Universität
Augsburg
University

Fakultät für Angewandte Informatik

Lehrstuhl für Organic Computing

Disentangling the Model Selection Tasks for Improved Explainability in a Rule-based Machine Learning System

Inaugural-Dissertation zur Erlangung des Doktorgrades an der Fakultät für Angewandte Informatik der Universität Augsburg

Michael Heider

2024

Erstgutachter:	Prof. Dr. Jörg Hähner
Zweitgutachter:	Prof. Dr. Bernhard Bauer
Drittgutachter:	Prof. Dr. Sven Tomforde
Tag der mündlichen Prüfung:	13. Februar 2025

Abstract

With the increasing capabilities of machine learning (ML) and other artificial intelligence (AI) methods comes a growing interest from many fields of application to employ these methods to increase automation of work tasks and improve the efficiency and effectiveness of operations. However, the systems will only see effective use if they are trusted by those responsible for the task itself. False predictions and flawed decisions can have detrimental effects, for example, on human life in medical applications or financial interest in industry. Therefore, it is reasonable for stakeholders of these systems to want to understand the reasoning of AI systems. Methods that can make this insight available to stakeholders have increasingly be summarized under the term explainable AI (XAI). While approaches exist towards making black-box models explainable, the use of inherently explainable models can be more straightforward and promising.

One family of algorithms producing inherently explainable models are Learning Classifier Systems (LCSs). Despite their name, they are a general rule-based ML (RBML) method and representatives for all major ML tasks have been proposed. To classify LCSs based on their mode of operation, this work introduces a new system that is more precise than the current state-of-the-art and based on descriptive ML terminology. While most researchers in the past have focused primarily on LCSs' algorithmic aspects, this work adopts a distinct perspective by approaching them through the lens of optimization. It discusses LCSs with regards to typical tasks involved in creating an ML model and what specific elements have to be optimized and how this is typically done. Critically, the task of model selection is usually performed by some metaheuristic component and involves the subtasks of *how many rules to use* and *where to place them*.

This work also proposes a template to assess use case–specific explainability requirements based on multiple stakeholders' inputs and extensively demonstrates its usage in a real-world manufacturing setting. There, stakeholders indeed request XAI models over black-box approaches and, according to their answers, LCSs should be a good fit. Additionally, the results laid out what LCS models in that application should look like which is, however, not achievable with the major state-of-the-art LCSs.

Therefore, a new LCS, called the Supervised Rule-based learning system (SupRB), is introduced in this work that is simpler than previous LCSs with clearer optimization objectives and models that can fulfil the stakeholders' requirements. In extensive testing on real-world data, SupRB demonstrates its capabilities of producing small yet accurate models that outperform those of well-established methods. This work also investigates numerous possible extensions for each component of SupRB with a special focus on its optimizers and presents the findings of the multiple studies in a comprehensive manner based on descriptive statistics, visualizations of results, and rigorous statistical testing. Then various paths for future research and application of SupRB are laid out which can advance the field of XAI considerably.

Preface

This thesis is the culmination of years of research that would not have been possible without the support of various people:

First, I want to thank my supervisor, Jörg Hähner, for his support throughout the years. Not only did he give me the opportunity to embark on this research in the first place but he also encouraged me to choose directions freely while still offering his guidance and assistance on potential steps. He always had an open ear for any issues. With another supervisor this work would not have been the same.

Next, my thanks go to my colleague and friend Helena Stegherr with whom I shared an office for the last 5.5 years and collaborated on numerous works making her my second most frequent co-author. Without her efforts in proofing my texts and her endless patience with my tendency to skimp on commas the readability would doubtlessly be worse than it is today. We shared many discussions on the possible next or future steps for SupRB and much of my understanding of metaheuristics is ultimately due to our joint efforts and her near infinite amount of knowledge in that field. I deeply hope that there is much joint work in the future.

The operational side of this research would not have been possible without the tireless efforts of my research assistant and long-time friend Roman Sraj. My requests for specific code changes or additional experiment runs have without doubt caused him many late evenings. Especially as we frequently shared late discussions and pair-programming sessions to improve SupRB. His further developments of the experiment code were especially crucial and noteworthy.

My knowledge of LCSs greatly benefited from the deep expertise of David Pätzel who has been a staple in this research community for years now. He shaped much of my early conceptions about SupRB and we frequently discussed advantages and disadvantages of different aspects of LCSs. I am glad to have co-organized IWLCS 2021/2022 and ERBML 2023 with him. He was also the one to originally recruit me for the OC group and is therefore ultimately responsible for this work. Specifically, I want to acknowledge his assistance in developing the statistical testing scheme for my publications and this thesis.

The implementation that was used for all experiments in this thesis is based on Jonathan Wurth's modular version of my concept of SupRB. This version enabled us to change individual components with much more ease and resolved many early issues with the code.

This work would have been impossible without the support of my family throughout the most stressful times. Their encouragements and belief in me kept me steadfast on this path.

I also want to thank my other co-authors and colleagues that have not yet been mentioned for our collaborations over the last few years. Even if these joint efforts had less impact on the specifics of this thesis they were nonetheless insightful and a source of joy.

Finally, I gratefully acknowledge the resources on the LiCCA HPC cluster—DFG Project-ID 499211671—which was essential to be able to complete this thesis extensive experiments.

Publications of the Author

[CHH24] Henning Cui, Michael Heider, and Jörg Hähner. “Positional Bias Does Not Influence Cartesian Genetic Programming with Crossover”. In: *Parallel Problem Solving from Nature – PPSN XVIII*. Springer Nature Switzerland, 2024, pp. 151–167. ISBN: 9783031700552. DOI: 10.1007/978-3-031-70055-2_10.

[Cui+23] Henning Cui, Andreas Margraf, Michael Heider, and Jörg Hähner. “Towards Understanding Crossover for Cartesian Genetic Programming”. In: *Proceedings of the 15th International Joint Conference on Computational Intelligence - ECTA*. Ed. by Niki van Stein, Francesco Marcelloni, H. K. Lam, Marie Cottrell, and Joaquim Filipe. SCITEPRESS - Science and Technology Publications, 2023, pp. 308–314. DOI: 10.5220/0012231400003595.

[GHH23] Markus Görlich-Bucher, Michael Heider, and Jörg Hähner. “Predicting Physical Disturbances in Organic Computing Systems Using Automated Machine Learning”. In: *Architecture of Computing Systems*. Vol. 13949. Springer Nature Switzerland, 2023, pp. 48–62. ISBN: 9783031427855. DOI: 10.1007/978-3-031-42785-5_4.

[Gör+23] Markus Görlich-Bucher, Michael Heider, Tobias Ciemala, and Jörg Hähner. “A Decision-Theoretic Approach for Prioritizing Maintenance Activities in Organic Computing Systems”. In: *Architecture of Computing Systems*. Vol. 13949. Springer Nature Switzerland, 2023, pp. 37–47. ISBN: 9783031427855. DOI: 10.1007/978-3-031-42785-5_3.

[Hei+22a] Michael Heider, Helena Stegherr, David Pätzel, Roman Sraj, Jonathan Wurth, Benedikt Volger, and Jörg Hähner. “Approaches for Rule Discovery in a Learning Classifier System”. In: *Proceedings of the 14th International Joint Conference on Computational Intelligence - ECTA,* INSTICC. Setúbal, Portugal: SciTePress, 2022, pp. 39–49. DOI: 10.5220/0011542000003332.

[Hei+22b] Michael Heider, Helena Stegherr, Jonathan Wurth, Roman Sraj, and Jörg Hähner. “Investigating the Impact of Independent Rule Fitnesses in a Learning Classifier System”. In: *Bioinspired Optimization Methods and Their Applications*. Ed. by Marjan Mernik, Tome Eftimov, and Matej Črepinšek. Cham: Springer International Publishing, 2022, pp. 142–156. ISBN: 978-3-031-21094-5. DOI: 10.1007/978-3-031-21094-5_11.

[Hei+22c] Michael Heider, Helena Stegherr, Jonathan Wurth, Roman Sraj, and Jörg Hähner. “Separating Rule Discovery and Global Solution Composition in a Learning Classifier System”. In: *Genetic and Evolutionary Computation Conference Companion (GECCO ’22 Companion)*. 2022. DOI: 10.1145/3520304.3529014.

[Hei+23a] Michael Heider, David Pätzel, Helena Stegherr, and Jörg Hähner. “A Metaheuristic Perspective on Learning Classifier Systems”. In: *Metaheuristics for Machine Learning: New Advances and Tools*. Ed. by Mansour Eddaly, Bassem Jarboui, and Patrick Siarry. Singapore: Springer Nature Singapore, 2023, pp. 73–98. ISBN: 978-981-19-3888-7. DOI: 10.1007/978-981-19-3888-7_3.

[Hei+23b] Michael Heider, Helena Stegherr, Richard Nordsieck, and Jörg Hähner. "Assessing Model Requirements for Explainable AI: A Template and Exemplary Case Study". In: *Artificial Life* 29.4 (2023), pp. 468–486. ISSN: 1064-5462. DOI: 10.1162/artl_a_00414.

[Hei+23c] Michael Heider, Helena Stegherr, David Pätzel, Roman Sraj, Jonathan Wurth, Benedikt Volger, and Jörg Hähner. "Discovering Rules for Rule-Based Machine Learning with the Help of Novelty Search". In: *SN Computer Science* 4.6 (2023), p. 778. ISSN: 2661-8907. DOI: 10.1007/s42979-023-02198-x.

[Hei+23d] Michael Heider, Helena Stegherr, Roman Sraj, David Pätzel, Jonathan Wurth, and Jörg Hähner. "SupRB in the context of rule-based machine learning methods: A comparative study". In: *Applied Soft Computing* 147 (2023), p. 110706. ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2023.110706.

[Hei+24] Michael Heider, Maximilian Krischan, Roman Sraj, and Jörg Hähner. "Exploring Self-Adaptive Genetic Algorithms to Combine Compact Sets of Rules". In: *2024 IEEE Congress on Evolutionary Computation (CEC).* 2024, pp. 1–8. DOI: 10.1109/CEC60901.2024.10612101.

[Hei19] Michael Heider. "Increasing reliability in FDM manufacturing". In: *Informatik 2019: 50 Jahre Gesellschaft für Informatik - Informatik für Gesellschaft (Workshop-Beiträge).* Ed. by C. Draude, M. Lange, and B. Sick. Gesellschaft für Informatik e.V., 2019. ISBN: 978-3-88579-689-3. DOI: 10.18420/inf2019_ws52.

[Hei24] Michael Heider. "Towards Self-Explaining Assistance Systems in Tomorrow's Factories". In: *Organic Computing—Doctoral Dissertation Colloquium 2023.* Ed. by Sven Tomforde and Christian Krupitzer. Kassel University Press, 2024. DOI: 10.17170/kobra-202402269661.

[HNH21] Michael Heider, Richard Nordsieck, and Jörg Hähner. "Learning Classifier Systems for Self-Explaining Socio-Technical-Systems". In: *Proceedings of LIFELIKE 2021 co-located with 2021 Conference on Artificial Life (ALIFE 2021).* Ed. by Anthony Stein, Sven Tomforde, Jean Botev, and Peter Lewis. 2021.

[HPH20] Michael Heider, David Pätzel, and Jörg Hähner. "Towards a Pittsburgh-style LCS for learning manufacturing machinery parametrizations". In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion.* ACM, July 2020. DOI: 10.1145/3377929.3389963.

[HPW22] Michael Heider, David Pätzel, and Alexander R. M. Wagner. "An overview of LCS research from 2021 to 2022". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion.* 2022, pp. 2086–2094. DOI: 10.1145/3520304.3533985.

[Kem+23] Neele Kemper, Michael Heider, Dirk Pietruschka, and Jörg Hähner. "Forecasting of residential unit's heat demands: a comparison of machine learning techniques in a real-world case study". In: *Energy Systems* (May 2023). ISSN: 1868-3975. DOI: 10.1007/s12667-023-00579-y.

[Mei+23] Lukas Meitz, Michael Heider, Thorsten Schöler, and Jörg Hähner. “On Data-Preprocessing for Effective Predictive Maintenance on Multi-Purpose Machines”. In: *Proceedings of the 12th International Conference on Data Science, Technology and Applications.* Ed. by Oleg Gusikhin, Slimane Hammoudi, and Alfredo Cuzzocrea. SCITEPRESS - Science and Technology Publications, 2023, pp. 606–612. doi: 10.5220/0012146700003541.

[Mei+24] Lukas Meitz, Michael Heider, Thorsten Schöler, and Jörg Hähner. “A Taxonomy for Complexity Estimation of Machine Data in Machine Health Applications”. In: *Proceedings of the 21st International Conference on Informatics in Control, Automation and Robotics.* SCITEPRESS - Science and Technology Publications, 2024, pp. 341–350. isbn: 978-989-758-717-7. doi: 10.5220/0012994900003822.

[MHH16] Sebastian von Mammen, Heiko Hamann, and Michael Heider. “Robot gardens: an augmented reality prototype for plant-robot biohybrid systems”. In: *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology.* Ed. by Dieter Kranzlmüller and Gudrun Klinker. VRST ’16. ACM, 2016, pp. 139–142. isbn: 9781450344913. doi: 10.1145/2993369.2993400.

[Nor+19] Richard Nordsieck, Michael Heider, Andreas Angerer, and Jörg Hähner. “Towards Automated Parameter Optimisation of Machinery by Persisting Expert Knowledge”. In: *Proceedings of the 16th International Conference on Informatics in Control, Automation and Robotics – Volume 1: ICINCO.* INSTICC. SciTePress, 2019, pp. 406–413. isbn: 978-989-758-380-3. doi: https://doi.org/10.5220/0007953204060413.

[Nor+20] Richard Nordsieck, Michael Heider, Andreas Angerer, and Jorg Hahner. “Evaluating the Effect of User-Given Guiding Attention on the Learning Process”. In: *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS).* Ed. by Esam El-Araby et al. IEEE, 2020, pp. 215–221. isbn: 97817281-72774. doi: 10.1109/acsos49614.2020.00044.

