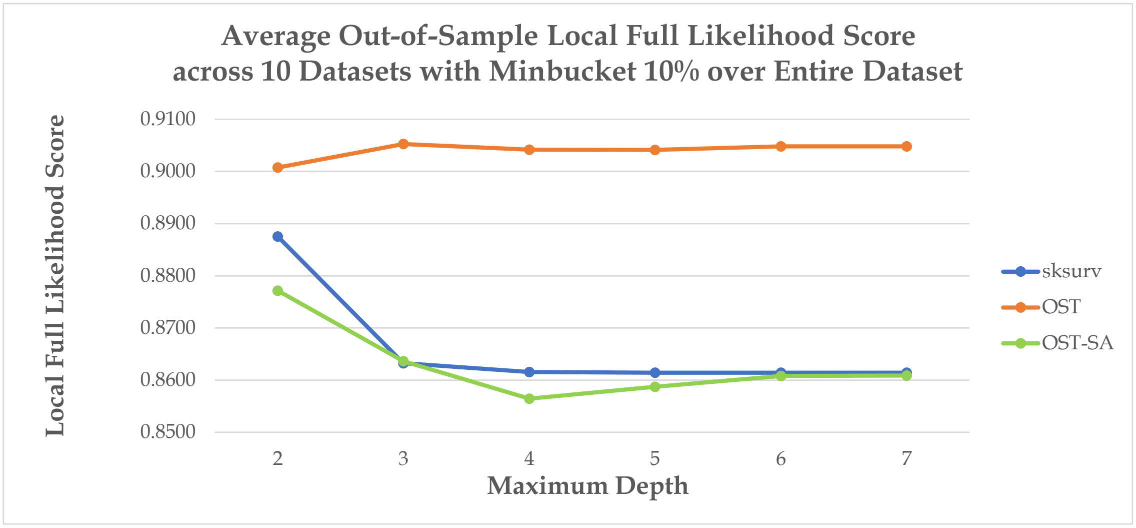

Figure 5.2: Average out-of-sample local full likelihood score (a lower score is better) averaged across 10 datasets from SurvSet, an open-source time-to-event dataset repository, trained with maximum depth 2 to 7 and minbucket 10% over entire dataset.

(white vs non-white), sex (male vs female), reason (one of the 3 reasons of unemployment), search (one of the 2 types of unemployment), and public employment (usage of a public employment agency). As shown in Figures 5.3, 5.4, and 5.5 where all models are trained on the same data split at maximum depth 7 and minbucket 10%, OST-SA achieves the best out-of-sample performance of 0.974 (a lower score is better) and also has the most complex tree structure with 4 split nodes. By comparing OST-SA with sksurv model which is the second-best model, both survival trees select the same set of 3 features: public employment,

No.	Dataset	n	p	Mean out-of-sample of local full likelihood score			Mean improvement of OST-SA over	
				sksurv	OST	OST-SA	OST	sksurv
1	aids	1151	11	0.8874	0.9601	**0.9090**	**0.0511** ± 0.0904	**-0.0216** ± 0.0464
2	breast cancer	198	80	0.8537	0.9629	**0.8716**	**0.0913** ± 0.0399	**-0.0179** ± 0.0665
3	diabetes	394	4	0.9126	0.9292	**0.9349**	**-0.0057** ± 0.0112	**-0.0223** ± 0.0120
4	divorce	3371	3	0.9936	0.9955	**0.9932**	**0.0023** ± 0.0032	**0.0005** ± 0.0010
5	german breast cancer study group 2	686	8	0.9056	0.8955	**0.8798**	**0.0157** ± 0.0174	**0.0258** ± 0.0262
6	mgus2	1338	5	0.8294	0.8127	**0.8170**	**-0.0044** ± 0.0111	**0.0123** ± 0.0242
7	unemployment	450	5	0.9779	0.9871	**0.9586**	**0.0285** ± 0.0234	**0.0193** ± 0.0204
8	veterans lung cancer	137	6	0.7076	0.8897	**0.7122**	**0.1774** ± 0.0997	**-0.0046** ± 0.0309
9	worcester heart attack study	500	14	0.7594	0.7895	**0.7529**	**0.0367** ± 0.0769	**0.0066** ± 0.0363
10	zinc	431	13	0.7870	0.8261	**0.7796**	**0.0465** ± 0.0595	**0.0074** ± 0.0356

Table 5.2: Out-of-sample local full likelihood score (a lower score is better) on 10 datasets from SurvSet, an open-source time-to-event dataset repository, trained with maximum depth 7 and minbucket 10% over entire dataset where n and p denote the number of data points and the number of features of each dataset, respectively. A positive mean improvement indicates OST-SA having better performance than OST or sksurv.

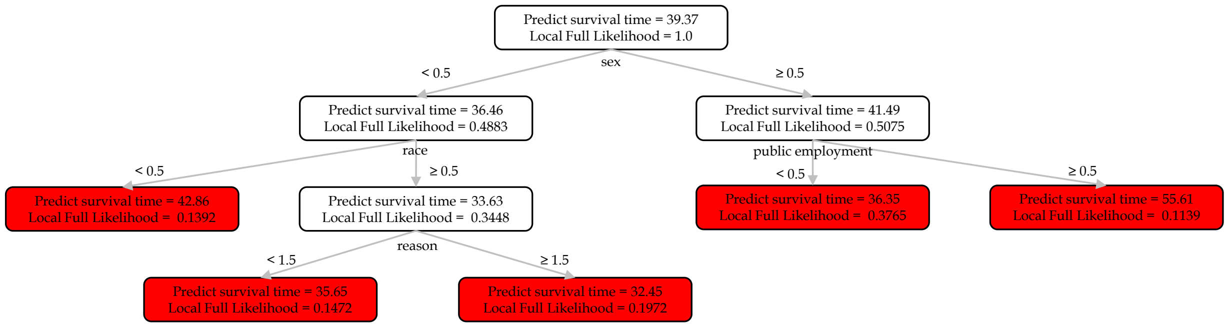
Predict survival time = 39.37
Local Full Likelihood = 1.0
sex
< 0.5
≥ 0.5
Predict survival time = 36.46
Local Full Likelihood = 0.4883
Predict survival time = 41.49
Local Full Likelihood = 0.5075
race
public employment
< 0.5
≥ 0.5
Predict survival time = 42.86
Local Full Likelihood = 0.1392
Predict survival time = 33.63
Local Full Likelihood = 0.3448
< 0.5
≥ 0.5
Predict survival time = 36.35
Local Full Likelihood = 0.3765
Predict survival time = 55.61
Local Full Likelihood = 0.1139
reason
< 1.5
≥ 1.5
Predict survival time = 35.65
Local Full Likelihood = 0.1472
Predict survival time = 32.45
Local Full Likelihood = 0.1972

Figure 5.3: OST-SA of unemployment dataset trained with maximum depth 7 and minbucket 10% over entire dataset. The OST-SA model has 4 splits and achieves out-of-sample local full likelihood score (a lower score is better) of 0.974.

Predict survival time = 39.37
Local Full Likelihood = 1.0

Figure 5.4: OST of unemployment dataset trained with maximum depth 7 and minbucket 10% over entire dataset. The OST model has only one split and achieves out-of-sample mean local full likelihood score (a lower score is better) of 1.

reason, and sex. However, OST-SA selects an additional feature, i.e., race and this feature helps improve the out-of-sample local full likelihood score from 0.9746 in OST to 0.974 in OST-SA. This points out that simulated annealing framework is beneficial in identifying a meaningful split that sksurv was unable to capture. In contrast, OST performs worse with the local full likelihood score of 1 and the survival tree only has one node. This is possibly

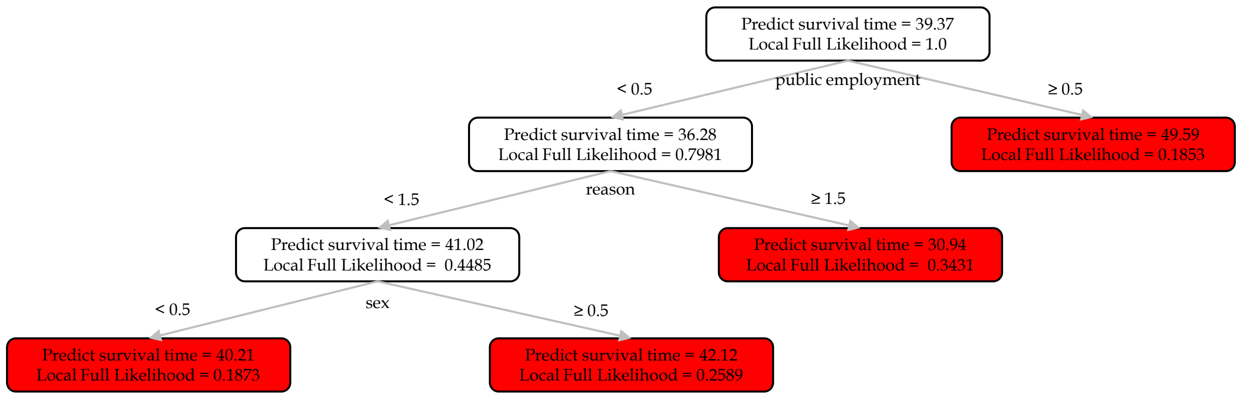
Predict survival time = 39.37
Local Full Likelihood = 1.0
public employment
< 0.5
≥ 0.5
Predict survival time = 36.28
Local Full Likelihood = 0.7981
Predict survival time = 49.59
Local Full Likelihood = 0.1853
reason
< 1.5
≥ 1.5
Predict survival time = 41.02
Local Full Likelihood = 0.4485
Predict survival time = 30.94
Local Full Likelihood = 0.3431
sex
< 0.5
≥ 0.5
Predict survival time = 40.21
Local Full Likelihood = 0.1873
Predict survival time = 42.12
Local Full Likelihood = 0.2589

Figure 5.5: sksurv survival tree of unemployment dataset trained with maximum depth 7 and minbucket 10% over entire dataset. The sksurv model has 3 splits and achieves out-of-sample local full likelihood score (a lower score is better) of 0.9746.

due to local search of OST is trapped in local minimas.

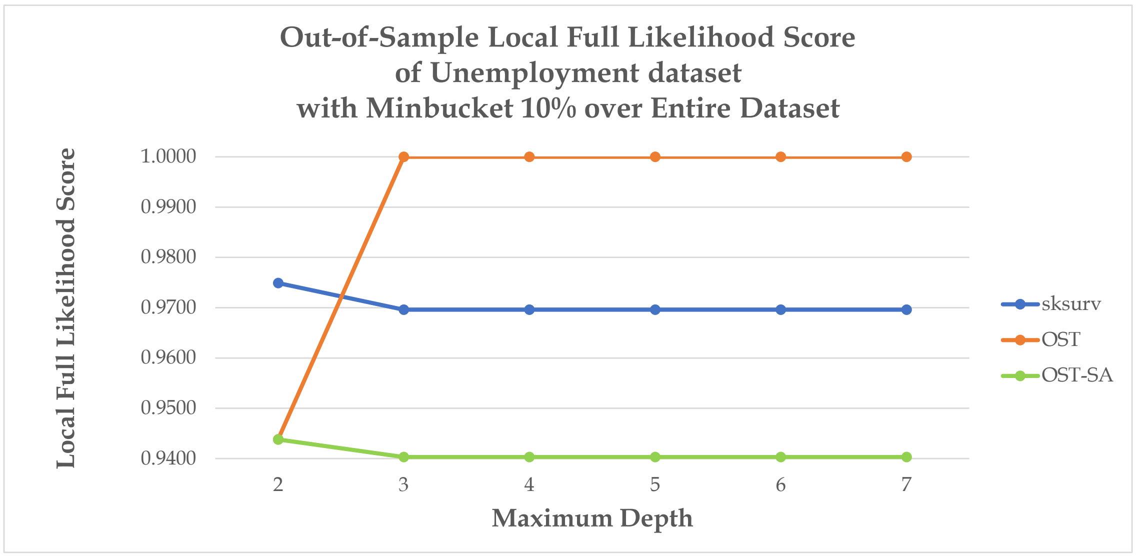

Figure 5.6: Out-of-sample local full likelihood score (a lower score is better) on unemployment dataset trained with maximum depth 2 to 7 and minbucket 10% over entire dataset.

As maximum depths increase, the performance improves while runtimes also increase as demonstrated in Figures 5.6 and 5.7. We use the same data split as that of the OST-SA, OST, and sksurv models in Figures 5.3, 5.4, and 5.5 to measure these performance and runtime trade-offs. The performance and runtime of OST-SA, OST and sksurv follow the same trend where they only differ between maximum depth 2 and 3. As the performance and runtime

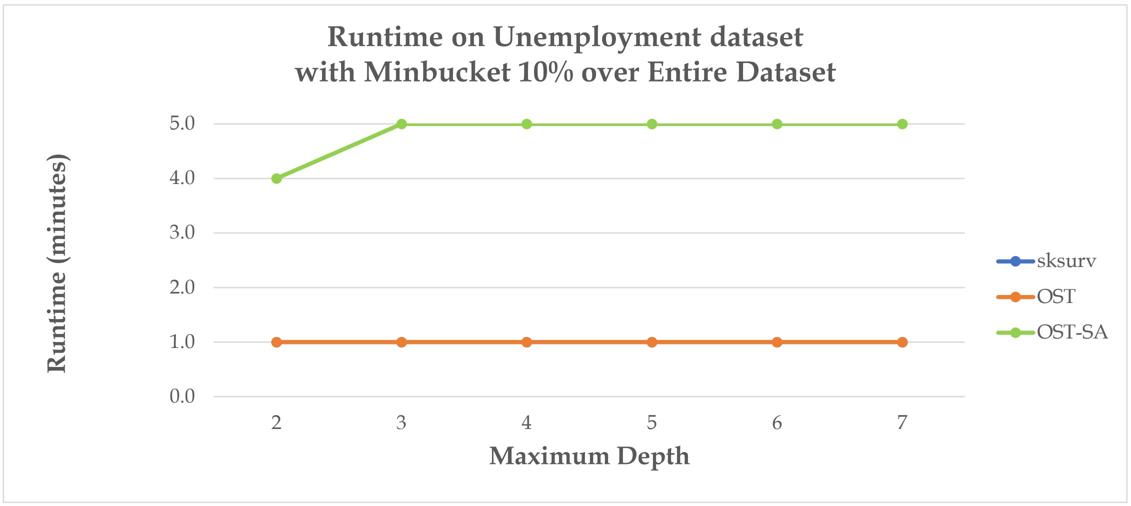

Figure 5.7: Runtime (in minutes) of unemployment dataset trained with maximum depth 2 to 7 and minbucket 10% over entire dataset.

stabilize at maximum depth 3, they stay the same from depth 3 to 7. This trend indicates that the survival trees already capture all meaningful splits at maximum depth 3. In terms of performance, OST-SA performs best, followed by sksurv and OST. Regarding runtime, sksurv and OST spend approximately 1 minute at all maximum depths while OST-SA is a lot slower and requires 4 to 5 minutes for model training. This is possibly due to a much larger number of warm start solutions that OST-SA uses in model training when compared to a single model training in sksurv. In addition, OST-SA also runs for a larger number of iterations to optimize a tree as determined by the cooling schedule compared to that of OST, which stops whenever no improvement can be made from 2 consecutive iterations.

5.4 Conclusion

Simulated annealing benefits decision tree construction and performs better than local search in reaching global optimality. The results were demonstrated on the average local full likelihood scores over the 10 real-world datasets. OST-SA achieved better out-of-sample local full likelihood score than that of OST at all maximum depths. Specifically, the results are statistically significant with p-values less than 0.5 at any maximum depths higher than 2.

Chapter 6

Case Studies

To demonstrate versatility of optimal decision trees with the simulated annealing algorithm, we conduct experiments on datasets in the medical domain: OCT-SA on the sarcoma dataset, OPT-SA on the Gastrointestinal Stromal Tumor (GIST) dataset, and OST-SA on the sarcoma dataset (the same dataset as that of the OCT-SA).

6.1 Classification Analysis for Sarcoma

6.1.1 Experimental Setup

To showcase the OCT-SA algorithm in the medical domain, we construct a model to predict whether or not patients with sarcoma [90] will die from this disease within 5 years after surgery [70]. This problem can be formulated as a binary classification problem that predicts patients' mortality status: alive or dead. We obtain this sarcoma dataset from Memorial Sloan Kettering Cancer Center (MSKCC). The dataset contains data of patients whose surgeries (definitive surgery for primary extremity and truncal liposarcoma (LPS) [24, 25, 42]) were performed at MSKCC between July 1982 and October 2017. We exclude any patients who are still alive and follow-up times are less than 5 years. Accordingly, 596 patients are eligible to be included in this 5-year mortality prediction.

We use 17 features of each patient in this analysis: 2 continuous features (age [34], and maximum tumor size [43]), and 15 binary features (gender [100], 3 tumor sites[1] [39, 60], tumor depth [87], bone invasion [33], nerve invasion [75], 5 histologic subtypes[2] [23], and 3 surgical margins[3] [31]). For any non-numeric features, we convert their feature values into numeric for feasibility with the OCT-SA algorithm. We split the dataset into training, validation, and testing dataset with the ratio of train:validation:test equals 70:15:15. We use 35 patients as minbucket and train a model with maximum depth 5. We decide this combination of hyperparameters based on clinical intuition from our medical collaborators. Similar to the evaluation on the 61 real-world benchmarking datasets, we also use Markov chain length $l = 2$ and search radius $r = 3$ when training OCT-SA. Then we tune only complexity penalty γ. We also train OCT and CART as our baseline models to compare with OCT-SA.

6.1.2 OCT-SA vs OCT vs CART

The OCT-SA, OCT, and CART of the sarcoma dataset, where we trained at maximum depth 5 and minbucket 35 patients, are shown in Figures 6.1, 6.2, and 6.3. By comparing out-of-sample performance, OCT-SA achieves the lowest Gini impurity score (a lower score is better) of 0.1600, followed by OCT whose score is 0.1686, and CART is the worst model with the score of 0.1698. In terms of tree complexity, OCT-SA has the highest complexity with 8 splits, followed by CART with 7 splits, and OCT is the simplest model with 4 splits. The out-of-sample Gini impurity scores and tree complexities indicate that OCT is a strong algorithm compared to CART, as 4 meaningful splits in OCT outperforms 7 splits in CART. Even though OCT is a strong algorithm, OCT-SA can still outperform OCT. This performance improvement reaffirms that probabilistically allowing bad transformations in simulated annealing is definitely effective. Regarding feature selection, all the 3 models,

[1]The 3 tumor sites are 3 binary features: lower extremity (y/n), trunk (y/n), and upper extremity (y/n).

[2]The 5 histologic subtypes are 5 binary features: dedifferentiated (y/n), myxoid (y/n), pleomorphic (y/n), round cell (y/n), and well differentiated (y/n)

[3]The 3 surgical margins are 3 binary features: resection R1 (y/n), R2 (y/n), and R3 (y/n).