[Nor+21] Richard Nordsieck, Michael Heider, Anton Winschel, and Jörg Hähner. “Knowledge Extraction via Decentralized Knowledge Graph Aggregation”. In: *15th IEEE International Conference on Semantic Computing, ICSC 2021, Laguna Hills, CA, USA, January 27-29, 2021.* IEEE, 2021, pp. 92–99. doi: 10.1109/ICSC50631.2021.00024.

[Nor+22a] Richard Nordsieck, Michael Heider, Alwin Hoffmann, and Jörg Hähner. “Reliability-based Aggregation of Heterogeneous Knowledge to Assist Operators in Manufacturing”. In: *2022 IEEE 16th International Conference on Semantic Computing (ICSC).* IEEE, Jan. 2022. doi: https://doi.org/10.1109/ICSC52841.2022.00027.

[Nor+22b] Richard Nordsieck, Michael Heider, Anton Hummel, and Jörg Hähner. “A Closer Look at Sum-based Embeddings for Knowledge Graphs Containing Procedural Knowledge”. In: *Proceedings of the Workshop on Deep Learning for Knowledge Graphs (DL4KG 2022) co-located with the 21th International Semantic Web Conference (ISWC 2022).* Ed. by Mehwish Alam, Davide Buscaldi, Michael Cochez, Francesco Osborne, and Diego Reforgiato Recupero. 2022.

[Nor+22c] Richard Nordsieck, Michael Heider, Anton Hummel, Alwin Hoffmann, and Jörg Hähner. “Towards Models of Conceptual and Procedural Operator Knowledge”. In: *Proceedings of the First International Workshop on Semantic Industrial Information Modelling (SemIIM 2022) co-located with the 19th Extended Semantic Web Conference ESWC 2022*. Ed. by Arild Waaler, Evgeny Kharlamov, Baifan Zhou, and Dongzhuoran Zhou. 2022.

[Nor+23] Richard Nordsieck, André Schweizer, Michael Heider, and Jörg Hähner. *PDPK: A Framework to Synthesise Process Data and Corresponding Procedural Knowledge for Manufacturing*. Available on arXiv: https://arxiv.org/abs/2308.08371. 2023.

[PHH23] David Pätzel, Michael Heider, and Jörg Hähner. “Towards Principled Synthetic Benchmarks for Explainable Rule Set Learning Algorithms”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO ’23 Companion)*. ACM, 2023. DOI: 10.1145/3583133.3596416.

[PHW21] David Pätzel, Michael Heider, and Alexander R. M. Wagner. “An overview of LCS research from 2020 to 2021”. In: *GECCO ’21: Genetic and Evolutionary Computation Conference, Companion Volume, Lille, France, July 10-14, 2021*. Ed. by Krzysztof Krawiec. ACM, 2021, pp. 1648–1656. DOI: 10.1145/3449726.3463173.

[SHH22] Helena Stegherr, Michael Heider, and Jörg Hähner. “Classifying metaheuristics: towards a unified multi-level classification system”. In: *Natural Computing* 21 (2022), pp. 155–171. DOI: 10.1007/s11047-020-09824-0.

[SHH23] Helena Stegherr, Michael Heider, and Jörg Hähner. “Assisting convergence behaviour characterisation with unsupervised clustering”. In: *Proceedings of the 15th International Joint Conference on Computational Intelligence, November 13-15, 2023, in Rome, Italy*. Ed. by Niki van Stein, Francesco Marcelloni, H. K. Lam, Marie Cottrell, and Joaquim Filipe. 2023, pp. 108–118. DOI: 10.5220/0012202100003595.

[Sid+24] Abubakar Siddique, Michael Heider, Muhammad Iqbal, and Hiroki Shiraishi. “A Survey on Learning Classifier Systems from 2022 to 2024”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO ’24 Companion. Melbourne, VIC, Australia: Association for Computing Machinery, 2024, pp. 1797–1806. ISBN: 9798400704956. DOI: 10.1145/3638530.3664165.

[Ste+21] Helena Stegherr, Michael Heider, Leopold Luley, and Jörg Hähner. “Design of large-scale metaheuristic component studies”. In: *GECCO ’21: Genetic and Evolutionary Computation Conference, Companion Volume, Lille, France, July 10-14, 2021*. Ed. by Krzysztof Krawiec. ACM, 2021, pp. 1217–1226. DOI: 10.1145/3449726.3463168.

[Ste+23] Helena Stegherr, Leopold Luley, Jonathan Wurth, Michael Heider, and Jörg Hähner. *A framework for modular construction and evaluation of metaheuristics*. Tech. rep. 2023-01. Fakultät für Angewandte Informatik, 2023.

[Wie+21] Andreas Wiedholz, Michael Heider, Richard Nordsieck, Andreas Angerer, Simon Dietrich, and Jörg Hähner. “CAD-based Grasp and Motion Planning for Process Automation in Fused Deposition Modelling”. In: *Proceedings of the 18th International Conference on Informatics in Control, Automation and Robotics, ICINCO 2021, Online Streaming, July 6-8, 2021.* Ed. by Oleg Gusikhin, Henk Nijmeijer, and Kurosh Madani. SCITEPRESS, 2021, pp. 450–458. DOI: 10.5220/0010571204500458.

[Wit+23] Tobias Wittmeir, Michael Heider, André Schweiger, Michaela Krä, Jörg Hähner, Johannes Schilp, and Joachim Berlak. “Towards Robustness Of Production Planning And Control Against Supply Chain Disruptions”. In: *Proceedings of the Conference on Production Systems and Logistics: CPSL 2023.* Ed. by David Herberger, Marco Hübner, and Volker Stich. Hannover : publish-Ing., 2023, pp. 65–75. DOI: 10.15488/13425.

[Wur+22] Jonathan Wurth, Michael Heider, Helena Stegherr, Roman Sraj, and Jörg Hähner. “Comparing different Metaheuristics for Model Selection in a Supervised Learning Classifier System”. In: *Genetic and Evolutionary Computation Conference Companion (GECCO ’22 Companion).* 2022. DOI: 10.1145/3520304.3529015.

[Wur+23] Jonathan Wurth, Helena Stegherr, Michael Heider, Leopold Luley, and Jörg Hähner. “Fast, Flexible, and Fearless: A Rust Framework for the Modular Construction of Metaheuristics”. In: *Proceedings of the Companion Conference on Genetic and Evolutionary Computation.* GECCO ’23 Companion. Lisbon, Portugal: Association for Computing Machinery, 2023, pp. 1900–1909. ISBN: 9798400701207. DOI: 10.1145/3583133.3596335.

[Wur+24] Jonathan Wurth, Helena Stegherr, Michael Heider, and Jörg Hähner. “GRAHF: A Hyper-Heuristic Framework for Evolving Heterogeneous Island Model Topologies”. In: *Proceedings of the Genetic and Evolutionary Computation Conference.* GECCO ’24. Melbourne, VIC, Australia: Association for Computing Machinery, 2024, pp. 1054–1063. ISBN: 9798400704949. DOI: 10.1145/3638529.3654136.

Contents

List of Figures

List of Tables

1 Introduction

With the increasing mainstream use of “AI”—which is often even touted to be “everywhere” soon—that we saw in the last few years, a whole host of new and old questions regarding the use of “intelligent” agents also made its way into the public eye. While most of these questions have been discussed for decades within the dedicated research communities, they were of secondary importance even in adjacent fields where often the successful application was so challenging that considerations about alignment, safety, trust, collaboration, resource use, data ownership, anonymity, copyright, etc. remained under the radar. However, the advances of the last decade(s) brought about considerable improvements and led to the successful application of artificial intelligence (AI)[1] techniques in a large number of new domains.

With this successful application in (applied) research came a large interest within various industries about the integration of such systems into their operation. However, this usually comes with a large number of challenges:[2] First is the lack of usable training data, exacerbated through the unique and often quite complex software landscapes where even existing databases were rarely sufficiently linked and a lot of data was never stored or even never digitally available to begin with. Many companies have been working for a few years now to improve at least that issue, although the success was often mixed and incomplete. Second, the data was often not sufficiently cleaned and verified and thus often misses elements, is mislabelled, shows unrealistic readings, is subject to heavy noise, or records unintended values (e.g. sensors were not installed at the correct location). Third, systems are often already very well-optimized and have been fine-tuned by engineers over decades and thus contain large imbalances between classes of data, e.g. predictive maintenance struggles with the lack of data about machines breaking down as maintenance intervals are usually shorter than the expected time until a breakdown to safe on costs. Fourth, unclear paths to value generation often lead

[1]For this thesis, I assume a rather broad definition of *AI*. In my view, this term should not be limited to machine learning models or even only to LLMs, but rather encompass the applications of other fields such as optimization (and with it, for example, evolutionary computation), as well as other forms of complex decision making systems. Usually, multiple different methods and models will together form an AI-based agent with which humans will interact, either on the physical plane, e.g. autonomous driving and intelligent robots, or in virtual space, e.g. the recently widespread and intensely discussed ChatGPT by OpenAI. However, I want to stress that—at least in my view—the currently publicly available systems are not yet truly intelligent even if human-competitiveness is regularly achieved (cf., for example, the yearly human-competitiveness challenge at GECCO (https://gecco-2024.sigevo.org/Humies)).

[2]All of this insight is based on extensive discussion with experts and practitioners over the last few years: Many in joint projects where we worked on this together, some based on less formal exchanges, and many also in the scope of the KI-Produktionsnetzwerk Augsburg (https://www.kiproduktionsnetzwerk.de).

to missing investments that could fix the issues above. Fifth is that in many cases where personalized data is concerned it is not clear whether data can even be used for training due to privacy laws.

But even after these (and many of the other unnamed issues on the path towards AI application) have been fixed, there is a large issue within the application of modern black-box models (e.g. deep learning): Stakeholders struggle to verify the correctness of these systems in general, as well as of individual predictions. Yet, in most settings, stakeholders will be held responsible for the successful fulfilment of their tasks. Thus, if a stakeholder were to rely on a model/agent making a prediction and that prediction was wrong, they would be considered at fault. As some decisions can have large monetary impact, e.g. misrouting an important shipment, or even harm human lives, e.g. missing a cancer in a vital organ or removing an organ based on a wrong diagnosis, stakeholders will usually be quite wary about an active usage of such systems. While this concern has been discussed within the research community (and some limited industries, such as insurance) for decades, these issues have only become relevant for the mainstream with the recent successes of finally solving many tasks with average performances similar to humans.

One possible solution to this problem is the use of explainable AI (XAI) (for an in-depth discussion of XAI, cf. also Chapter 3 and [e.g. Bar+20; Bac+22]). In general, the field of XAI tries to develop methods that directly create transparent systems, transform black-box models into such systems, or make them otherwise explainable. However, there seems to be no general-purpose solution to XAI even if we only considered supervised learning [DK18]. Instead, the correct approach to XAI should be made on a case-by-case basis, including the verification if XAI is even needed or if the correct selection and presentation of metrics are sufficient to satisfy concerned stakeholders.

Within XAI, one approach is to directly train models or build systems that are human verifiable as they are transparent and interpretable. Easy to interpret models, such as Linear Regression or Decision Trees (DTs), are often much more limited in their capabilities to solve the relevant learning tasks and can also become too complex to interpret [Bar+20]. For example, a tree with depth n would split the data up to n times based on one of the features within each path to the leaf node, where each split introduces two new paths resulting in a tree with up to 2^n leaves. Arguably, even smaller depths reach sizes where the tree is only theoretically interpretable but practically impossible to analyse. Imposing size restrictions further limits their ability to approximate the training data well, however, it can also lead to less overfitting. Despite this, the key advantages of directly training interpretable models are that they approximate the real data rather than approximating what a black-box model envisions the data to be like and that they allow much more in-depth inspections of the model.

Highly regarded among the interpretable models are rule-based machine learning (RBML) models [Bar+20]. All of these models utilize rules that are trained from data and can be read directly by a human inspecting the model structure. The specific rule design and how the individual rules are combined to form the model and make predictions varies between the individual RBML branches and a large variety of training schemes has been proposed in the past. The most commonly applied RBML methods are tree-based models (based on DTs). However, some of DT's improvements towards better predictions come at a steep cost regarding their explainability. XGBoost and Random Forests (RFs) are often used improvements based on the concepts of boosting and bagging (cf. Section 2.2.3), respectively, but they are usually impossible to interpret due to the large number of—often also complex—trees that make up these models.

Another group of RBML models with a long research history are *Learning Classifier Systems (LCSs)* [Hei+23a].[3] An in-depth discussion of LCSs is found in Chapter 2. In short, they can be seen as a generalization of DTs where the restriction of non-overlapping rule responsibilities has been relaxed. Therefore, one point of the feature space can now be matched by more than one rule. This allows for smoother function approximation but comes at the cost of more potential variety in the rules. However, the individual rules remain human readable (cf. Table 5.4 for an example). This should make them a powerful option for XAI, however, they are rarely considered in more mainstream sources outside of the evolutionary computation community. Possible reasons often discussed in the community include a simple lack of notoriety (most researchers in the (X)AI communities are not aware of LCSs), a lack of efficient implementations, training schemes that are perceived as overly complex, and too few dedicated researchers to keep up with the advances in other subfields of AI. Yet, it should be noted that the recent boom in XAI did bring new interest to the field and effective placement of the expected successful applications of LCSs for XAI should improve the notoriety of LCSs as well.