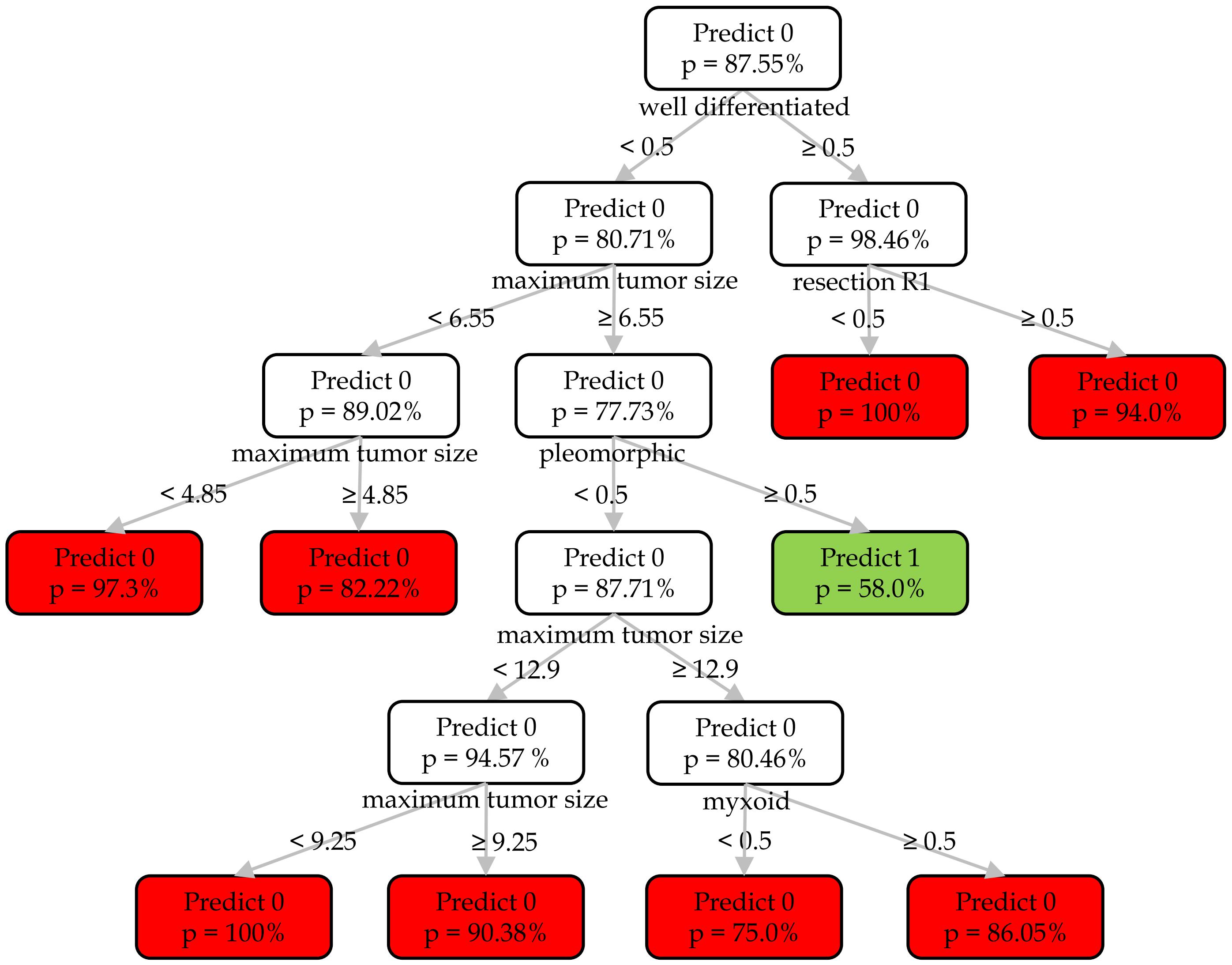

Figure 6.1: OCT-SA of sarcoma dataset trained with maximum depth 5 and minbucket 35 patients. The OCT-SA model has 8 splits and achieves out-of-sample Gini impurity score (a lower score is better) of 0.1600.

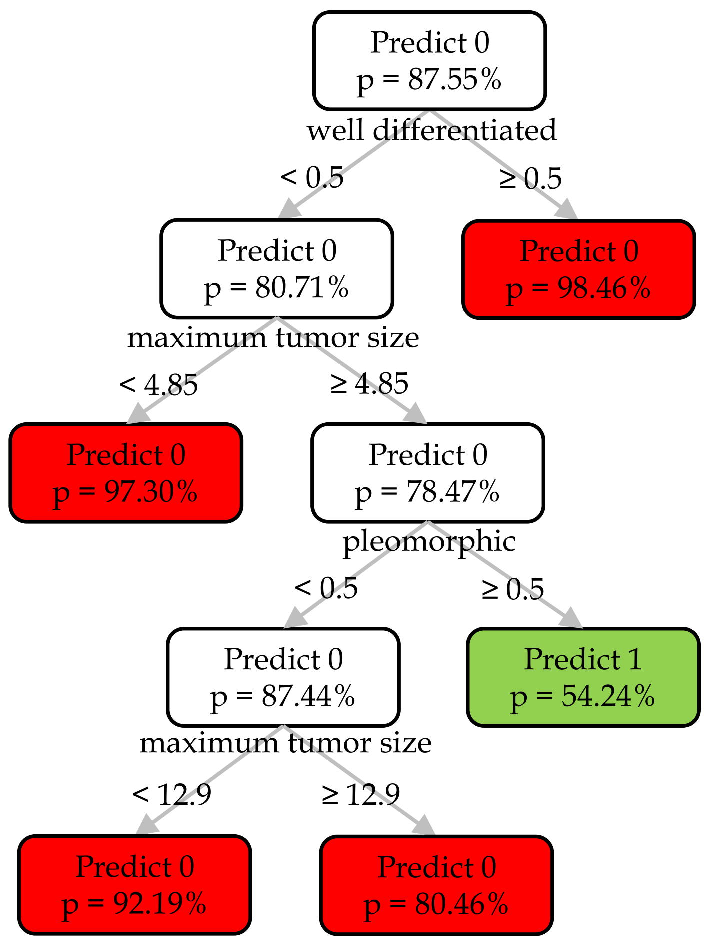

Figure 6.2: OCT of sarcoma dataset trained with maximum depth 5 and minbucket 35 patients. The OCT model has 4 splits and achieves out-of-sample Gini impurity score (a lower score is better) of 0.1686.

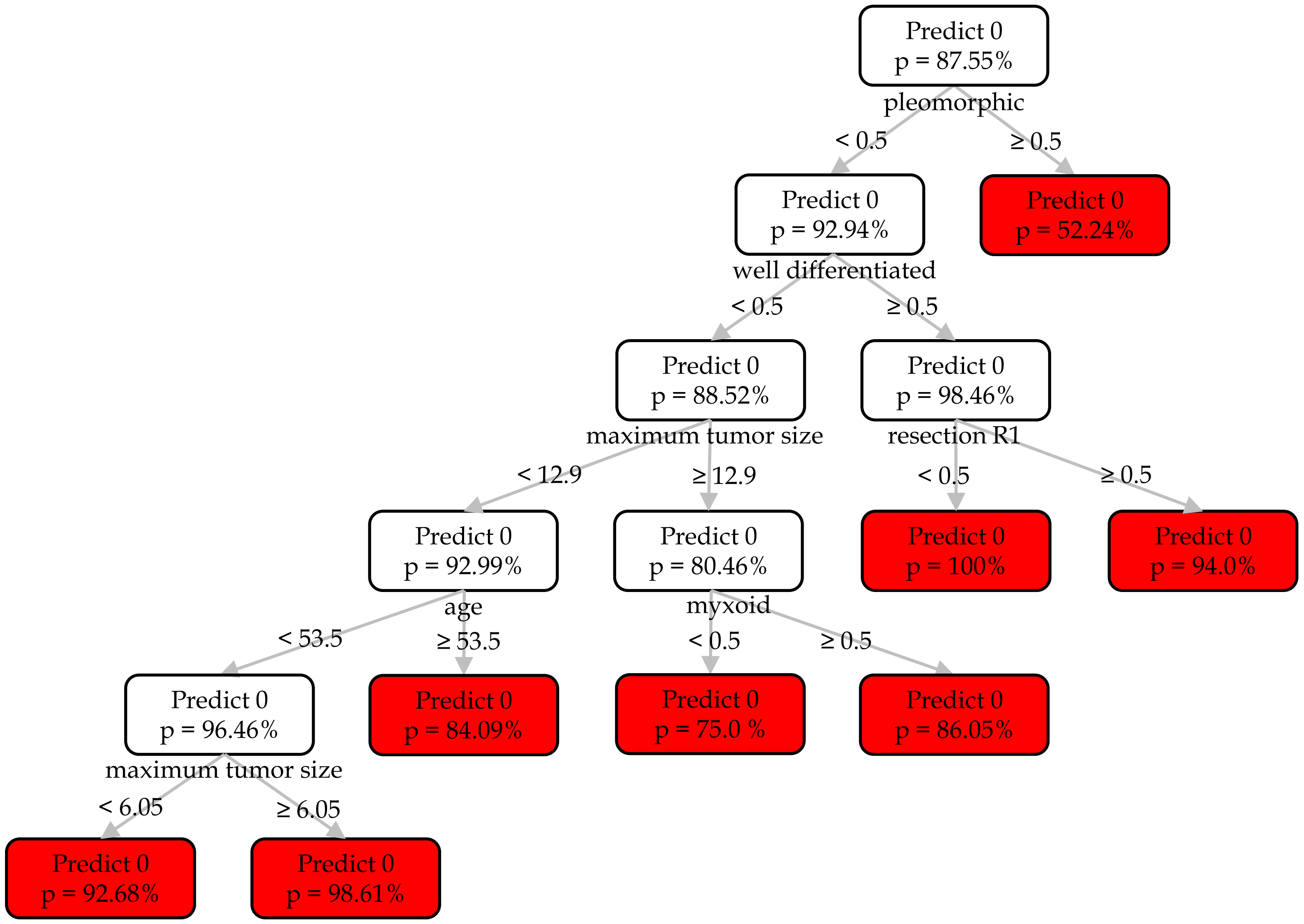

Figure 6.3: CART of sarcoma dataset trained with maximum depth 5 and minbucket 35 patients. The CART model has 7 splits and achieves out-of-sample Gini impurity score (a lower score is better) of 0.1698.

i.e., OCT-SA, OCT, and CART, select maximum tumor size, and 2 histologic subtypes (well differentiated, and pleomorphic). In addition to these 3 features, OCT-SA selects a histologic subtype myxoid, and surgical margin R1. Then CART selects age as an additional feature. To sum up, OCT-SA performs best in terms of out-of-sample Gini impurity score for classification analysis, as it can determine additional meaningful splits on extra features better than OCT, while CART performs worse on both the classification performance, and the complexity of its tree structure.

6.2 Prescriptive Analysis for Gastrointestinal Stromal Tumor (GIST)

6.2.1 Experimental Setup

In order to demonstrate an application of OPT-SA in the medical domain, we construct a model to select subgroup of patients with Gastrointestinal Stromal Tumor (GIST) [94] who will benefit from getting chemotherapy after tumor removal surgery [76], where the chemotherapy will reduce the chance that the tumor recurs within 5 years after surgery [56, 96]. We can translate this problem to a prescriptive analysis, where we want to prescribe a binary treatment (undergoing chemotherapy, or no chemotherapy), in order to maximize the probability of recurrence free survival at 5 years after surgery.

We obtain the GIST dataset from MSKCC, the same institution from which we obtain the sarcoma dataset for classification analysis case study. Originally, the dataset contained 536 patients who underwent surgery at MSKCC from 1982 to 2017. We use a matching framework based on prognostic strata [52, 77, 97] to match the 82 treated patients to the untreated patients, so there are 164 patients in total in our prescriptive analysis. In terms of counterfactual models to predict counterfactual outcomes, we train 2 separate survival Random Forest models: one model for the treated patient group, and another model for the untreated patient group. We train these counterfactual models using the direct method, as shown in Equation (1.4), to estimate rewards and measure the performance of these reward estimators using Harrell's C-statistic [45–47, 93].

We include 3 significant tumor characteristics, as identified based on clinical intuition of our medical collaborator, in our analysis: maximum tumor size, primary mitotic count, and tumor site (gastric site vs non-gastric site) [27, 66, 67]. We encode all of these feature values in numeric values, so that they are feasible for the OPT-SA algorithm when determining parallel split. We define the prescription as a binary treatment over chemotherapy (prescribing chemotherapy or no chemotherapy). The outcomes of interest are whether or not tumors recurred within 5 years after surgery. In terms of rewards, they are the probabilities of recurrence free survival at 5 years after surgery, and we want to maximize the mean rewards score. As not all treatment options were actually prescribed to each patient in our observed dataset, we use a Random Forest counterfactual model to estimate the counterfactual outcomes.

Since the GIST dataset is relatively small, we use all data samples, i.e., 164 patients to train the model, so there is no unseen test dataset. In terms of model training, we train the models with maximum depths ranging from 2 to 5. We cap the maximum depth at 5 in order to ensure models are not too complicated, which will increase the chance of model adoption by physicians in the real world. We vary minbuckets as 3%, 5%, and 10% of the total number of training samples. As we did in the evaluation on the 10 real-world benchmarking datasets for prescriptive analysis, we also use Markov chain length $l = 2$ and search radius $r = 3$ when training OPT-SA. The complexity penalty γ is the only hyperparameter that we tune.

Finally, we select the OPT-SA model that uses all 3 tumor characteristics in the model and achieves the highest in-sample mean rewards score. Following the recommendation from our medical collaborator, we target the model that selects all 3 features of tumor characteristics, as such model is more likely to make clinical sense and provide interesting feature interactions. To make performance comparison with OPT-SA, we also train OPT and CART as our baseline models.

6.2.2 OPT-SA vs OPT vs CART

Figures 6.4, 6.5, and 6.6 demonstrate the OPT-SA, OPT, and CART of the GIST dataset. We trained these 3 models with maximum depth ranging from 2 to 5, and minbucket 3%, 5%, or 10% over the entire dataset. Based on the clinical intuition of our medical collaborators, we only select models that split all 3 features, i.e., maximum tumor size, primary mitotic count, and tumor site (gastric site vs non-gastric site). Accordingly, all the 3 features are present in the OPT-SA, OPT, and CART. Since the GIST dataset is rather small, we use the entire dataset for training and have no hold-out test set. That is why we can measure only in-sample mean rewards score. In terms of performance ranking based on in-sample mean rewards score, OPT-SA ranks number one, followed by CART, and finally OPT, with their mean rewards scores equal 0.8238, 0.8223, and 0.8178 (a higher score is better) respectively. By examining the tree complexities, OPT is the simplest model with no split, followed by OPT-SA with 6 splits, and CART is the most complex model with 8 splits. This is one example scenario where OPT can fail to overcome CART. This kind of failure is possibly due to local search in OPT getting stuck in local minima. Given less complex tree structure, OPT-SA can still outperform CART on the in-sample mean rewards score. Such performance improvement proves that simulated annealing can effectively find more meaningful split than CART.

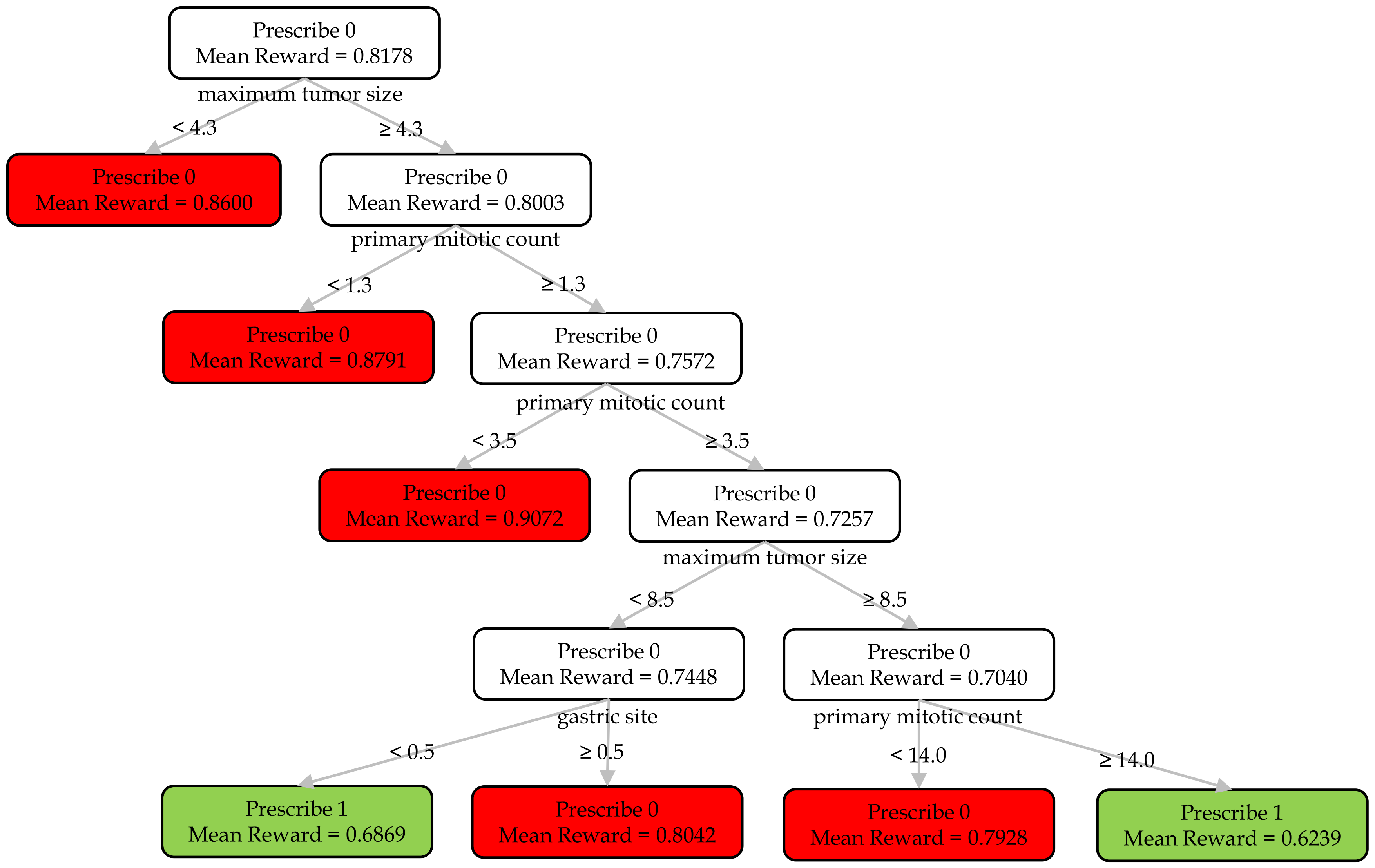

Figure 6.4: OPT-SA of GIST dataset trained with maximum depth ranging from 2 to 5, and minbucket 3%, 5%, or 10% over entire dataset. The OPT-SA model has 6 splits and achieves in-sample mean rewards score (a higher score is better) of 0.8238.