The use of LCSs has been proposed for a large variety of different applications from manufacturing [Hei+23b] to medicine [Woo+24].[4] A notable LCSs application from the field of intelligent systems are Organic Computing (OC) systems [MSU11; MT17]. OC systems (and similar approaches from neighbouring fields) closely relate to what the public envisions AI to be like,[5] which is most often less a software system that operates on some obscured optimization or prediction task but rather a cyber-physical system where an intelligent piece of software interacts with humans, other systems, and the environment through sensors and actuators. Within OC systems, the usage of LCSs has been discussed and actualized for a long time, with a special focus on their explainability to humans in safety-critical applications such as traffic management [STH16]. To this day, they remain an important field for the usage of LCSs in the real world [Kru+22].

1.1 Research Gaps and Contributions

Despite their quite obvious potential for widespread application within next generation AI systems that will likely be focused on assisting humans rather than fully taking over,[6] LCSs are rarely considered as the decision maker inside these systems (with the notable exception of

[3]Other RBML methods are highlighted in Section 2.2.
[4]The LCSs surveys [UM09; PSN20b; PHW21b; HPW22; Sid+24] all feature a variety of additional applications.
[5]Given the boom of ChatGPT and similar "AI applications" this view might have shifted recently.
[6]At least this is the sentiment conveyed by both potential users as well as management that we regularly encounter within industry.

OC systems). Reasons such as a low notoriety and therefore a lack of highly optimized implementations and dedicated researchers in the field are often discussed within the community but do not paint a full picture:

The field of LCSs is highly complex and relatively fractured. LCS researchers have worked on all major types of learning tasks and proposed a large number of changes to the existing systems [UM09; PSN20b; PHW21b; HPW22; Sid+24]. However, the major survey from Urbanowicz and Moore [UM09] was still the most up-to-date source aimed at researchers rather than students [UB17]. The effects of the metaheuristic, usually an evolutionary algorithm (EA), on the training process are mostly discussed (if at all) from an algorithmic perspective rather than on a conceptual level, even though it is crucial for the training success. Therefore, the *first contribution* of this thesis (*C1*) is an elaborate *metaheuristic perspective on LCSs*, including

- a reintroduction of what LCSs are,
- what their models' structure is after training (rather than focussing on the training scheme which is most common when for example discussing "what is an LCS"),
- how this fits into the common machine learning (ML) terminology of *model selection* and *model fitting*,
- which of these tasks can be and usually are solved by EAs,
- how LCSs relate to techniques producing similar models,
- an updated classification scheme that replaces the old and imprecise system of "Pittsburgh-style", "Michigan-style" and "Hybrid" which was based on original conception of the idea rather than solid ML terminology,
- an in-depth view on the representations on which metaheuristics must operate,
- a summary of different operators and fitness functions useful to find good rules and models, and
- an overview of systems that function quite similar to LCSs, building the same type of model, but do not call themselves LCSs, for example because they use different optimizers than EAs.

Most of these discussions/perspectives are novel and should help as a useful source for readers of all knowledge levels about LCSs. They are based on joint work original published as [Hei+23a] and are featured in Chapter 2.

While it is quite clear that few rules make better explainable models than tens of thousands of rules, the actual number of rules and what they should ideally look like has not been precisely determined. Most past research regarding LCSs has focussed on minimizing the prediction errors and was—quite commonly—not done on real-world data or use cases. While some useful developments were made from this, oftentimes the models became increasingly complex, producing ever larger models.[7] This led me to doubt their practical applicability as an XAI system in the real-world. The *second contribution* of this thesis (*C2*) is an easy-to-use *catalogue of questions for stakeholders of a prospective XAI system*, e.g. as part of an intelligent agent assisting humans in their work, that *determines what form a model created by an LCSs should have*. It features seven questions from an abstract and general level to details around the structure of the LCS model itself. For its successful application, all relevant stakeholders (not only the direct users) of a system should answer these questions. This also has the benefit that they feel included in the process of creating these XAI-based systems which can in itself already increase the trust in them. The questionnaire itself is presented in Section 3.4. An extensive presentation of its successful application within a real-world scenario is given in Section 3.5. This includes not only the answers which confirm that explainability is indeed important and an outright requirement for any system used in this scenario, but also a meta-discussion on the consequences for designing an LCS that produces such models (cf. Section 3.5.6). This also forms a crucial part of *C2*. We can expect that the answers will vary slightly in other domains but it is plausible that the general trend, e.g. models should feature small numbers of rules, will be similar. Regardless, the questionnaire should be used whenever a new domain (or even only a different plant from the same company) is encountered.

From *C1* and *C2*, I gained some critical insights into the existing LCSs, their optimization approach (training scheme), and their applicability for XAI tasks.[8] I found that:[9]

1. The fitness of rules within LCS models, which is critical for guiding the optimizer during model selection (determining the number of rules to use and the allocation of the input space, i.e. the "if-part" of each rule), is not independent of the fitness of other rules in major LCSs. Most single-solution online learning (Michigan-style) systems, such as XCS, use fitness sharing mechanisms within a respective "niche", which could lead to rules' fitnesses being assessed wrongly and not representative of their true prediction accuracy. Crucially, it also makes the fitness value harder to explain. Usually, stakeholders will be interested in the performance of a specific rule but with a niche-based fitness we cannot view that in isolation. Most multi-solution batch learning (Pittsburgh-style) systems assign one fitness value to a whole set of rules. This leads to individual rules not having any clearly defined fitness/accuracy/"quality". Critically, this causes the search process to be more random and less efficient. When a rule is removed from an individual, it is purely removed by chance rather than based on whether it is expected to make a valuable contribution. Also, very good rules can be altered negatively without the meta-

[7]For example, many articles featuring the XCS classifier system (XCS) [Wil95] utilize models using thousands to tens of thousands of rules (macro-classifiers) to solve their proposed tasks.

[8]Please note that I focus primarily on supervised batch learning. While many of the insights/arguments will be applicable for online, unsupervised, and reinforcement learning as well, they might have to be adjusted slightly.

[9]While I try to keep this list as self-contained as possible, I will have to argue based on some of the core concepts of LCSs and methods employed in specific systems. If a reader is wholly unaware of LCSs, I recommend to start by reading at least Section 2.1 and the beginning of Section 2.3, i.e. Section 2.3.1 (although the entire section could be helpful), before continuing here.

heuristic noticing.[10] Especially in more complex learning tasks, this can lead to a loss in optimization guidance and the addition of unnecessary rules to the current solution(s).

2. The training mechanisms are often quite complex, involving multiple steps with different mechanisms influencing the same aspect of model selection, e.g. in XCS, rules are created by covering and the EA which both determine the allocation of feature space to specific rules and increase the number of rules in the model, which is then also potentially influenced by the deletion mechanism. Then there are aspects such as subsumption, which combines redundant rules, also influencing both aspects of model selection (as defined in Section 2.1.3 and paraphrased above). I would argue that the training of many LCSs is most often not guided by how to effectively apply optimization techniques but rather organically grown from an experimental algorithm design perspective. This makes these systems difficult to troubleshoot and optimize and hinders explaining them to others, and while most stakeholders will probably not want detailed explanations about the training process itself (cf. Section 3.5.6), this stops interested (applied) researchers—or even data scientists within industry that are searching for a usable ML approach—from considering most major LCSs.

3. Existing LCSs lack mechanisms to keep models as small as expected from stakeholders (often off by orders of magnitude). While subsumption and compaction of XCS and a fitness penalty in multi-solution batch learners can reduce the number of rules, their application is neither straightforward nor 100% effective.[11] The training processes will almost always generate models with inefficient or outright redundant rules on more complex learning tasks.

4. We may only use rules as simple as possible, e.g. with hyperrectangular matching and linear local models, within our LCS models, even though more complex options have been shown in the past to achieve better results in terms of predictive power and generalization (cf. Section 6.1.2 and Section 6.2 for a discussion of these options).

This leads to the *third and main contribution* of this thesis (*C3*): a new LCS algorithm that efficiently learns a simple and small set of rules from data. Its crucial innovation is the separation of rule discovery from the composition of models (solutions to the learning task) from these rules, thereby disentangling the two model selection tasks from each other as the allocation of feature space to rules becomes independent from selecting an appropriate model size (number of rules). In this system, which we named the Supervised Rule-based Learning System (SupRB), each rule has a fitness value independent from other rules and the fitness of the model as a whole. This allows a much greater control over the different optimization tasks and operators that target their aspects more clearly and concisely. As the name suggests, this system is meant to be used for supervised learning.[12] Its rules are simple and human readable (as demonstrated

[10]Note that a single change might make fitness effects relatively obvious but due to the large number of parameters in modern ML models, usually, several changes will be made at the same time before the new fitness is assessed.

[11]Note that compaction is always applied post-hoc and its effectiveness is therefore limited by the quality of the current rule set.

[12]While we did only explore regression tasks within the publications that went into this thesis, I illustrate a path towards adapting the system to classification as well. Theoretically, regression should be more complex to solve

in Table 5.4) and its models are accurate yet small (cf. Section 5.2.2 for a benchmark-based comparison with other RBML approaches). A detailed description of SupRB is given in Section 4.1. In addition to proposing the system itself, we also worked on a multitude of possible improvements I present in Chapter 6. These should largely be seen as part of *C3* which therefore also covers a deep look into all of the major aspects of SupRB.

Overall, SupRB should enable the usage of LCSs for XAI in a way that was impossible with previous systems.

1.2 How my Previous Publications Fit into this Thesis

After introducing the research gaps and the contributions this thesis makes to fill them, I want to take this section to specifically highlight how my published articles fit into this. Of course, I published a lot more (together with my colleagues and co-authors) than the works that were directly relevant to this thesis, but I will limit the following to the relevant publications and only cite my own work in it to make it easier to read.[13] Each section within this thesis that was at least partially published before will also signify this to the reader, yet this section should serve as a general overview. A late-stage concept for this thesis was—together with a short overview of the then available results—presented at the 2023 edition of the OC-DDC [Hei24].

While working on *C1*, which is a metaheuristic (and therefore optimization-driven) perspective on LCSs (cf. Section 1.1), a first step was to gain some clarity on what really sets different metaheuristics apart. For this, we proposed a unified multi-level classification system that allows to assess whether metaheuristics are really different from each other (and how different they are) beyond the consideration of only different metaphors [SHH22]. While this also motivated a proposal towards large-scale studies on components [Ste+21a], its primary impact on this thesis is the understanding it offers towards how we should approach optimization—and specifically research focused on optimization—within the field of LCSs. Additionally, our surveys on LCS literature [PHW21b; HPW22; Sid+24] improved my theoretical understanding and knowledge about the state-of-the-art and help convey these to other researchers. While featuring them in detail in this thesis would take away from the focus of this work, their results still informed many decisions I made. Finally, the majority of Chapter 2 (and with it the comprehensive presentation of *C1*) was originally published in [Hei+23a].

The explainability questionnaire and its answers from a first case study, which form *C2* and make up the largest part of Chapter 3, was published in two stages: First, we published the original questions as part of the workshop on Lifelike Computing Systems [HNH21]. Then, we published the questionnaire together with a real-world case study from the manufacturing domain in the Journal of Artificial Life as an extension of the first paper [Hei+23b]. This case study report contained the answers of a large and diverse set of stakeholders and demonstrates

than classification, which is why we focused on that in our extensive experimentation described in the latter chapters of this thesis.

[13]A full list of my current publications is found before the main body of this thesis right before the table of contents.

the usefulness of these question. Additionally, we made some assumptions about model design for LCSs from that.

The majority of this work—and therefore a large share of the relevant publications—falls under *C3*, which fills the research gap I found based on *C1* and *C2* and is presented in Section 1.1. This begins with the proposal of a new style of LCSs, named SupRB, which we introduced at GECCO [Hei+22c].[14] We originally benchmarked this system against a modern version of XCSF on a variety of learning tasks and I presented the results at BIOMA [Hei+22b]. An extension of this benchmark was made for the Journal of Applied Soft Computing, which also included DTs and RFs [Hei+23d] and went into more detail (cf. Chapter 5). Based on our findings and theoretical understanding of SupRB and the optimization of LCSs, we made a number of extensions to the system. First, we investigated whether other metaheuristics perform better than the originally proposed genetic algorithm (GA) for solution composition [Wur+22]. Then, we tested the use of novelty search–based evolution strategies for the rule discovery component of SupRB, which I presented at ECTA, where it won best paper [Hei+22a]. We were subsequently invited to submit an extension of this article to Springer Nature Computer Science in which we included different archiving techniques for the novelty search approaches and perform a comprehensive statistical analysis of the results [Hei+23c]. Ultimately and most recently, we revisited solution composition by investigating the effects of self-adaptive GAs which I presented at IEEE CEC [Hei+24].

1.3 Outline

After motivating my work (cf. Chapter 1), summarizing the existing research gaps and this thesis' contributions to fill these gaps (cf. Section 1.1), and presenting my relevant (co-authored) publications and how they fit into the context of the thesis (cf. Section 1.2), this section gives an outline of the remaining body of text, suggests what text is most essential and should be read at minimum, and gives some general pointers on how this thesis should be read in my view.

Chapter 2 provides a novel metaheuristic perspective on evolutionary RBML and LCSs specifically. Section 2.1 goes into detail on the concept of LCSs. A special focus is on the models they produce and which optimization tasks this entails from an ML-theoretic perspective, which is often overlooked in contemporary LCS research. Section 2.2 highlights the machine learning systems which produce models most similar to LCSs. In Section 2.3, I reintroduce our new concept of classifying LCSs, which should replace the old system of dividing them into Pittsburgh-style and Michigan-style which is based on which university first proposed a similar system but has no clear information about the model from an ML perspective. This section

[14]Originally, we tried to approach an abstracted and generalized version of parametrization of industrial machinery (which is a central aspect of [HNH21; Hei+23b], *C2*, and Chapter 3) by using a traditional "Pittsburgh-style" system. We presented that at GECCO two years earlier and called it SupRB-1 [HPH20]. However, we found that the system could not be improved to the level we would need for practical application, which then spawned the entirely novel approach of SupRB.

also introduces how a metaheuristic working on model selection tasks in LCSs (which is where they are usually employed) typically represents an individual, common operators, and fitness functions. Finally, Section 2.4 presents some other ML systems that could be considered LCSs but usually are not for various reasons.