Figure 6.5: OPT of GIST dataset trained with maximum depth ranging from 2 to 5, and minbucket 3%, 5%, or 10% over entire dataset. The OPT model has no split and achieves in-sample mean rewards score (a higher score is better) of 0.8178.

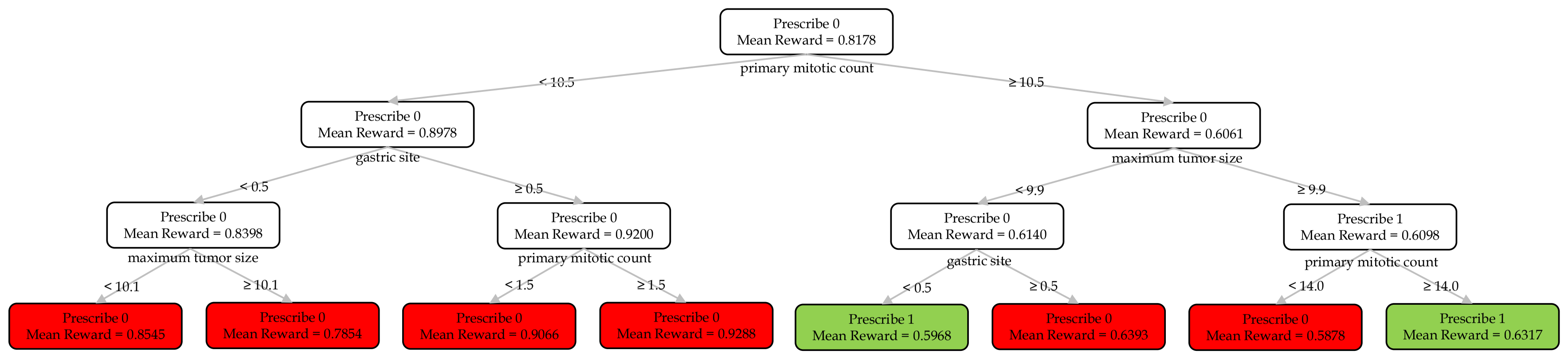

Figure 6.6: CART of GIST dataset trained with maximum depth ranging from 2 to 5, and minbucket 3%, 5%, or 10% over entire dataset. The CART model has 7 splits and achieves in-sample mean rewards score (a higher score is better) of 0.8223.

6.3 Survival Analysis for Sarcoma

6.3.1 Experimental Setup

In order to showcase the OST-SA algorithm in the real-world medical dataset, we conduct survival analysis on the sarcoma dataset, the same dataset we use to conduct classification analysis in Section 6.1. Instead of focusing on a specific time point, i.e., 5 years after surgery, like that of classification analysis case study, we perform survival analysis on the full span of survival time of patients in the dataset [28]. Our aim is to estimate how long each subgroup of patients survives after surgery. Accordingly, we do not exclude any patients, and use all 712 patients in our analysis. They are all the patients with sarcoma who had surgery at MSKCC between July 1982 and October 2017. In contrast, the classification analysis case study only includes 596 patients, as we need to remove any censored data for the 5-year time point after surgery.

We also use the same 17 features as that of classification analysis in this survival analysis: 2 continuous features (age, and maximum tumor size), and 15 binary features (gender, 3 tumor sites[4], tumor depth, bone invasion, nerve invasion, 5 histologic subtypes[5], and 3 surgical margins[6]). Similarly, we convert all non-numeric features to numbers for feasibility when finding parallel split in the OST-SA algorithm. We use the same training and unseen test dasaset as that of classification analysis case study. We split the dataset into training, validation, and testing dataset with the ratio of train:validation:test equals 70:15:15. We train OST-SA, OST, and sksurv models with the same hyperparameter as that of our classification analysis, i.e., 35 patients as minbucket, and maximum depths ranging from 2 to 5. These hyperparameters are based on clinical intuition of our medical collaborators. When training OST-SA, we use Markov chain length $l = 2$ and search radius $r = 3$, as they are the

[4]The 3 tumor sites are 3 binary features: lower extremity (y/n), trunk (y/n), and upper extremity (y/n).

[5]The 5 histologic subtypes are 5 binary features: dedifferentiated (y/n), myxoid (y/n), pleomorphic (y/n), round cell (y/n), and well differentiated (y/n)

[6]The 3 surgical margins are 3 binary features: resection R1 (y/n), R2 (y/n), and R3 (y/n).

hyperparameters we use when training OST-SA on the 10-real world benchmarking datasets. The only hyperparameter that we tune when training OST-SA is complexity penalty γ.

6.3.2 OST-SA vs OST vs sksurv

As we investigate models' performance trained with maximum depth ranging from 2 to 5, and minbucket 35 patients, all models achieve the best local full likelihood score at maximum depth 4 as shown in Figures 6.7, 6.8, and 6.9. In terms of out-of-sample performance, sksurv survival tree achieves the best out-of-sample local full likelihood score (a lower score is better) of 0.6739, followed by OST-SA with the score of 0.7123, and finally, OST obtains the worst score of 0.7269. The trend of out-of-sample performance in this survival analysis case study follows the same trend as that of the 10 real-world benchmarking datasets for survival analysis. Regarding tree complexity, OST is the simplest model with 4 splits, followed by sksurv with 7 splits, and OST-SA is the most complex model with 11 splits. The results indicate that simulated annealing can benefit decision tree growing process for survival analysis in the real-world medical datasets. That is why OST-SA outperforms OST on out-of-sample performance, as it can identify additional meaningful splits to improve out-of-sample local full likelihood score. Interestingly, sksurv is a very strong algorithm, and can effectively identify 7 meaningful splits that outperforms 11 splits in OST-SA. By considering feature selection, all the 4 features selected by OST also get selected by OST-SA and sksurv. Those 4 features are 2 histologic subtypes (well differentiated, and pleomorphic), maximum tumor size, and surgical resection R1. In addition to these 4 features, sksurv selects one additional feature: a round cell histologic subtype. For the case of OST-SA, it selects additional 2 features (besides the 4 features that all models select): a myxoid histologic subtype and age. In summary, OST-SA outperforms OST by achieving better out-of-sample local full likelihood score as it can effectively identify meaningful splits on additional features, while it is interesting to note that sksurv can still perform better than OST-SA in terms of the out-of-sample performance with the simpler tree structure.

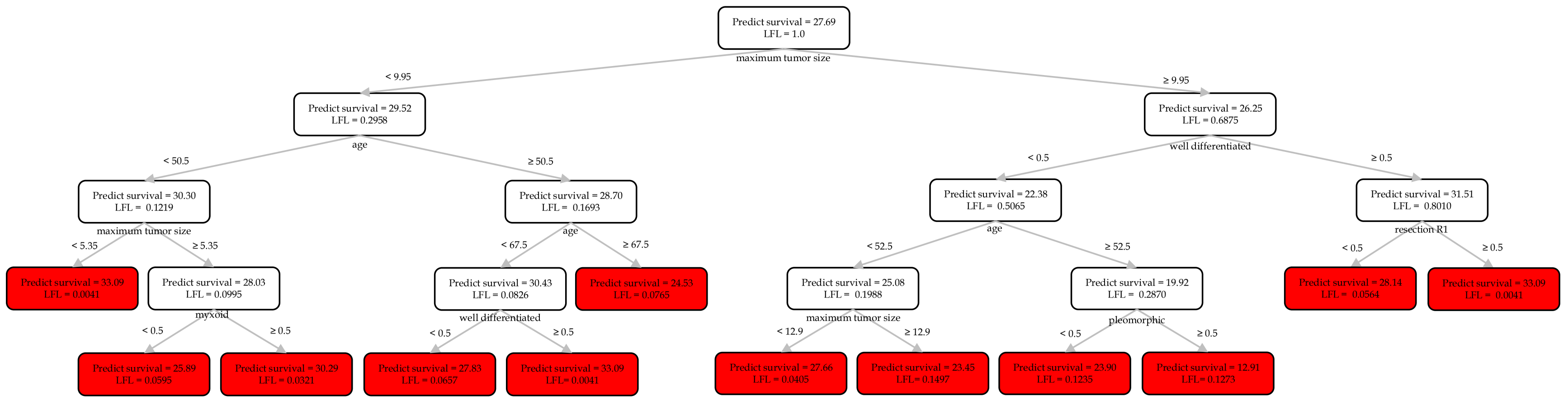

Figure 6.7: OST-SA of sarcoma dataset trained with maximum depth 4 and minbucket 35 patients. The OST-SA model has 11 splits and achieves out-of-sample local full likelihood (LFL) score (a lower score is better) of 0.7123.

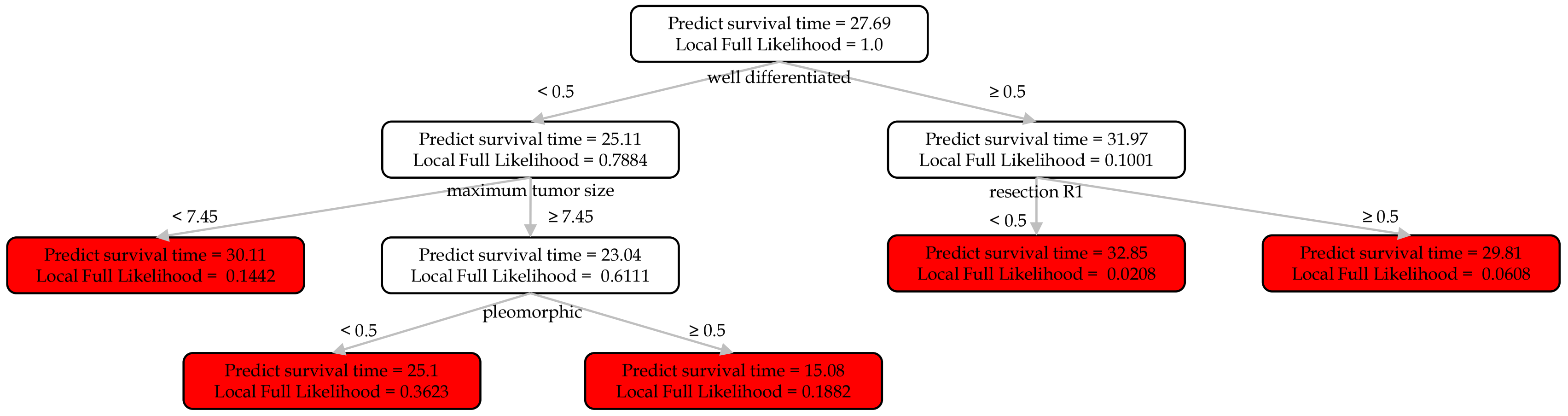
Predict survival time = 27.69
Local Full Likelihood = 1.0
well differentiated
< 0.5
≥ 0.5
Predict survival time = 25.11
Local Full Likelihood = 0.7884
maximum tumor size
< 7.45
≥ 7.45
Predict survival time = 31.97
Local Full Likelihood = 0.1001
resection R1
< 0.5
≥ 0.5
Predict survival time = 30.11
Local Full Likelihood = 0.1442
Predict survival time = 23.04
Local Full Likelihood = 0.6111
pleomorphic
< 0.5
≥ 0.5
Predict survival time = 32.85
Local Full Likelihood = 0.0208
Predict survival time = 29.81
Local Full Likelihood = 0.0608
Predict survival time = 25.1
Local Full Likelihood = 0.3623
Predict survival time = 15.08
Local Full Likelihood = 0.1882

Figure 6.8: OST of sarcoma dataset trained with maximum depth 4 and minbucket 35 patients. The OST model has 4 splits and achieves out-of-sample local full likelihood score (a lower score is better) of 0.7269.

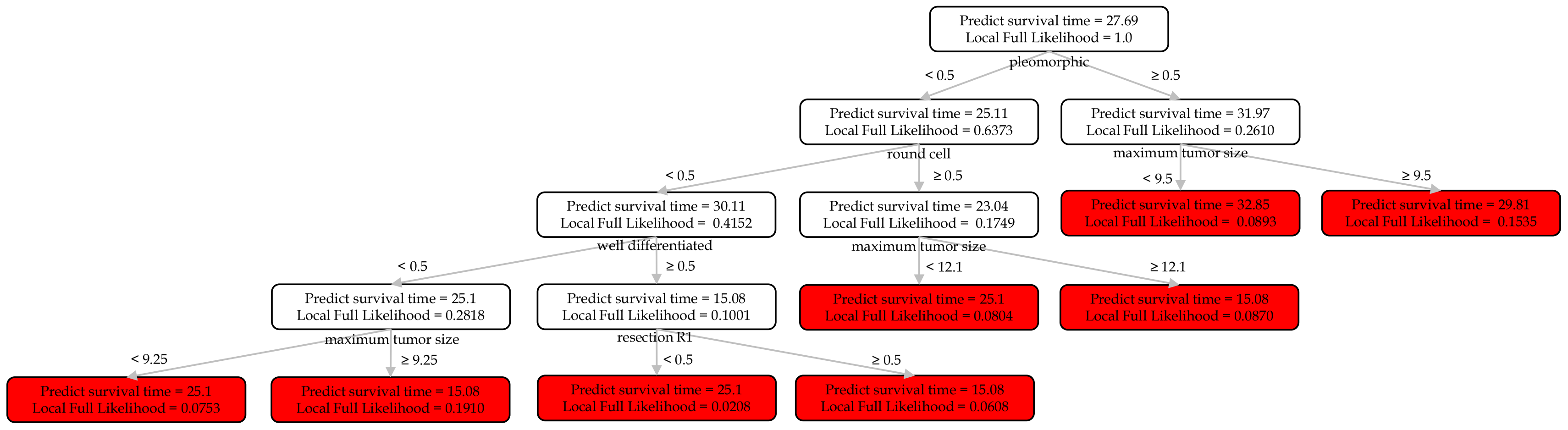
Predict survival time = 27.69
Local Full Likelihood = 1.0
pleomorphic
< 0.5
≥ 0.5
Predict survival time = 25.11
Local Full Likelihood = 0.6373
round cell
Predict survival time = 31.97
Local Full Likelihood = 0.2610
maximum tumor size
< 0.5
≥ 0.5
< 9.5
≥ 9.5
Predict survival time = 30.11
Local Full Likelihood = 0.4152
well differentiated
Predict survival time = 23.04
Local Full Likelihood = 0.1749
maximum tumor size
Predict survival time = 32.85
Local Full Likelihood = 0.0893
Predict survival time = 29.81
Local Full Likelihood = 0.1535
< 0.5
≥ 0.5
< 12.1
≥ 12.1
Predict survival time = 25.1
Local Full Likelihood = 0.2818
maximum tumor size
Predict survival time = 15.08
Local Full Likelihood = 0.1001
resection R1
Predict survival time = 25.1
Local Full Likelihood = 0.0804
Predict survival time = 15.08
Local Full Likelihood = 0.0870
< 9.25
≥ 9.25
< 0.5
≥ 0.5
Predict survival time = 25.1
Local Full Likelihood = 0.0753
Predict survival time = 15.08
Local Full Likelihood = 0.1910
Predict survival time = 25.1
Local Full Likelihood = 0.0208
Predict survival time = 15.08
Local Full Likelihood = 0.0608

Figure 6.9: sksurv survival tree of sarcoma dataset trained with maximum depth 4 and minbucket 35 patients. The sksurv model has 7 splits and achieves out-of-sample local full likelihood score (a lower score is better) of 0.6739.

Chapter 7

Conclusion

The optimal decision trees with simulated annealing successfully improve out-of-sample performance on optical decision trees with a greedy approach on classification, prescriptive, and survival analysis. First, OCT-SA outperforms OCT on out-of-sample Gini impurity scores over 61 real-world benchmarking datasets at all maximum depths. The results are statistically significant at maximum depths 3, 4, and 7. Second, OPT-SA outperforms OPT on 3 out of 10 real-world benchmarking datasets. Although the improvement was not found for all datasets, the results suggest there are certain types of datasets for which simulated annealing can better benefit the decision tree construction process. Third, OST-SA outperforms OST on 10 real-world benchmarking datasets at all maximum depths. The results are statistically significant at any maximum depths higher than 2.

Beyond the improvement on benchmarking datasets, the optimal decision trees with simulated annealing also exhibit improvement on case studies in the medical domain. First, OCT-SA can provide more accurate mortality prediction of patients with sarcoma to predict whether or not they will die within 5 years after surgery. This improvement in classification performance helps healthcare professionals to better focus on critical cases. Second, OPT-SA prescribes a better treatment option than OPT for patients with GIST. This better prescription leads to a decrease in the recurrence rate of GIST in patients. Third, OST-

SA predicts more accurate survival time than OST for patients with sarcoma. This better accuracy helps the healthcare institutions to derive more appropriate care for patients.

Even though the optimal decision trees with simulated annealing achieve satisfactory performance in classification, prescriptive, and survival analysis, further improvement can still be made. In addition to classification, prescriptive, and survival analysis, regression analysis is another problem domain that is widely-used. Accordingly, we can also apply this type of optimal decision tree with simulated annealing framework to regression analysis. In investigating results on benchmarking datasets, improvement was not found on all datasets. This failure could possibly be due to our model training being susceptible to bias. One potential mitigation is to incorporate robust optimization in model training [9, 15, 64, 95].

Besides OCT, OPT, and OST, Interpretable AI [51] also provides Optimal Regression Tree (ORT) for regression analysis. As ORT also utilizes local search like that of OCT, OPT, and OST, it may fail to reach global optimality. By replacing local search with simulated annealing in ORT and obtaining ORT-SA, we can improve the predictive performance of ORT as some worse transformations in ORT-SA can result in a better final model. We can adopt a similar framework as that of OCT-SA to implement ORT-SA with the change from class prediction in OCT-SA to continuous value prediction in ORT-SA. In addition, we also need to scale the objective value of ORT-SA to be in the range from 0 to 1, i.e., the same range as the objective value of OCT-SA. This scaling process requires an appropriate scaling factor for each dataset.

One of the underlying reasons why simulated annealing still fails to outperform local search in some cases could be due to overfitting [48] or failure to obtain appropriate hyperparameters in model training [35]. By incorporating robustness in model training, the trained models are likely to be less susceptible to noisy data and can better capture meaningful information [36, 53]. This approach can result in more generalized models, which leads to better out-of-sample performance. Regarding hyperparameter tuning, the current approach selects the best combination of hyperparameters solely on scoring over validation datasets.

This procedure can yield inappropriate hyperparameter selection if a validation dataset is not a good representative of its corresponding training dataset [89]. Similarly, utilizing robustness to take into account the characteristics of validation datasets can possibly reduce bias in hyperparameter selection [98].