Next, I provide insights into the concept of explainability (which was already motivated in Chapter 1) and a thorough case study towards assessing specific requirements systems should fulfil in applied use cases in Chapter 3. This chapter starts with Section 3.1, which takes a first look at some of the existing explainability methods of RBML and some options to measure this more abstract concept based on hard metrics. Section 3.2 introduces a typical manufacturing setting for which I motivate why we need an XAI system rather than a black-box ML approach to increase automation further and then argue why LCSs are especially suited for this in Section 3.3. A key contribution of this work is a template to assess the requirements XAI models need to fulfil to be useful in a given setting. This is presented as a seven question questionnaire in Section 3.4 that should involve all relevant stakeholders. The questionnaire is then used in a real-world case study at a plant of REHAU SE and I present the answers as well as what this means for what we want LCS models to look like in Section 3.5.

Based on the insights of Chapter 2 and Chapter 3, I then present the new evolutionary RBML system SupRB in Chapter 4. I start with a general description including visualizations and pseudocode in Section 4.1, which provides a full yet compact description of all necessary details. Then, I discuss the systems computational complexity including a critical discussion on why O-notation might be misleading here in Section 4.2. Section 4.3 goes into detail how SupRB can fulfil the explainability requirements laid out earlier in Section 3.5.6 and how explainable the system generally is—including discussions on both training and inference or model inspection. It also critically assesses the current limitations of the system in that regard, which concerns most other LCSs and many RBML systems as well.

After these extensive yet rather theoretical deliberations on SupRB, Chapter 5 benchmarks SupRB against its closest competing systems from the field of RBML that are still in widespread use (XCSF, DT, and RF). I start by revisiting the explainability of RBML models, although with a different focus than in the previous sections in Section 5.1. The experimental setup and our results, which confirm the hypotheses (SupRB is more accurate than DT but less than RF and equal to XCSF, but is more compact than XCSF and much more than RF while being competitive to DT), are presented in Section 5.2 in extensive graphical and tabular form and supported by sound Bayesian statistical testing. As the confirmation that SupRB is indeed a working system that is also significantly different from other RBML approaches was a central element to justify further testing and development of SupRB, I summarize the key points of this benchmark in Section 5.3.

The largest chapter of this work, Chapter 6, presents our extensive experiments on improving SupRB, where we extended the basic system from Chapter 4 with a variety of options for all major components of the system and tested them in a variety of benchmarks. I keep the basic experimental setup of Chapter 5 and use similar graphical and statistical tools to make the comparisons easier to follow for the reader. Each rule is made up of two major parts and some performance metrics. The first part, the condition or matching function, determines which subspace of the feature space the rule is responsible for. I present experiments based on the internal representation—which literature discussed as important for the optimization process in XCS—of the interval-based matching function and some discussion on the possibility and trade-offs of more complex functions in Section 6.1. The second part of a rule, its local model, is invoked on each matching rule to make a prediction on the data. I only used linear models for regression in this thesis, but feature an in-depth discussion on more options and their negative trade-offs with regards to explainability in Section 6.2. After determining all matching rules and getting their individual predictions, these have to be combined to form a system prediction. This is called mixing. In Section 6.3, I present a total of twelve different options for the mixing model of SupRB based on two assumptions: the impact of a rule's experience on its mixing weight should be capped (cf. Section 4.1 for the mixing model itself) and a prediction based on less rules is easier to read and analyse for humans (which was an insight of Section 3.5). After determining the makeup of rules and how we can combine them to create models, the next step is to create them, which is usually called rule discovery. I present six additional options for this based on Evolution Strategies with novelty search components in Section 6.4. The last major component of SupRB missing is the solution composition. This optimizer determines good subsets from the pool of rules. I first present experiments to determine whether the GA is a good metaheuristic for this or whether we should replace it with another mainstream optimizer in Section 6.5. Then, I investigate the usage of self-adaptive GAs in Section 6.6.

In Chapter 7, I discuss the potential avenues for further research based on my work. I highlight both practical approaches towards explainability and applied usage of SupRB, as well as further foundational and benchmarking-related considerations. For this, I make suggestions for all components of SupRB and how it could be embedded into an AI-based system or agent that interacts with non-scientists.

Finally, I summarize my findings and conclusions from the previous chapters in Chapter 8. In this, I critically assess the current state and capabilities of SupRB as well as its potential for improvement, but also its current shortcomings. Additionally, deliberations on the results of our case study as well as the questionnaire in general are included.

A Suggestion for Minimal Reading

Although I of course suggest to read the thesis cover to cover, I found it quite useful to have some guidance on which elements might be most relevant to me when reading similar works:

To readers already familiar with LCSs and other RBML approaches, I suggest to at least read Section 2.1.2 and Section 2.1.3, as well as Section 2.3.1 from Chapter 2, as these subsections contain the most relevant concepts to follow the essence of the later chapters and our most critical findings. Those that are new to the field will benefit most from this thesis if they include Sections 2.1 to 2.3 into their reading.

Readers familiar with the concept of explainability and LCSs in industry applications can focus their reading of Chapter 3 on Section 3.4 and Section 3.5.6. Although the introductory parts of

Section 3.5, which introduce the setting, the participants, and the general approach, might be interesting as well, the in-depth discussions of our findings are less relevant and most readers will probably take away most relevant info by starting with Section 3.5.5 and cherry-picking the relevant other parts of Section 3.5 that did not become sufficiently clear.

From Chapter 4, readers not familiar with SupRB should consider its description in Section 4.1, as the remaining work will make extensive reference to that and build on the presented concepts. Section 4.3 discusses explainability of SupRB specifically, so, to readers that want to either apply SupRB or research or improve the explainability of the system, I suggest to also read this section.

Readers that want to learn more about the explainability of RBML should consider Section 5.1, but most readers can focus on the key results of our benchmarks Section 5.3.

As Chapter 6 goes into a lot of detail and presents very different aspects of SupRB and its possible improvements, I recommend to read the parts that are individually most relevant, although I find that all results are insightful and relevant in their own right. I assume that traditional LCSs researchers will be most interested in Section 6.1 and Section 6.2, but readers that are most concerned with explainability will most likely not be able to pick up any of the options presented here and want to focus on the simplest possible rule design and can thus skip these sections. Section 6.3 might have some interesting aspects for both groups but is also rather technical and the real-world impact of such changes to the mixing model is not yet verified. Researchers from an optimization background, however, will probably focus their reading on Sections 6.4 to 6.6.

To all readers, I suggest Chapter 7 as it discusses the possible future paths within the field and SupRB specifically and of course Chapter 8 which summarizes and discusses this entire work.

How to Read this Thesis

As stated, I suggest reading this thesis front to back, but some adjustments could be made based on the previous segment. In general, I try to signify my own personal work and opinions by using *I*, whereas joint work (even if conceptualized and designed by myself) will usually involve *we* as it occurred with the relevant support of my respective co-authors or research assistants. I tried to keep the self-containedness of the chapters as high as possible and reintroduce relevant terminology in the individual chapters, although I will often have to assume knowledge about general ML, evolutionary computation, and LCS concepts and terminology, as well as the definitions of SupRB from Section 4.1. While the figures (especially related to the extensive statistical testing) will feature an introduction on how to interpret them, I suggest to consult the relevant surrounding text, especially from the sections where I used the respective concepts first, and—if applicable—the cited papers for deeper insights and explanations.

2 Evolutionary Rule-based Machine Learning—A Metaheuristic Perspective

In this thesis, I present my investigations into a new approach at learning compact rule sets with the help of metaheuristic-based machine learning (cf. Chapter 4). This was inspired by the considerable gaps in existing research as briefly laid out in Section 1.1. Therefore, this chapter will revisit previous work on that topic with a special focus on one of the first approaches in this direction [Hol76], Learning Classifier Systems (LCSs). Within this chapter, I will close a gap regarding the theory and perception of LCSs, especially concerning their optimization process, which equates to the first major contribution of this thesis, *C1*, as defined in Section 1.1. This chapter is based on joint work concerning the same topic published as a book chapter [Hei+23a].

Given that many out-of-field researchers (even within the evolutionary computation community) often find the term *LCS* confusing, as LCSs have been used for all major learning paradigms (supervised, unsupervised, and reinforcement learning) and a majority of research—including this thesis—is not (limited to) classification, we recently proposed [PHH23] to switch to the term Rule Set Learner (for the algorithm) or Rule Set Learning (for the concept), both abbreviated to RSL. The term is based on the fact that Learning Classifier Systems (LCSs) build sets of if-then rules to approximate models for any learning task. Within the LCSs community, it has already been somewhat common that some researchers use the more general and more descriptive term evolutionary rule-based machine learning (ERBML), but we think that RSL is beneficial to contrast models constructed by LCSs (or RSLs) from those by other rule-based systems, e.g. decision trees. Additionally, we would like to stress that—from our perspective—evolutionary optimization is not required but actually any metaheuristic may be employed In this thesis, I will use the terms LCS and RSL largely interchangeably but mostly focus on LCS for historic reasons. The traditional term is still too relevant to replace it, however, slowly phasing it out and using it primarily for legacy discussions and specific systems that still carry it in its name, e.g. the XCS classifier system (XCS), seems like the best approach to me.

Originally, LCSs were approaches to utilize Genetic Algorithms (GAs) for interactive tasks [e. g. Hol76; Wil95]. While, in the beginning, somewhat independently developed from reinforcement learning (RL), the two fields merged later. In this chapter, an in-depth view at LCSs from the metaheuristics side is provided. To our knowledge, this has not been done before [Hei+23a], despite the employed metaheuristic playing a central role in these methods. Indeed, no notable amount of research attention has so far been received by neither the metaheuristics used in LCSs nor their operators. [Hei+23a] (and by extension this chapter) aims to raise awareness of this fact and, by organizing existing research, hopefully provides inspiration for new developments in both metaheuristics and LCSs, thus, building a foundation for future research(ers) to more easily remedy this deficiency.

This chapter shortly introduces the field of LCSs, explaining the overall approach and discussing the learning and different optimization tasks that these systems are expected to solve (Section 2.1). The relation of LCSs to other similar machine learning (ML) approaches is described in Section 2.2 with a focus on highlighting similarities and differences and, thus, deepen the reader's understanding. This chapter's main contribution, the role of metaheuristics in LCSs, is elaborated on in Section 2.3 where the types of systems that different design choices result in are discussed and, in the process, a new classification system (instead of the often ambiguous but well-known Michigan/Pittsburgh one) is proposed. Furthermore, approaches to solution representation, metaheuristic operators, and fitness functions are laid out. Finally, recent ML approaches that employ metaheuristics and that were developed somewhat independently of LCSs but that rather closely resemble what we would call an LCS are presented (Section 2.4). Some of the techniques from these fields may be relevant for LCS as well.

2.1 What are Learning Classifier Systems?

While there is no official definition for what qualifies as an LCS, there appears to be a consensus [cf., e.g., Dru08; UB17] that an LCS constitutes an ML algorithm responsible for constructing an ensemble model composed of a collection of simpler *local models*. Each local model is designed to address a specific partition of the problem space. Importantly, this division of the problem space is not always rigid (resulting in disjoint sets), as local models may share overlapping responsibilities. This overlap is facilitated by associating each local model with a corresponding *matching function*, which, for any given input, determines whether that input falls within the domain modelled by the local model. Although the inclusion of overlapping responsibilities introduces additional complexity, it is regarded as a desirable characteristic. For instance, it can lead to smoother overall models compared to those based on completely disjoint partitions of the problem space. In regions where responsibilities overlap, the predictions of local models that cover the input are typically *mixed*, usually through the use of weighted averages. The combination of a matching function and a local model can be conceptualized as an *if-then rule* (or simply a *rule*), which is the common term within the broader field of rule-based machine learning (RBML). Due to historical reasons, the LCS literature commonly refers to a rule, along with its associated mixing weight (and potentially other bookkeeping parameters), as *a classifier*—hence, the name *Learning Classifier Systems.* In this chapter and the remainder of this thesis, we endeavour to minimize the use of the somewhat overloaded term *classifier*, preferring instead to employ the term *rule*, especially because the work presented within the subsequent chapters focusses on regression tasks rather than classification. We use the term *rule* to describe a tuple comprising a matching function, a local model, a mixing weight, and additional bookkeeping parameters, such as the rule's accuracy record. Similarly, consistent with existing LCS literature, we utilize the term *condition* to denote the genotypic representation of a matching function (i.e. a specific matching function corresponds to the phenotype of a particular condition).

The majority of the LCSs that currently enjoy popularity within the field have been developed in an ad-hoc manner [Dru08]. This means that these systems' mechanisms were not deduced from a set of central, formally specified assumptions concerning the learning task (e.g. the available data). Instead, especially in the early stages of LCS development, and likely owing to their role as proofs of concept for ideas related to artificial evolution and genetics, the methods to be employed were chosen first. They were then combined in such a way as to handle specific types of inputs [see, for example, Hol76]. Subsequent LCSs have built upon these early systems. For instance, XCS [Wil95] represents a simplification (and improvement) of ZCS, which, in turn, is a simplification of Holland's original framework [Wil94].

2.1.1 Learning Tasks

LCSs are a family of ML methods. Currently, with the exception of preprocessing and postprocessing steps, the fundamental components found within ML can be roughly categorized as *supervised learning* (SL), *unsupervised learning*, and *reinforcement learning* (sometimes referred to as *sequential decision making*) [Bis09]. Although there exist LCSs designed for unsupervised learning [cf. e.g. TBP07], their prominence has diminished in recent years. The majority of contemporary LCS research is dedicated to supervised learning (SL) and RL applications [PSN20a; PHW21a; HPW22].

The initial generations of LCSs were exclusively focused on RL tasks [Hol76; Wil95; Wil94], even though the term "reinforcement learning" was not in use during these early stages (the development of RL and early LCS frameworks occurred somewhat independently). However, it has become evident over time that these LCSs, while achieving competitive performance on SL tasks such as regression and classification (as discussed in the subsequent paragraph), do *not* perform well on complex RL tasks [Bar02; Ste+20]. In retrospect, this limitation led many researchers to analyse what LCS literature refers to as "single-step problems", which are essentially RL tasks with episode lengths of one. In many cases, these investigations effectively involve classification tasks, due to the finite set of reward levels. Additionally, many tasks occur within the context of deterministic environments, resulting in noise-free classification problems. Nevertheless, several existing LCSs, such as anticipatory classifier systems [e.g. BS02; Orh+20], where rules also model the behaviour of the environment, *are* indeed capable of solving more challenging RL tasks.

A larger portion of LCS research primarily revolves around SL, particularly *classification* and *regression* tasks. As of the time of this writing, the most widely used (and without doubt most extensively investigated) LCSs for these tasks are predominantly derived from XCS [PSN20a; PHW21a; HPW22; UB17]. An exception worth noting is *BioHEL* [cf. e.g. BK06; FKB13]. XCS is one of the systems originally designed for RL [Wil95], but has increasingly been applied exclusively to classification tasks [PSN20a; PHW21a; HPW22] rather than RL, although, there is still research regarding XCS and RL, [e.g. ST22]. Despite this shift, XCS retains several RL-specific mechanisms that are either not directly beneficial or even detrimental to classification tasks. Consequently, the *UCS* derivative emerged [BG03; UM15], explicitly tailored for supervised classification tasks.

In 2002, Wilson introduced *XCSF* [Wil02a], an XCS derivative designed for function approximation. XCSF has undergone continuous improvements, remaining competitive, at least in input spaces with low dimensionality [e.g. SMH18; Ste19]. Additionally, there exist other less recognized LCS-based regressors [e.g., Dru08]. It is worth noting that one application of regression involves approximating action-value functions in RL problems [SB18], effectively returning to the early goals of LCS research.

In conclusion, it can be seen that LCSs constitute a versatile framework suitable for a wide range of learning problems.

2.1.2 LCS Models

Most models built by LCSs are of a form similar to [PHH23; Hei+23a]:

$$\hat{f}_{\mathcal{M},\theta}(x) = \sum_{k=1}^{K} m(\psi_k; x)\, \gamma_k\, \hat{f}_k(\theta_k; x) \tag{2.1}$$

where

- x is the input for which a prediction is to be made,
- K is the overall number of rules in the model, which, together with the set of matching function parameters $\psi_k{}_{k=1}^{K}$, forms the *model structure* $\mathcal{M} = (K, \{\psi_k\}_{k=1}^{K})$,
- $m(\psi_k; x)$ is the output of the matching function of rule k defined by its parameters ψ_k for an arbitrary input x; typically, $m(\psi_k; x) = 1$ iff rule k is responsible for input x (i.e. if its local model models that input), and $m_{\psi_k;k}(x) = 0$ otherwise,
- γ_k is the mixing weight of rule k; mixing weights usually fulfil $\sum_{k=1}^{K} m_k(x)\gamma_k = 1$ and $0 \leq \gamma_k \leq 1$ (this can easily be achieved using normalization, which is standard for almost all LCSs),
- $\hat{f}_k(\theta_k; x)$ is the output of the *local model* (sometimes also referred to as a *submodel*)—a discriminative function $\hat{f}_k(\theta_k) : \mathcal{X} \rightarrow \mathcal{Y}$—of rule k (with parameters θ_k) for input x,
- and $\theta = \{\theta_k, \gamma_k\}_{k=1}^{K}$ forms the (whole) model's *parameters* out of the local models' individual parameters θ_k and their corresponding mixing weights γ_k.

Note that although the sum goes over all rules, by multiplying with $m(\psi_k; x)$ only the rules that match the considered input x contribute to the result (for non-matching rules, the respective summand is 0). Furthermore, instead of binary matching, that is, $m(\psi_k; x)$ being a function with image $\{0, 1\}$, *matching by degree* can be an option: $m(\psi_k; x)$ then has image $[0, 1]$ and may correspondingly take on any value between 0 and 1. However, since all of the more prominent LCSs use binary matching, we assume the same for the remainder of this chapter—unless we explicitly write otherwise.

We first presented this version of Equation (2.1) in [PHH23], basing it on [Hei+23a], which served as the foundation of this chapter.

2.1.3 Overall Algorithmic Structure of an LCS

To construct such an ensemble model, an LCS undertakes two primary tasks, each of which can be understood as having two subcomponents:

Model selection consists of

(I) determining the suitable number of rules, denoted as K, for the given task and

(II) allocating each rule a segment of the input space, essentially choosing the parameters $\psi_k{}_{k=1}^{K}$ for the matching function $m_{\psi_k;k}(\cdot)$ for a rule k.

It is crucial to emphasize, once more, that in LCS, the input space isn't divided into distinct partitions. Instead, an input may be modelled by multiple local models. The set of matching functions (combined with the parameter K) is also referred to as the model's *model structure*, $\mathcal{M} = (K, \{\psi_k\}_{k=1}^{K})$ in Equation (2.1).

Model fitting selects the best-performing local models and combines them optimally, given a fixed model structure. This usually involves

(III) fitting each local model $\hat{f}_k(\theta_k, \cdot)$ to the data its corresponding rule is—based on the already set matching function—responsible for, involving adjustments of its parameters θ_k. It is often advantageous to perform this independently for each local model, allowing for partial (rather than full) retraining when new training examples are introduced or the model structure is modified (typically, model selection is done iteratively and alternatingly with model fitting) [Dru08].

(IV) fine-tuning or fitting the mixing weights $\gamma_k{}_{k=1}^{K}$ (see the introduction to Section 2.1) to handle overlapping responsibilities among rules optimally.

These four subtasks are briefly discussed below, with a focus on the potential (and frequent) involvement of metaheuristics.

Determining the Number of Rules (K): The choice of the number of rules, represented by K, serves as a primary indicator of the overall model's complexity and, therefore, expressiveness. A higher number of rules increases expressiveness but comes at the cost of increased training effort and reduced interpretability. Additionally, when rules become especially numerous, overfitting occurs which causes worse generalization capabilities. To illustrate, assume the extreme case of less data points within the training data than rules in the constructed model. Conversely, too few rules may hinder the model's ability to sufficiently capture patterns in the data effectively, resulting in underfitting. Hence, selecting an appropriate number of rules (Subtask I) constitutes an optimization problem.

Allocating Matching Functions to Rules: Subtask II hinges on the previously chosen number of rules. It involves assigning responsibility to individual local models for modelling specific portions of the problem space. Notably, this is *the subtask for which all existing LCSs employ metaheuristics.* The quality of a solution in this subtask is highly dependent on the chosen number of rules. Assuming a model with few rules, then, for the problem space to be properly covered, we require at least one of those rules to be responsible for a larger part of the space than if there were more rules in the model. Therefore, Subtask I and II are often solved concurrently by using a single metaheuristic, or are solved by employing a combination of a metaheuristic for Subtask II and a simple heuristic for Subtask I. For instance, this latter approach has been adopted in XCS [Wil95] and its primary derivatives, where the optimization of the set of matching functions utilizes GAs [e.g. Wil95; BG03].

Fitting Individual Local Models to Data: The process of fitting each local model to the subset of training data, determined by the respective rule's matching function, (Subtask III) depends highly on the type of local model involved. For instance, linear local models may employ methods like least squares [Dru08]. Several well-known LCSs, particularly XCS derivatives, engage in online learning, where training data is processed instance-by-instance, enabling predictions during the learning process. Consequently, local models must support this type of learning. For example, in the original XCS, local models' parameters are updated incrementally using gradient-based approaches [Wil95]. Overall, fitting a local model also constitutes an optimization problem (e.g. minimizing the expected risk on the data matched by the corresponding rule) typically not addressed with metaheuristics.

Combining Local Model Predictions: Given fixed sets of matching functions and fitted local models, the remaining question (Subtask IV) is how to combine local model predictions into a single prediction (also called the mixing of predictions). As previously noted, an LCS's local models may overlap in their responsibilities, and for areas where that is the case, a sensible combination of the predictions of several local models needs to be made. Several popular LCSs adopt a straightforward approach, where each rule is assigned a quality metric (e.g. accuracy on the training data matched by the rule), and a weighted average is computed proportional to these metrics. This is how XCS handles mixing [e.g. Wil95]. Although some research has explored this task [e.g., Dru08], much of LCS research has largely neglected it and has not thoroughly investigated various methods [Dru08]. Generally, computing mixing weights for fixed matching functions and local models is another optimization problem, minimizing the expected risk for the whole model (rather than individual local models as in Subtask III), and while there are cases where provably optimal solutions exist, their computation can be costly. As such, inexpensive heuristics based on rule properties or metrics (like those used in XCS) may be more practical [Dru08]. To date, there haven't been any approaches employing metaheuristics for Subtask IV.

This concludes the high-level discussion of the four subtasks. Section 2.3 delves deeper into how Subtask I and II are approached using metaheuristics in existing LCSs, along with the challenges faced in doing so.

2.1.4 LCSs in Metaheuristics Literature

After presenting LCSs within the context of ML, we will now provide a brief overview of their presentation in existing literature on metaheuristics. Generally, it can be observed that, if mentioned at all, LCSs are typically discussed from a rather limited perspective, predominantly focusing on older systems. Additionally, recent work often excludes LCSs entirely. In the following, we will summarize the key aspects we have identified—note how these findings contrast with the more general description of these systems provided in this chapter.

LCSs are frequently categorized as a unique form of production system [Gol89; Wei09], methods for Genetics-based machine learning (GBML) [RBK12], evolutionary reinforcement learning (EvoRL) or Evolutionary Algorithms for reinforcement learning (EARL) [Wei09], or policy optimization [Luk13]. It is commonly assumed that LCSs employ a GA and construct sets of if-then rules. The fitness evaluation is typically based on individual rule statistics, such as accuracy or strength [BFM97; RBK12]. While it remains uncertain whether Evolutionary Algorithms (EAs) are the exclusive means for use in LCSs [Wei09; RBK12], the significance of the solution representation and the applied metaheuristic operators is evident. Specifically, the operators for generational replacement and mutation need to be tailored for use in LCSs [Gol89]. Nevertheless, GAs employed in LCSs are often likened to those applied in standard search and optimization problems [Gol89].

2.2 ML Systems Similar to LCSs

LCS models, as detailed in the previous section, exhibit certain resemblances to other established ML frameworks. We will now provide a brief informal comparison between LCSs and these frameworks, aiming to delineate the unique approach of LCSs. This distinction becomes particularly relevant given the somewhat flexible definition of what constitutes an LCS and the existence of numerous variations. To accomplish this, we will draw distinctions between LCSs and other techniques, including Decision Trees (DTs), Mixture of Experts (MoE) systems, Genetic Programming (GP), as well as ensemble learning methods like *bagging* and *boosting*.

2.2.1 Decision Trees

Decision trees represent one of the most well-known rule-based systems frequently applied to both regression and classification tasks. They can be generated automatically from data or manually crafted by domain experts. A typical DT partitions the input space into a set of disjoint regions, achieved by successively splitting the input space along its axes in an axis-parallel manner. The resulting hierarchical structure forms a tree, where each path from the root to a leaf node defines a distinct region. For each such region, a local model is fitted, typically stored in the corresponding leaf node. For classification, DTs, for example, often utilize a constant function corresponding to the majority class of the training data within that region. Deciding where to split the input space, i.e. in which dimension, at what value, and at what level of the tree, presents an optimization task with various solution approaches. Notable—very often applied—traditional algorithms for constructing DTs include CART [Bre+84] and C4.5 [Qui93]. In addition to these and related methods, metaheuristics have been explored for tree construction over the years: For instance, Boryczka and Kozak [BK10] employ Ant Colony Optimization (ACO) to construct trees similar to CART, Barros et al. [Bar+12] propose an EA-based hyperheuristic to create heuristics that subsequently build DTs, Podgorelec, Šprogar, and Pohorec [PŠP12] introduce a GA optimizing a population of DTs based on both accuracy and size, and Custode and Iacca [CI23] propose to use evolutionary methods to train interpretable DTs.

A direct correspondence exists between LCSs and DTs: Similar to a path from the DT root to a leaf, a rule in an LCS specifies a region and a local model. The only conceptual distinction lies in LCSs allowing regions to overlap, while classical DTs do not. Thus, it is possible to transform a DT into an LCS model straightforwardly without information loss. Conversely, the reverse transformation can be performed by expanding an LCS model with new regions for each overlap, and then further splitting these regions to achieve a proper hierarchy among them. Given that LCS models permit regions to be arbitrarily positioned, it can be anticipated that a relatively large number of regions (and consequently, tree nodes) may need to be added to obtain a proper DT from an LCS model. It is worth noting that fuzzy DT approaches [e.g., Jan98; BBM20] produce predictive models that closely resemble, if not match, the above definition of LCS models that employ matching by degree. However, the way these models are fitted fundamentally differs.

2.2.2 Mixture of Experts

Mixture of Experts (MoE) is a research direction that evolved independently of LCSs [e.g. Jac+91; JJ94; YWG12]. Nevertheless, these two approaches share significant similarities, with distinctions primarily arising from the MoE framework, as they usually

1. take a probabilistic view, resulting in prediction distributions instead of point estimates returned by LCSs models,
2. lack localized (using matching functions) submodels,
3. do not train submodels independently,
4. and do not feature constant mixing weights but weights depend on the input.