References

[1] Odd Aalen. Nonparametric inference for a family of counting processes. *The Annals of Statistics*, pages 701–726, 1978.

[2] Odd Aalen, Ornulf Borgan, and Hakon Gjessing. *Survival and event history analysis: a process point of view.* Springer Science & Business Media, 2008.

[3] Paul D Allison. *Event history and survival analysis: Regression for longitudinal event data*, volume 46. SAGE publications, 2014.

[4] Maxime Amram, Jack Dunn, and Ying Daisy Zhuo. Optimal policy trees. *Machine Learning*, 111(7):2741–2768, 2022.

[5] Jordan Ash and Ryan P Adams. On warm-starting neural network training. *Advances in neural information processing systems*, 33:3884–3894, 2020.

[6] Arthur Asuncion and David Newman. Uci machine learning repository, 2007.

[7] Peter C Austin. An introduction to propensity score methods for reducing the effects of confounding in observational studies. *Multivariate behavioral research*, 46(3):399–424, 2011.

[8] Heejung Bang and James M Robins. Doubly robust estimation in missing data and causal inference models. *Biometrics*, 61(4):962–973, 2005.

[9] D. Bertsimas and D. den Hertog. *Robust and Adaptive Optimization*. Dynamic Ideas LLC, 2022. ISBN 9781733788526. URL https://books.google.com/books?id=V_RPzwEACAAJ.

[10] Dimitris Bertsimas and Jack Dunn. Optimal classification trees. *Machine Learning*, 106(7):1039–1082, 2017.

[11] Dimitris Bertsimas and Jack Dunn. *Machine learning under a modern optimization lens*. Dynamic Ideas LLC, 2019.

[12] Dimitris Bertsimas and John Tsitsiklis. Simulated annealing. *Statistical science*, 8(1): 10–15, 1993.

[13] Dimitris Bertsimas and John N Tsitsiklis. *Introduction to linear optimization*, volume 6. Athena Scientific Belmont, MA, 1997.

[14] Dimitris Bertsimas, Jack Dunn, and Nishanth Mundru. Optimal prescriptive trees. *INFORMS Journal on Optimization*, 1(2):164–183, 2019.

[15] Dimitris Bertsimas, Jack Dunn, Colin Pawlowski, and Ying Daisy Zhuo. Robust classification. *INFORMS Journal on Optimization*, 1(1):2–34, 2019.

[16] Dimitris Bertsimas, Jack Dunn, Emma Gibson, and Agni Orfanoudaki. Optimal survival trees. *Machine Learning*, 111(8):2951–3023, 2022.

[17] L Breiman, JH Friedman, R Olshen, and CJ Stone. Classification and regression trees. 1984.

[18] Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

[19] Leo Breiman. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical science*, 16(3):199–231, 2001.

[20] RS Bucy and RS Diesposti. Decision tree design by simulated annealing. *ESAIM: Mathematical Modelling and Numerical Analysis*, 27(5):515–534, 1993.

[21] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[22] Taane G Clark, Michael J Bradburn, Sharon B Love, and Douglas G Altman. Survival analysis part i: basic concepts and first analyses. *British journal of cancer*, 89(2): 232–238, 2003.

[23] Jean-Michel Coindre. Grading of soft tissue sarcomas: review and update. *Archives of pathology & laboratory medicine*, 130(10):1448–1453, 2006.

[24] Janice N Cormier and Raphael E Pollock. Soft tissue sarcomas. *CA: a cancer journal for clinicians*, 54(2):94–109, 2004.

[25] Kimberly Moore Dalal, Michael W Kattan, Cristina R Antonescu, Murray F Brennan, and Samuel Singer. Subtype specific prognostic nomogram for patients with primary liposarcoma of the retroperitoneum, extremity, or trunk. *Annals of surgery*, 244(3): 381, 2006.

[26] Daniela-Emanuela Danacica and Ana-Gabriela Babucea. Using survival analysis in economics. *survival*, 11:15, 2010.

[27] Ronald P DeMatteo, Jason S Gold, Lisa Saran, Mithat Gönen, Kui Hin Liau, Robert G Maki, Samuel Singer, Peter Besmer, Murray F Brennan, and Cristina R Antonescu. Tumor mitotic rate, size, and location independently predict recurrence after resection of primary gastrointestinal stromal tumor (gist). *Cancer: Interdisciplinary International Journal of the American Cancer Society*, 112(3):608–615, 2008.

[28] Ian C Dickinson, Duncan J Whitwell, Diane Battistuta, Bridie Thompson, Nichola Strobel, Amit Duggal, and Peter Steadman. Surgical margin and its influence on survival in soft tissue sarcoma. *ANZ journal of surgery*, 76(3):104–109, 2006.

[29] Erik Drysdale. Survset: An open-source time-to-event dataset repository. *arXiv preprint arXiv:2203.03094*, 2022.

[30] Jack William Dunn. *Optimal trees for prediction and prescription.* PhD thesis, Massachusetts Institute of Technology, 2018.

[31] Makoto Endo and Patrick P Lin. Surgical margins in the management of extremity soft tissue sarcoma. *Chin Clin Oncol*, 7(4):37, 2018.

[32] Joost Engelfriet. Bottom-up and top-down tree transformations—a comparison. *Mathematical systems theory*, 9(2):198–231, 1975.

[33] Peter C Ferguson, Anthony M Griffin, Brian O'Sullivan, Charles N Catton, Aileen M Davis, Ally Murji, Robert S Bell, and Jay S Wunder. Bone invasion in extremity soft-tissue sarcoma: impact on disease outcomes. *Cancer: Interdisciplinary International Journal of the American Cancer Society*, 106(12):2692–2700, 2006.

[34] Andrea Ferrari, Iyad Sultan, Tseng Tien Huang, Carlos Rodriguez-Galindo, Ahmad Shehadeh, Cristina Meazza, Kirsten K Ness, Michela Casanova, and Sheri L Spunt. Soft tissue sarcoma across the age spectrum: a population-based study from the surveillance epidemiology and end results database. *Pediatric blood & cancer*, 57(6):943–949, 2011.

[35] Matthias Feurer and Frank Hutter. Hyperparameter optimization. *Automated machine learning: Methods, systems, challenges*, pages 3–33, 2019.

[36] Benoît Frénay and Michel Verleysen. Classification in the presence of label noise: a survey. *IEEE transactions on neural networks and learning systems*, 25(5):845–869, 2013.

[37] German Gemar, Laura Moniche, and Antonio J Morales. Survival analysis of the spanish hotel industry. *Tourism Management*, 54:428–438, 2016.

[38] Adrian Gepp and Kuldeep Kumar. The role of survival analysis in financial distress prediction. *International research journal of finance and economics*, 16(16):13–34, 2008.

[39] Craig H Gerrand, Robert S Bell, Jay S Wunder, Rita A Kandel, Brian O'Sullivan, Charles N Catton, Anthony M Griffin, and Aileen M Davis. The influence of anatomic location on outcome in patients with soft tissue sarcoma of the extremity. *Cancer*, 97 (2):485–492, 2003.

[40] Irène Gijbels. Censored data. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(2):178–188, 2010.

[41] Manish Kumar Goel, Pardeep Khanna, and Jugal Kishore. Understanding survival analysis: Kaplan-meier estimate. *International journal of Ayurveda research*, 1(4):274, 2010.

[42] Robert Grimer, Ian Judson, David Peake, and Beatrice Seddon. Guidelines for the management of soft tissue sarcomas. *Sarcoma*, 2010, 2010.

[43] Robert J Grimer. Size matters for sarcomas! *The Annals of The Royal College of Surgeons of England*, 88(6):519–524, 2006.

[44] Balazs Györffy, Andras Lanczky, Aron C Eklund, Carsten Denkert, Jan Budczies, Qiyuan Li, and Zoltan Szallasi. An online survival analysis tool to rapidly assess the effect of 22,277 genes on breast cancer prognosis using microarray data of 1,809 patients. *Breast cancer research and treatment*, 123:725–731, 2010.

[45] Frank E Harrell, Robert M Califf, David B Pryor, Kerry L Lee, and Robert A Rosati. Evaluating the yield of medical tests. *Jama*, 247(18):2543–2546, 1982.

[46] Frank E Harrell Jr, Kerry L Lee, Robert M Califf, David B Pryor, and Robert A Rosati. Regression modelling strategies for improved prognostic prediction. *Statistics in medicine*, 3(2):143–152, 1984.

[47] Frank E Harrell Jr, Kerry L Lee, and Daniel B Mark. Multivariable prognostic models: issues in developing models, evaluating assumptions and adequacy, and measuring and reducing errors. *Statistics in medicine*, 15(4):361–387, 1996.

[48] Douglas M Hawkins. The problem of overfitting. *Journal of chemical information and computer sciences*, 44(1):1–12, 2004.

[49] Mohammad Hossin and Md Nasir Sulaiman. A review on evaluation metrics for data classification evaluations. *International journal of data mining & knowledge management process*, 5(2):1, 2015.

[50] Torsten Hothorn, Kurt Hornik, Carolin Strobl, and Achim Zeileis. Party: A laboratory for recursive partytioning, 2010.

[51] LLC Interpretable AI. Interpretable ai documentation, 2023. URL https://www.interpretable.ai.

[52] Heikki Joensuu. Risk stratification of patients diagnosed with gastrointestinal stromal tumor. *Human pathology*, 39(10):1411–1419, 2008.

[53] George H John. Robust decision trees: Removing outliers from databases. In *KDD*, volume 95, pages 174–179, 1995.

[54] David S Johnson, Christos H Papadimitriou, and Mihalis Yannakakis. How easy is local search? *Journal of computer and system sciences*, 37(1):79–100, 1988.