This differentiation implies that an MoE is generally more expressive than an LCS [1], primarily because localization (matching) in LCSs essentially determines whether the local model's output is multiplied by 0 or a constant mixing weight, while MoE allows for more diverse mixing possibilities. However, LCSs are inherently more interpretable since binary decisions, as in LCSs, are more comprehensible than decisions involving seemingly arbitrary values. It's important to note that MoE-like models with mixing weights independent of the input are often referred to as *unconditional mixture models* [Bis09]. Consequently, typical LCSs can be viewed as intermediate models, falling between unconditional mixtures and MoE models in terms of both interpretability and expressiveness. Furthermore, independent local model training in LCSs can result in slightly worse model performance in regions where local models overlap [Dru08]. Thus, an MoE is expected to outperform an equivalent LCS. However, independent local model training offers two significant advantages that may lead to improved performance given the same computational resources: More efficient model structure search (e.g. when changing a single matching function, only the corresponding rule needs to be refitted) and, for some local model forms, the absence of local optima during local model fitting [Dru08].

Presently, there are two MoE-inspired formulations of LCSs, one by Drugowitsch [Dru08] and another by Edakunni et al. [Eda+09].

Drugowitsch [Dru08] developed an LCS for regression and classification by incorporating matching into the standard MoE model. This results in a fully Bayesian probabilistic model fitted using variational Bayesian inference, independent of the employed model structure search methods (including the GAs and Markov Chain Monte Carlo (MCMC) methods explored by Drugowitsch). Unlike typical MoEs, local models are trained independently, leading to the aforementioned advantages and disadvantages. This model provides probability distributions over all possible outputs for any input, which distinguishes it from other LCSs; to our knowledge, no other LCS is currently able to do this.

Edakunni et al. [Eda+09] proposed a more specific approach, closely modelling UCS, an LCS for classification, using an MoE. Their system offers a simpler training routine compared to Drugowitsch's, as it handles only binary inputs (similar to the original UCS) and models all possible rules. However, training becomes infeasible in high-dimensional spaces. In a subsequent paper, Edakunni, Brown, and Kovacs [EBK11] extended their model with a GA for model selection and introduced iterative learning, features not yet supported by Drugowitsch's system.

2.2.3 Bagging and Boosting

Two of the most prominent ensemble learning techniques are bagging [Bre96] and boosting [FS96].

[1]At least as long as the LCS uses the typical binary matching functions. A matching-by-degree LCS can actually be more expressive than a comparable MoE.

Bagging [Bre96] involves the creation of multiple bootstrap datasets from the training data, followed by training one (weak) learner on each of these datasets. For making predictions on a specific input, the predictions from all available weak learners are either averaged (for regression) or combined through majority voting (for classification). This technique is known to reduce prediction errors, particularly when the weak learners exhibit instabilities, as is the case with DTs, neural networks (NNs), and RBML [Die00].

While there may be a superficial resemblance between the set of weak learners in bagging and the set of local models in LCSs, several significant differences exist. Other than in bagging, in LCSs,

- no bootstrapping is performed to assign data points to the local models; instead a good partition of the data set is *learned* by performing model selection.
- the learned data partitions not only impact the training process but also play a role in prediction.
- local model predictions are combined based on a quality measure rather than using a simple unweighted average.

Boosting, exemplified by the well-known AdaBoost algorithm [FS96], trains a sequence of weak learners (submodels) sequentially. After training each submodel, the error function used in training is adjusted based on the performance of the previously trained submodels. To make predictions, the submodels are combined through weighted averaging (for regression) or weighted majority voting (for classification).

LCSs differ from boosting in the following ways:

- LCSs have localized submodels, meaning they do not model all possible inputs[2].
- Submodels in LCSs are trained independently of each other.
- In LCSs, there is a direct optimization of the function that combines submodels. While boosting repeatedly modifies the error function, which bears some resemblance to the interaction of matching and mixing in LCSs, the matching and mixing processes in LCSs are more explicitly optimized.

In summary, while bagging, boosting, and LCSs share some similarities in terms of combining multiple submodels, each approach has unique characteristics and differences in terms of how submodels are trained, localized, and aggregated to make predictions.

[2]If matching by degree is used (though not the case for any of the prominent LCSs), they are softly localized, meaning that inputs are somewhat weighted, which aligns more with the concept of boosting.

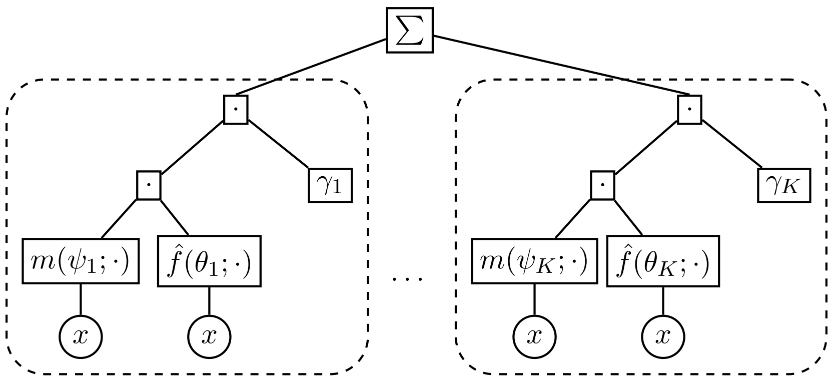


Figure 2.1: Direct transformation of an LCS model (cf. Equation (2.1)) to a GP tree. We summarized the nested summation of the local model outputs as a single large sum. Input nodes are circular. Another version of this figure first appeared in [Hei+23a].

2.2.4 Genetic Programming

Genetic programming (GP) [Koz93], as a well-known symbolic method applicable to various ML tasks, involves evolving syntax trees that represent functions or programs. In comparison to LCSs, GP approaches typically offer more degrees of freedom. While it's possible to directly transform an LCS model into a GP syntax tree (e.g. see Figure 2.1), the reverse transformation is not as straightforward. This is because GP trees, in general, do not need to conform to the typical structure of an LCS model. Unlike LCSs, GP does not involve the explicit generation of local models.

There are several approaches that incorporate GP techniques into LCSs. For instance, Iqbal, Browne, and Zhang [IBZ14a] utilize compact GP trees, referred to as *code fragments*, as conditions to enhance the exploration of model structure space in a more efficient manner by reusing building blocks from prior training on subproblems.

2.3 The Role of Metaheuristics in LCS

This section delves deeper into the role of metaheuristics in the learning process of LCSs. Section 2.1 has already established that metaheuristics are nearly always employed to tackle the task of model selection. This task encompasses Subtask II, involving the selection of matching functions, and Subtask I, focusing on determining an appropriate number of rules. Subtask I is sometimes approached using simpler heuristic methods rather than proper metaheuristics, whereas all LCSs use metaheuristics for Subtask II. In the following, we will first introduce various options for the general structure of the metaheuristic process and then proceed to analyse the representations, operators, and fitness functions utilized in this context.

2.3.1 Four Types of LCSs

In the early years of genetic-based/evolutionary ML, two distinct “schools” emerged, with which many of the investigated systems have since been associated: the Michigan and Pittsburgh approaches [DeJ88]. The classical definitions of these two terms are as follows [e.g. DeJ88; Fre02]:

Pittsburgh-style systems are methods using a population-based metaheuristic (e.g. a GA) at the level of complete solutions to the learning tasks. These systems maintain a *population of rule sets*, diversifying and intensifying their conditions. The operators of the metaheuristic work at the level of entire sets of conditions.

Michigan-style systems, on the other hand, consider a single solution (a single rule set) as a population on whose conditions a metaheuristic operates. In these systems, the operators of the employed metaheuristic operate on *individual conditions*.

However, this differentiation can be somewhat problematic. Firstly, there are many systems that don’t neatly fit into one of these two classes, as evidenced by the ”hybrid” systems listed in [UM09]. Secondly, the two terms primarily focus on population-based approaches. But what about an approach that uses a metaheuristic like Simulated Annealing for model structure search and gradient-based local optimization for model fitting? This metaheuristic doubtlessly works at the level of complete solutions to the problem, and might, in turn, be seen as a Pittsburgh-style system. On the other hand, only a single solution is considered at any given time and individual rules within that solution are modified by the metaheuristic, making it also resemble a Michigan-style system.

We suggest an alternative distinction for LCSs/RSL and similar systems based on two fundamental design decisions that significantly impact how a particular RBML algorithm operates:

Training data processing *Online* algorithms update their model after each data point whereas *batch* algorithms process the entire available data at once [Bis09, p. 143].

Model structure search LCSs exhibit substantial differences based on whether they consider a single model structure (and therefore a single solution to the learning task at hand) at a time or more than one. We propose distinguishing between *single-solution* and *multi-solution* LCSs.

These two dimensions give rise to four basic types of RBML algorithms: online single-solution, online multi-solution, batch single-solution, and batch multi-solution systems. These proposed terms also accommodate a continuum of approaches, such as mini-batch methods that fall between batch and online techniques, thereby eliminating the need for a “hybrid” category that encompasses everything not fitting into the existing categories. Moreover, these terms have relevance beyond the LCSs/RSL community, enhancing clarity and accessibility in the field.

Table 2.1 provides a concise overview of several prominent LCSs and their alignment with the two design decisions discussed. According to our definitions, systems previously classified as Michigan-style predominantly belong to the category of *online single-solution systems*, while those labelled Pittsburgh-style typically fall into the *batch multi-solution systems* category. XCS and its derivatives [Wil95; Wil02a; BG03] are online learners that employ a traditional Michigan-style metaheuristic, categorizing them as online single-solution methods. BioHEL [BK06] conducts batch learning but exclusively considers a single set of rules, placing it in the batch single-solution methods category. GAssist [Bac04; FKB13], a classic Pittsburgh-style system, operates with multiple sets of rules and conducts batch learning, categorizing it as a batch multi-solution method. It's noteworthy that, to our knowledge, there are currently no widely recognized online multi-solution LCSs.[3]

Table 2.1: Popular and well-known examples classified into the different types of LCS algorithms using our proposed system. A version of this table first appeared in [Hei+23a].

	online	batch
single-solution	XCS(F) [Wil95; Wil02a], UCS [BG03]	BioHEL [BK06]
multi-solution		GAssist [Bac04]

Each of the identified types of RBML algorithms presents its unique set of advantages and disadvantages. Batch algorithms are often simpler and more straightforward to analyse formally than online systems because model selection and model fitting are divided more easily [Dru08]. This is especially evident for existing population-based single-solution systems, such as XCS with its EA operating on individual rules, which are often difficult to analyse formally as the rules both compete for a place in the population but also cooperate to form a good global model/solution to the learning task.

Batch multi-solution systems can entail a relatively high computational cost. This is because, for each considered model structure, local models need to be fitted until convergence. This process can be far more computationally intensive than online single-solution systems which only have to fit the local models until convergence once and typically do so iteratively. Although, it should be noted that online single-solution systems perform fitting steps alternatingly with model selection steps, making the comparison slightly unfair as it is reasonable to assume that significantly more fitting steps are required until convergence than if the model structure were held fixed. An indicator for this might be found in the enormous amount of training examples provided regularly in different studies to many state-of-the-art Michigan-style LCSs [e.g. Nak+17; Ste19; LBX20]. This leads us to assume that the comparison may actually not be as unfavourable for batch multi-solution systems as initially expected but a thorough study investigating this hypothesis has to our knowledge not yet been performed.

[3]An argument could be made for classifying a few reinforcement learning systems, such as PPL by Bishop, Gallagher, and Browne [BGB22] which does feature multiple solutions at a time and is described as "Pittsburgh-style", as an online multi-solution LCS. However, this is not (yet) a widely recognized system within the community or without, which is why it is not included in Table 2.1.

2.3.2 Metaheuristic Solution Representation

Metaheuristics optimize the set of matching functions and, at that, operate on the set of conditions (useful representations of matching functions, cf. Section 2.1).

Early LCSs were designed mostly for binary input domains and these systems are still among the most-used and researched ones [UM09]. Matching functions for these domains are typically represented by ternary strings, that is, binary strings extended by an additional symbol #, the so-called *wild card*, that represents any of the other two options and thus enables generalization. An example for the representation of a matching function $m : \{0, 1\}^5 \to \{0, 1\}$ is

$$(1, 1, 0, 1, \#). \tag{2.2}$$

It assigns 1 to (matches) the inputs $(1, 1, 0, 1, 0)$ and $(1, 1, 0, 1, 1)$ and 0 to any other inputs, [e.g. Wil95; UB17].

For real-valued or mixed integer problem domains, many different representations have been proposed. Among the simplest that are commonly applied are hyperrectangular [Wil00] and hyperellipsoid [BLW08] conditions. An example for a hyperrectangular condition for a matching function $m : \mathbb{R}^3 \to \{0, 1\}$ is the 3-dimensional interval $[l, u)$ (with $l, u \in \mathbb{R}^3$) which can be seen as a tuple

$$(l_1, u_1, l_2, u_2, l_3, u_3). \tag{2.3}$$

It matches all $x = (x_1, x_2, x_3) \in \mathbb{R}^3$ that fulfil[4] $l \leq x < u$:

$$m(x) = \begin{cases} 1, & l \leq x < u \\ 0, & \text{otherwise} \end{cases} \tag{2.4}$$

Aside from the mentioned ones, several more complex function families have been proposed, e.g. neural networks [e.g. BH03] and GP-like code fragment graphs [IBZ14a] (also referred to in Section 2.2.4).

In general, any representation is possible, including composites or combinations of other representations, as long as appropriate operators can be defined, cf. [e.g. UB17]. It has to be kept in mind, however, that more complex representations often lead to more complex operators being required as well as that both the size and the topology of the search space is directly influenced by the representation (e.g. there are functions $[0, 1]^5 \to \{0, 1\}$ that cannot be represented by the above-introduced representation of ternary strings of length 5). Furthermore, too simplified or restricted encodings of the feature space can result in parts of the matching functions space being inaccessible. Variable length representations may alleviate some of these problems but can render the operator design more difficult.

[4]This is equivalent to $(l_1 \leq x_1 < u_1) \wedge (l_2 \leq x_2 < u_2) \wedge (l_3 \leq x_3 < u_3)$.

2.3.3 Metaheuristic Operators

Metaheuristic operators need to not only be compatible with the chosen form of conditions but also perform well in optimizing the corresponding sets of matching functions. This can be a challenging task for ML practitioners without a strong metaheuristic background, in turn causing the same operators to be used repeatedly, regardless of whether they may actually be suboptimal.

An important aspect in LCSs is *when* the metaheuristic is invoked and whether it operates on the entire set of conditions or on a subset. For instance, in XCS [Wil95], for each input provided to the system, the GA applies its operators only to the conditions of a subset of rules; namely to the ones that matched the input seen last and of those only to the ones that also proposed the action taken. Due to the dependence on the input introduced by only considering matching and used rules, the GA may operate on a different subset in the very next iteration.

The initial set of conditions is usually created at random, possibly slightly directed by requiring the corresponding matching functions to match certain inputs (matching functions that do not match any of the training data may not be that useful since their merit cannot be estimated properly). [E.g. Wil95; UB17]

For the generation of new individuals from existing ones, existing LCSs use both recombination as well as mutation operators [UM09; UB17]. Recombination operators in single-solution systems may exchange condition attributes (e.g. hyperrectangular boundaries) whereas, in multi-solution systems, they probably should also include an option to recombine sets of conditions in a meaningful way (e.g. exchanging entire conditions between sets of conditions).

Optimizing matching functions poses a difficult problem if the training data is sparse, or, more generally, if there are sparsely sampled parts of the input space: On the one hand, changing a condition only leads to a detectable difference in accuracy-based fitness if that change alters the subset of the training data that is matched by the corresponding matching function. On the other hand, these differences, if occurring, can be very large (e.g. if a rule now matches three training examples while, before, it only had matched two). This means that, depending on the training data and initialization, the operators may have a *low locality* if fitness computation only takes into account accuracy statistics on the training data. As a result, areas between training data points may not be covered by solution candidates because there is no fitness signal when exploring having some rules match them. Choosing a combination of suitable operators and a fitness measure that present a consistent answer to this issue is an open problem; the common workaround is to simply rely on comparably large amounts of training data.

While multi-solution systems can explore different solution sizes rather naturally, single-solution systems require explicit mechanisms to control and optimize condition/rule set size. A popular option for population-based single-solutions (used, e.g., in XCS [Wil95]) is a simple heuristic: There is a maximum number of rules (a hyperparameter) that, when violated by rule generation mechanisms, gets enforced by deleting rules based on roulette wheel or tournament selection. Aside from that, the *numerosity* mechanism is typically employed (also used, e.g., in XCS [Wil95]): If there is a well-performing rule r_1 that is responsible for more inputs than another rule r_2 but the inputs matched by r_2 are already covered by r_1, then r_2 may be replaced by a copy of r_1 (typically, no real copies are used but instead a counter associated with each rule in the set). The set of conditions thus contains one less unique condition.

In existing online single-solution systems, metaheuristic operators gradually change parts of conditions of the rule set in a steady-state fashion. At that, the central challenges are maintaining a healthy diversity and identifying the required niches—and keeping them in the rule set [UB17]. Selection is usually based on either roulette wheel or tournament selection from either the whole set of rules (especially in earlier LCSs [UM09]) or subsets (e.g. in XCS [Wil95]), the latter promoting niching. Niching is also promoted when performing rule discovery for examples that are not yet matched by any rule in the set. The mutation operator modifies a single condition; how this occurs primarily depends on the specific representation but is typically stochastic, balancing generalization and specialization pressures. Commonly used operators include bitflip [Wil95] and Gaussian mutation [Dru08]. Recombination also works at the condition level and is usually a single-point, two-point or uniform crossover with encoding-dependent crossover points. The replacement of rules usually employs elitism operators. [E.g. UB17]

Batch multi-solution systems are more similar to other well known optimization approaches [UB17]. Most existing systems of this category are generational rather than steady-state. Unlike in existing online single-solution systems, parents are condition sets and not individual conditions and are selected from a population of condition sets, typically using roulette wheel or (primarily in later systems) tournament selection. The mutation operator mutates at two levels: at the level of condition sets by adding and removing rules, as well as at the level of individual conditions using a method appropriate for the rule representation, for example, Gaussian mutations of all bounds when using an interval-based condition. The recombination operator mostly exchanges rules between rule sets but can also be extended to additionally exchange parts of individual conditions. [E.g. Bac04]

2.3.4 Typical Fitness Functions

As already noted in Section 2.1.3, the problem that an LCS's metaheuristic tries to solve is model structure selection, that is, choosing the size of the rule set (Subtask I) and proper matching functions (Subtask II). The goal of this optimization task is to enable the model to be optimal after it has been fitted. At that, optimality of the model, and with it, optimality of the model selection, is typically defined slightly handwavy based on the fitness measure used for model selection. That fitness measure commonly weighs a high accuracy of overall system predictions against a low model structure complexity (i. e. number of rules) [e.g. Wil95; Bac04; BG07]. However, there are also lesser-known, but more principled approaches to what an LCS is meant to learn that we cannot expand on here for the sake of brevity; for example, the one by Drugowitsch [Dru08] based on Bayesian model selection.

The need for high accuracy and a low number of rules induces a multi-objective optimization problem with conflicting goals (the highest accuracy can consistently be achieved with a very high number of rules, e.g., one rule per training example). However, the utilized fitness functions are not always modelled explicitly multi-objectively but often use a (weighted) sum of the objectives—or even focus on only one of them. The exact fitness computation within the system strongly influences the metaheuristic to be used; therefore, we will shortly describe the different options and their implications.

Online single-solution systems usually incorporate niching and thus require mechanisms for fitness sharing. In earlier LCSs for RL, there was an *explicit fitness sharing* mechanism that split the reward among all rules in the same, activated niche [e.g. Hol76; Wil94]. The more generally applicable technique is *implicit fitness sharing*, which is based on computing the fitness relative to the rules in the same niche and applying metaheuristic operators only within that niche [e.g. Wil95]. The fitness functions are usually to be maximized and are often based on either rule strength [e.g. Wil94] or rule accuracy [e.g. Wil95]. Strength-based fitness is often used in earlier LCSs built for RL settings; it builds on the sum of RL rewards after applying the rule. Accuracy-based fitness, on the other hand, is based on the frequency of the rule's correct predictions; and, due to its increased stability, much more common these days.

In batch multi-solution systems, purely accuracy-based fitness functions can result in bloating [Dru08], that is, a significant amount of additional rules being included in the rule set that do not improve the solution. To resolve this issue, multi-objective fitness functions with the second objective being the reduction of rule set size are used. They are often modelled as weighted sums of the individual objectives and thus still treatable like a single-objective problem (scalarization). One example for this is the utilization of the minimum description length (MDL) principle for the fitness function [BG07] in BioHEL [BK06] and GAssist [Bac04]. MDL is also a common strategy in optimization for feature selection problems.

2.4 Metaheuristic-centric RBML Approaches Similar to LCSs

We next discuss the similarity of LCSs to other metaheuristics-focussed RBML systems, that is, other systems utilizing GAs, ACO, other metaheuristics or hybrids, and Artificial Immune Systems (AISs). These were developed independently of LCSs, but are often based on the same ideas (e.g. [Hol76]) or on the work of [Fre02], which summarizes genetic approaches for data mining. Furthermore, they are used to construct if-then rules, mostly applied to classification tasks, and often divided into Michigan and Pittsburgh approaches [Fre02]. The main difference between these metaheuristics-focussed RBML systems and LCSs is that most of them do not utilize any additional bookkeeping parameters.

The list of learning systems in this section is not exhaustive, but aims at providing a broader view with some short examples. It is important to note that we restrict ourselves to systems

- that are more or less close to the definition of an LCS given in Section 2.1 but whose authors do not relate them to existing LCS research (or do not explicitly call them LCSs)

- that somewhat lie at the very border of the field of LCS research and may thus not be known well.

While LCSs are commonly associated with evolutionary (or more specifically genetic) algorithms, any metaheuristic can be used [UB17; RBK12]. This makes the comparison of these other metaheuristics-focussed RBML systems even more relevant, as algorithms from both fields can profit from the respective other field's research.

2.4.1 Approaches based on Evolutionary Algorithms

The degree of resemblance to LCSs is most obvious for approaches which also utilize EAs and of which a non-exhaustive overview is given in this section. Most of these approaches, for example the general description of GAs for data mining and rule discovery by [Fre02], have been proposed with—if at all—only little differentiation or comparison to LCSs.

Approaches utilizing GAs for the discovery and optimization of classification rules are for example described in [van+97; ALF99; AS12; MSC19; Mir+20]. These are mostly subsumed under the definition of Michigan-style systems, though many of them are actually batch single-solution systems. They differ from LCSs in terms of the operators used in the GA, the fitness computation, or the use of additional strategies. For example, [Mir+20] extended their RBML system to perform multi-label classification, while [ALF99] utilize a parallel GA in their approach. A classification rule mining system using a multi-objective GA (MOGA) is presented by [Gup+17]. There are also approaches using evolutionary techniques such as co-evolution, which is applied to sets of examples, the rules being induced at the end [JLZ06]. This presents an inverse order of the process compared to the traditional LCS approach. Furthermore, other EAs can be used for rule discovery, for example a quantum-inspired differential evolution algorithm [SYZ10].

While there often is no direct relation provided between other evolutionary RBML systems and LCSs, at least some summaries describe a few of the different approaches [e.g. Fre03] or provide experimental comparisons [TF10]. An exception is the combination of DTs and a GA by [SSC12], which includes two rule inducing phases (a decision tree produces rules, the GA refines these rules) and which is simultaneously described as a Michigan-style LCS with three phases.

2.4.2 Approaches based on the Ant System

Another branch of approaches for (classification) rule discovery is based on the ant system, or Ant Colony Optimization (ACO), with the Ant-Miner as the most prominent representative. Their similarity to LCSs is strongly dependent on the variant of LCS and on how much the utilized metaheuristic is seen as a defining component. For example, the Ant-Miner [PLF02] is a batch single-solution system with an overall concept similar to BioHEL [BK06]. Its pheromone table is similar to the attribute tracking concept in ExSTraCS [UB17]. Furthermore, it uses the same rule representation strategies as LCSs in general, with the exception that continuous variables are more often discretized in the Ant-Miner. Nevertheless, the ACO algorithm [DS04] is quite dissimilar from GAs and the resulting RBML systems can exhibit further differences.

The Ant-Miner develops if-then rules whose condition consists of a concatenation of terms (i. e. *attribute*, *operator*, and *value*). Rules are constructed by probabilistically adding terms to an empty rule under utilization of a problem-dependent heuristic function and the ACO-typical pheromone table. Afterwards, the rule is pruned and the pheromone table is updated and the next rule is constructed. This process is repeated until the maximum population size (number of ants) is reached or the same rule has been constructed more than once. Then, the best rule is selected and the data points it matches and correctly classifies are removed from the training set. The overall algorithm is repeated until enough cases in the training set are covered by the aggregated rule set. [PLF02]

There are several extensions and variants for the Ant-Miner [BK09; AS17], for example, different pheromone update functions or heuristic functions and adaptations to cope with continuous data [LAM03; OFJ08]. Furthermore, there also exists a regression rule miner based on Ant-Miner [BO15] and a batch multi-solution Ant-Miner variant [OFJ13]. Also, other ACO-based classifier systems have been developed simultaneously to Ant-Miner [e.g. SJK04].

2.4.3 Approaches based on other Metaheuristics or Hybrids

Next to GAs and ACO, there are many more metaheuristics and hybrid algorithms that can be utilized in RBML, especially for classification rule mining [DJ19]. They share roughly the same basic view on rules as well as a classification into Michigan- and Pittsburgh-style approaches, although the term Michigan-style often subsumes both online and batch single-solution systems. Again, their similarity to LCSs depends strongly on the respective variants and the underlying definitions but a direct integration into existing LCS research is often not provided. While this section can not present these approaches exhaustively, it showcases further insights on how metaheuristics can be applied to RBML.

Particle Swarm Optimizations (PSOs), for example, has been used for classification rule discovery [SSN03; SSN04] as well as for a regression rule miner [MM12]. Furthermore, the Artificial Chemical Reaction Optimization Algorithm (ACROA) was used to optimize classification rules as well [Ala12]. While these approaches all use population-based metaheuristics, it is not impossible (or infeasible) to use single-solution based optimizers, as was demonstrated in [Moh+08] where Simulated Annealing (SA) determines fuzzy rules for classification. This SA variant was also extensively compared to the LCSs GAssist and XCSTS.

Hybrid approaches, that is, algorithms combining two different metaheuristics to combine their benefits, are common to classification rule discovery as well. They are, again, often presented as Michigan-style systems; however, many of them perform batch single-solution learning. There exist, for example, hybrids of PSO and ACO [HF05; HF08], SA and Tabu Search (TS) [CJM12], and ACO and SA [SK11]. Some of these hybrids explicitly divide their rule discovery process into two phases; this is the case, for example, for the HColonies algorithm, a combination of

AntMiner+ and Artificial Bee Colony (ABC) optimization [AM14b]. Additionally, batch multi-solution hybrids are possible, as presented by an Ant-Miner-SA combination [ND20].

Another type of hybrid systems for classification rule mining combines not only two meta-heuristics, but utilizes them in what the authors call a Michigan-style phase and a subsequent Pittsburgh-style phase. In our new classification system, these would simply fall into the batch multi-solution category. For example, [Tan+03] use a combination of a GA and GP—the exact choice is dependent on the types of attributes—in the Michigan phase to generate a pool of rules and then perform a Pittsburgh-style optimization with a GA in the second phase to evolve the best rule set from the pool. Similarly, [AM14a] use a hybrid of ACO and a GA. AntMiner+ is used in the first phase to construct several solution based on different subsets of the training data, while the GA uses these models as an initial population for optimization. This approach utilizes the smart crossover developed for Pittsburgh-style LCSs, which is an indication for at least some overlap between the two research communities.