[55] Edward L Kaplan and Paul Meier. Nonparametric estimation from incomplete observations. *Journal of the American statistical association*, 53(282):457–481, 1958.

[56] Rakesh Kapoor, Divya Khosla, Pankaj Kumar, Narendra Kumar, and Anjan Bera. Five-year follow up of patients with gastrointestinal stromal tumor: Recurrence-free survival by risk group. *Asia-Pacific Journal of Clinical Oncology*, 9(1):40–46, 2013.

[57] Md Rezaul Karim, M Ataharul Islam, et al. *Reliability and survival analysis*. Springer, 2019.

[58] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[59] John P Klein, Melvin L Moeschberger, et al. *Survival analysis: techniques for censored and truncated data*, volume 1230. Springer, 2003.

[60] Mariam P Korah, Andrea T Deyrup, David K Monson, Shervin V Oskouei, Sharon W Weiss, Jerome Landry, and Karen D Godette. Anatomic tumor location influences the success of contemporary limb-sparing surgery and radiation among adults with soft tissue sarcomas of the extremities. *International Journal of Radiation Oncology* Biology* Physics*, 82(2):933–939, 2012.

[61] Hyafil Laurent and Ronald L Rivest. Constructing optimal binary decision trees is np-complete. *Information processing letters*, 5(1):15–17, 1976.

[62] Michael LeBlanc and John Crowley. Relative risk trees for censored survival data. *Biometrics*, pages 411–425, 1992.

[63] Elisa T Lee and Oscar T Go. Survival analysis in public health research. *Annual review of public health*, 18(1):105–134, 1997.

[64] Wei Liu, Sanjay Chawla, David A Cieslak, and Nitesh V Chawla. A robust decision tree algorithm for imbalanced data sets. In *Proceedings of the 2010 SIAM International Conference on Data Mining*, pages 766–777. SIAM, 2010.

[65] James F Lutsko and Bart Kuijpers. Simulated annealing in the construction of near-optimal decision trees. In *Selecting Models from Data: Artificial Intelligence and Statistics IV*, pages 453–462. Springer, 1994.

[66] Markku Miettinen and Jerzy Lasota. Gastrointestinal stromal tumors: pathology and prognosis at different sites. In *Seminars in diagnostic pathology*, volume 23, pages 70–83. Elsevier, 2006.

[67] Markku Miettinen, Leslie H Sobin, Jerzy Lasota, et al. Evaluation of malignancy and prognosis of gastrointestinal stromal tumors: a review. *Human pathology*, 33(5): 478–483, 2002.

[68] Stephen L Morgan and Christopher Winship. *Counterfactuals and causal inference.* Cambridge University Press, 2015.

[69] Sreerama K Murthy and Steven Salzberg. Decision tree induction: How effective is the greedy heuristic? In *KDD*, pages 222–227, 1995.

[70] T Nakamura, RJ Grimer, SR Carter, RM Tillman, A Abudu, L Jeys, and A Sudo. Outcome of soft-tissue sarcoma patients who were alive and event-free more than five years after initial treatment. *The Bone & Joint Journal*, 95(8):1139–1143, 2013.

[71] Wayne Nelson. Hazard plotting for incomplete failure data. *Journal of Quality Technology*, 1(1):27–52, 1969.

[72] Wayne Nelson. Theory and applications of hazard plotting for censored failure data. *Technometrics*, 14(4):945–966, 1972.

[73] Stefano Nembrini, Inke R König, and Marvin N Wright. The revival of the gini importance? *Bioinformatics*, 34(21):3711–3718, 2018.

[74] Yaghout Nourani and Bjarne Andresen. A comparison of simulated annealing cooling strategies. *Journal of Physics A: Mathematical and General*, 31(41):8373, 1998.

[75] David M Panicek, Stephen D Go, John H Healey, DH Leung, MF Brennan, and JJ Lewis. Soft-tissue sarcoma involving bone or neurovascular structures: Mr imaging prognostic factors. *Radiology*, 205(3):871–875, 1997.

[76] Trisha M Parab, Michael J DeRogatis, Alexander M Boaz, Salvatore A Grasso, Paul S Issack, David A Duarte, Olivier Urayeneza, Saloomeh Vahdat, Jian-Hua Qiao, and Gudata S Hinika. Gastrointestinal stromal tumors: a comprehensive review. *Journal of gastrointestinal oncology*, 10(1):144, 2019.

[77] Cheol Keun Park, Eui Jin Lee, Minji Kim, Ho-Yeong Lim, Dong Il Choi, Jae Hyung Noh, Tae Sung Sohn, Sung Kim, Mi Jung Kim, Hun Kyung Lee, et al. Prognostic stratification of high-risk gastrointestinal stromal tumors in the era of targeted therapy. *Annals of surgery*, 247(6):1011–1018, 2008.

[78] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[79] Trond Petersen. The statistical analysis of event histories. *Sociological Methods & Research*, 19(3):270–323, 1991.

[80] Sebastian Pölsterl. scikit-survival: A library for time-to-event analysis built on top of scikit-learn. *The Journal of Machine Learning Research*, 21(1):8747–8752, 2020.

[81] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1:81–106, 1986.

[82] J Ross Quinlan. *C4. 5: programs for machine learning.* Elsevier, 2014.

[83] R R Core Team et al. R: A language and environment for statistical computing. 2013.

[84] Albert Reuther, Jeremy Kepner, Chansup Byun, Siddharth Samsi, William Arcand, David Bestor, Bill Bergeron, Vijay Gadepally, Michael Houle, Matthew Hubbell,

Michael Jones, Anna Klein, Lauren Milechin, Julia Mullen, Andrew Prout, Antonio Rosa, Charles Yee, and Peter Michaleas. Interactive supercomputing on 40,000 cores for machine learning and data analysis. In *2018 IEEE High Performance extreme Computing Conference (HPEC)*, pages 1–6. IEEE, 2018.

[85] Charles J Romeo. Conducting inference in semiparametric duration models under inequality restrictions on the shape of the hazard implied by job search theory. *Journal of Applied Econometrics*, 14(6):587–605, 1999.

[86] Paul R Rosenbaum and Donald B Rubin. The central role of the propensity score in observational studies for causal effects. *Biometrika*, 70(1):41–55, 1983.

[87] Anders Rydholm and Pelle Gustafson. Should tumor depth be included in prognostication of soft tissue sarcoma? *BMC cancer*, 3(1):1–5, 2003.

[88] Guillermo Salinas-Escudero, María Fernanda Carrillo-Vega, Víctor Granados-García, Silvia Martínez-Valverde, Filiberto Toledano-Toledano, and Juan Garduño-Espinosa. A survival analysis of covid-19 in the mexican population. *BMC public health*, 20(1): 1–8, 2020.

[89] Miriam Seoane Santos, Jastin Pompeu Soares, Pedro Henrigues Abreu, Helder Araujo, and Joao Santos. Cross-validation for imbalanced datasets: Avoiding overoptimistic and overfitting approaches [research frontier]. *ieee ComputatioNal iNtelligeNCe magaziNe*, 13(4):59–76, 2018.

[90] Keith M Skubitz and David R D'Adamo. Sarcoma. In *Mayo Clinic Proceedings*, volume 82, pages 1409–1432. Elsevier, 2007.

[91] Suryakanthi Tangirala. Evaluating the impact of gini index and information gain on classification using decision tree classifier algorithm. *International Journal of Advanced Computer Science and Applications*, 11(2):612–619, 2020.

[92] Terry Therneau, Beth Atkinson, Brian Ripley, and Maintainer Brian Ripley. Package 'rpart'. *Available online: cran. ma. ic. ac. uk/web/packages/rpart/rpart. pdf (accessed on 20 April 2016)*, 2015.

[93] Hajime Uno, Tianxi Cai, Michael J Pencina, Ralph B D'Agostino, and Lee-Jen Wei. On the c-statistics for evaluating overall adequacy of risk prediction procedures with censored survival data. *Statistics in medicine*, 30(10):1105–1117, 2011.

[94] Margaret von Mehren, Brian P Rubin, Douglas B Flieder, and Chandrajit P Raut. Gastrointestinal stromal tumors. *Textbook of Uncommon Cancer*, pages 470–492, 2017.

[95] Daniël Vos and Sicco Verwer. Efficient training of robust decision trees against adversarial examples. In *International Conference on Machine Learning*, pages 10586–10595. PMLR, 2021.

[96] Ming Wang, Jia Xu, Yun Zhang, Lin Tu, Wei-Qing Qiu, Chao-Jie Wang, Yan-Ying Shen, Qiang Liu, and Hui Cao. Gastrointestinal stromal tumor: 15-years' experience in a single center. *Bmc Surgery*, 14:1–10, 2014.

[97] Hui Yan, Pierre Marchettini, Yair IZ Acherman, Sue Ann Gething, Erwin Brun, and Paul H Sugarbaker. Prognostic assessment of gastrointestinal stromal tumor. *American journal of clinical oncology*, 26(3):221–228, 2003.

[98] Xue Ying. An overview of overfitting and its solutions. In *Journal of physics: Conference series*, volume 1168, page 022022. IOP Publishing, 2019.

[99] Zhiliang Ying, Sin-Ho Jung, and Lee-Jen Wei. Survival analysis with median regression models. *Journal of the American Statistical Association*, 90(429):178–184, 1995.

[100] TL Znajda, JS Wunder, RS Bell, and AM Davis. Gender issues in patients with extremity soft-tissue sarcoma: a pilot study. *Cancer nursing*, 22(2):111–118, 1999.