2.4.4 Artificial Immune Systems

Artificial immune systems (AISs) are another class of algorithms inspired by biological processes and suitable for ML and optimization tasks. AISs are differentiated by the general strategies they employ, that is, clonal selection theory, immune network theory, negative selection and danger theory. [RBK12; HT08; Tim+08]

First of all, the similarity of AIS algorithms and LCSs depends strongly on the strategy. Clonal selection– and negative selection–based AISs are more similar to evolutionary RBML systems than immune network or danger theory AISs. Furthermore, both AISs and LCSs entail many variants, depending on the learning task and implementation choices. At that, for example, solution encoding and operator choice further increases or decreases the similarity between these approaches. Finally, note that AISs research often acknowledges the similarities and differences to LCSs. [FPP86; Gar05]

3 Assessing Explainability Requirements

This chapter will focus on interpretability, comprehensibility, and explainability [1] of machine learning (ML) models with a special focus on rule-based ML (RBML) systems, especially rule set learners (RSLs) which are traditionally called Learning Classifier Systems (LCSs). Section 3.1 will provide more general insights into explainability (without copying the large bodies of existing literature directly). I advise the reader to follow Barredo Arrieta et al. [Bar+20]'s seminal paper for a very in-depth view on the topic. Section 3.1 will also discuss explainability in RBML and specifically the aforementioned LCSs. The remaining Sections 3.2 to 3.5 will present a general template to assess LCS model requirements with regards to explainability and will exercise through that template in a real-world case study [Hei+23b; HNH21]. The resulting questionnaire, the demonstration of its first successful application, and the consequences that arise from that form the second major contribution of this thesis, *C2* (cf. Section 1.1). The implications on model design gained from that case study (detailed in Section 3.5.6) will then serve as a foundation for design choices made in the algorithmic designs presented in the following chapters.

3.1 Explainability for Rule-based Machine Learning

Explainability of ML models can—at least to some degree—be achieved [CPC19; Bar+20] by

- using *transparent models*, allowing interpretation of decisions and comprehension of the model based on the model structure itself, or
- applying *post-hoc methods*, utilising visualisation, transformation of models into intrinsically transparent models and similar techniques on models that are not by themselves transparent.

Which of these two ways is suitable for the application at hand, however, depends on the part of the model and prediction process that should be explained and the required properties of the explanations [CPC19]. Furthermore, it is often necessary to consider the trade-off between performance and explainability, which is, however, also application-dependent [Her+22].

[1]To enable an easier access for the reader, these terms and further similar terms will, throughout the entirety of this thesis, be summarized under the umbrella term of *explainability*, unless specifically needed otherwise, due to their inconsistent use in literature and strong similarities in many definitions where the minute differences are not of consequence for making specific statements at the abstraction level of this work. This seems especially reasonable as the wider field converges on the term of explainable AI (XAI).

Generally, linear or logistic regression, decision trees, K-nearest neighbours, rule-based learners, generalized additive models and Bayesian models are considered intrinsically transparent [Bar+20]. There are many more post-hoc methods, which differ, e.g., in their targets for explanation, their applicability or the type of explanations [VL21; Zho+21; Kau+23]. In addition, the explainability in terms of quality of explanations of these approaches is specific for an individual application and the target audience of the explanations. This means, explainability usually has to be evaluated on a case-by-case basis, with the evaluation being application-grounded, human-grounded or functionally-grounded [DK18]. For application- and human-grounded evaluation, the specific use case and human feedback on the quality of explanations is of special importance. This also relates to determining beforehand what a good explanation has to entail by performing user studies [Hei+23b]. Functionally-grounded evaluation of explanations usually uses some kind of metric, e.g. the information transfer rate [SB19].

As rule-based learning systems, Rule Set Learners (RSLs) / Learning Classifier Systems (LCSs) generally fall into the domain of transparent models and are regarded as excellent for interpretability due to their relation to human behaviour [Bar+20]. However, several factors can limit the degree to which humans can easily comprehend the model and follow its decision making process, therefore reducing the practical explainability of the models. For LCSs, the number of rules and their specific formulation are the prime factors impeding their inherent interpretability. The high impact of the number of rules in a model lead to techniques to alleviate this problem, for example the promotion of smaller individuals in batch multi-solution (Pittsburgh-style) LCSs through adjustments of the fitness function [BG07] or subsumption and compaction methods in online single-solution (Michigan-style) LCSs [LBX19; LBX21a; TMU13]. On the individual rule level, explainability is primarily hindered by the form of conditions and the local models making the prediction. Conditions of rules in complex feature spaces are harder to understand than those that operate directly on the data, e.g. higher level features aggregating multiple sensor readings versus the readings themselves. Additionally, conditions can be formulated using non-linear functions rather than readable decision boundaries [e.g. BO02]. Local models of rules in the form of complex black-box models, such as neural networks [e.g. LL06], are also harder to understand than linear or constant models, even if these local black box models are usually much smaller than a model of the same class that encompasses the complete problem space would need to be. An improved understanding of singular rules can be pursued by promoting simplicity during training through a suitable fitness function, and by applying post-hoc analyses typical for the respective models, e.g. feature importance estimations in neural networks. There are also different visualization techniques to attain or improve post-hoc explainability for LCS models: Feature Importance Maps and variations thereof are used for better understanding interactions of rules when the knowledge in the resulting model depends on the cooperative information of individual rules [LBX21b]. Visualization for distinguishing predictive from non-predictive attributes is done by utilizing heat-maps [UGM12]. These can also show patterns resulting from attribute interaction.

For rule-based learners and decision trees, there are several approaches trying to determine if one model is more interpretable than another. Lakkaraju, Bach, and Leskovec [LBL16] state that decision sets are more interpretable than decision lists, which again are more interpretable than decision trees. Their comparison is based on different metrics, i.e. the size of the rule set, the length of individual rules, the coverage of data points and the overlap of rules, and, additionally, the accuracy of a rule based on its correct or incorrect coverage of data points. Margot and Luta [ML21] also provide metrics for evaluating the interpretability of rule-based models, namely a predictivity score, a stability score and a simplicity score. Related to these approaches is explainability for Random Forests, which are not considered transparent. Explainability is usually provided post-hoc by simplification, for example by extracting rules with methods like inTrees [Den18] or by constructing decision paths with, for example, CHIRPS [HGA20]. The degree of explainability can then again be determined utilizing metrics such as the rule length, coverage, precision, or time to respond [Myl+22]. Virgolin et al. [Vir+21] proposed to use a user-informed explainability metric during a multi-objective evolutionary model selection process, where the score of a model was determined based on a model trained concurrently to the evolutionary search by using active learning.

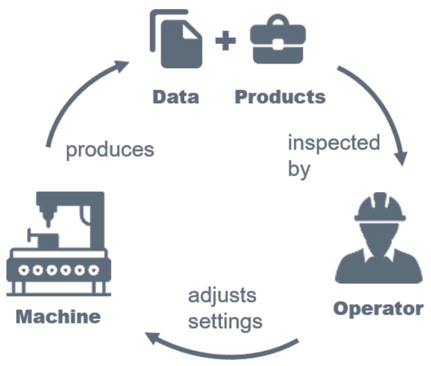
3.2 An Example Application for Audience-appropriate Explainability

As discussed in the previous section, explainability always has to keep the target audience in mind if we expect it to actually provide value rather than being just a buzzword. One of the critical aspects here is that explainability might not be achievable for every learning task and associated target group combination and many of the potential target groups might not even care to understand. However, there are many tasks where explainability can be critical and where groups actually want (or need) to understand why certain predictions are made or certain actions are undertaken. In this section, a scenario in which explainability is important will be introduced, while the following sections will discuss why LCSs/RSLs are an appropriate method to approach this scenario (cf. Section 3.3), will present a template on how to determine which aspects of a model would need to be explained (or whether explainability is relevant at all), with a special focus on models constructed by LCSs (cf. Section 3.4), and, finally, will showcase the application of this template in a real-world industrial scenario in line with the current section and a user study of relevant stakeholder groups (cf. Section 3.5). Importantly, Section 3.5.6 will layout implications for what a model should look like to be appropriate for the specific audiences and the use case. Based on these implications, the later chapters of this thesis have made assumptions about algorithmic design of the largest contribution (*C3*) of this work, the Supervised Rule-based Learning System (SupRB). The remainder of this chapter has previously been published in [Hei+23b], which itself is a substantial extension of [HNH21].

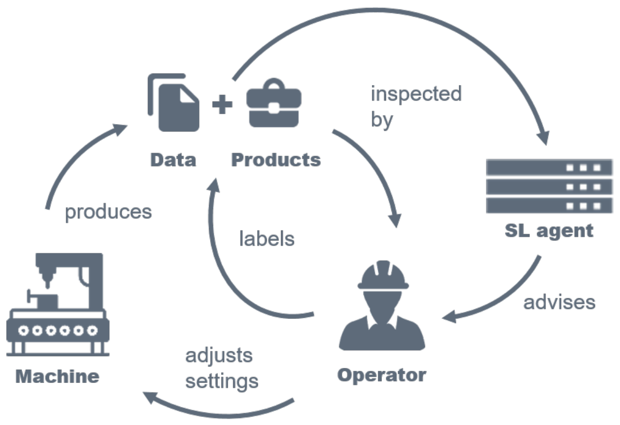
Increasing automation of manufacturing creates a continuous interest in properties commonly associated with lifelike[2] [Ste+21b] or organic computing systems [MSU11; MT17], such as self-adaptation or self-optimisation, within the production industry [Per+16]. These properties are often achieved using data driven and learning methods [Zha+17; Lug+19; Sch+20], as with increasing digitalisation and internet of things (IoT) efforts, where more and more devices

[2]https://lifelikecs.organic-computing.de/

are interconnected and partake in complex problem solutions, data can be collected in large amounts. In modern factories, products are usually inspected by the machines' operators or specialized quality assurance personnel to assess their quality, cf. Figure 3.1a. For the sake of simplicity, we subsume both roles under the term 'operator'. Recent advances in automated inspection often integrate computer vision-based approaches [Mar+17; Mar+23]. However, these can be of limited use when quality is not assessable from the surface, e.g. structural or chemical properties that involve laboratory testing. Thus, these systems currently can only partially automate inspection while the conclusions with regards to machine reconfiguration are still reached manually in many cases. This requires a large amount of operator knowledge and experience to achieve optimal or even satisfactory results. In settings with heterogeneous machines and few operators, the strain on operator experience is further increased and production can be seriously threatened by a loss of qualified personnel, e.g. through retirement.



(a) Operator-in-the-loop in modern manufacturing [Hei+23b].



(b) Assisted production using an agent trained with supervised learning (SL) during operation [Hei+23b].

Figure 3.1: A possible transition of modern operator-in-the-loop manufacturing to the usage of a supervised learning–based agent.

To reduce reliance on specific knowledge of operators and improve the self-adapting and self-optimizing systems, the operator can be assisted by decision support systems. These can easily incorporate large amounts of information simultaneously and are less biased to well known settings, especially compared with operators that only have limited understanding of or experience with the machines. Such decision support systems utilize learning from past experience and ongoing human expert feedback. Combining human operators and supervised learning (SL) agents that collaboratively adjust machines (or lines thereof) that manufacture products expands the socio-technical system with a collaborative decision making dimension, cf. Figure 3.1b.

Typical shopfloor environments will feature many workers operating on many machines, but not necessarily in a one to one array, e.g. multiple workers might be needed to operate a single machine while multiple other machines can be operated by a single worker due to automation. Additionally, to utilize the available data most efficiently, not every machine should need its own model, but models should generalise over multiple machines of the same or similar type. For production lines where multiple models would participate, the parametrization choices of preceding machines would need to be accounted for by subsequent models, e.g. through the help of models of higher abstraction. In this environment, each individual model takes input from and advises multiple operators, while each individual operator might interact with different models throughout a shift.

An integral element for implementing these systems is that operators are able to trust decisions made by their recommendation agents. This requires the system to be self-explaining in both adequate form and abstraction level. However, when form and abstraction level can be considered adequate is highly use case–specific and may also be user-specific (cf. question 2 of Section 3.4) [Her+22; BP21]. It involves an explanation regarding the basis of the recommendation, e.g. what input parameters led to this output, as well as an assessment of the quality of the decision, e.g. what is the expected error in quality when executing the recommended parametrization. In this chapter, we posit that Learning Classifier Systems are well-suited to be used within the proposed SL agent by reviewing different explainability techniques in light of this setting (cf. Section 3.3). We then introduce a template of research questions that need to be addressed to successfully apply LCSs (or other rule-based systems) in this context within Section 3.4. In Section 3.5, we demonstrate the successful usage of those questions in a case study where we utilize them in a sequence of interviews with stakeholders from a producing company, the REHAU SE.

3.3 LCSs in Industrial Decision Support Systems

Many different LCSs have been proposed over the years and while originally envisioned as a powerful reinforcement learner, they have been extended for all learning paradigms [UM09]. In our view, their structure of overlapping rules especially motivates their application within a decision support system as this structure strikes a good balances between models' explainability and their performance which is not offered by any other common learning paradigm (cf. Sections 2.1.2 and 4.3). However, we acknowledge that as there is a plethora of possibilities to train such a model, choosing the 'right' LCS for an actual implementation needs to be done use case–specific, as some LCSs will yield better performing models than others and their transparency varies (for more details on transparency of models and related concepts cf. [Bar+20; Bac+22]).

For the following, we consider the application as a decision support system that proposes settings to an operator and informs them of the reasoning behind this choice to be an SL task. This can be solved with either online or offline learning as long as the model used to make recommendations provides a compacted version of itself for inference and subsequently serving explanations. The LCS learns from experiences including sensor readings, product information, used machine settings and resulting quality measures, all of which will be a mixture of real and categorical values. When tasked with assisting an operator, the SL agent uses sensor readings and product information to propose machine settings and predict the expected quality.

Besides the previously introduced explainability techniques, LCSs also easily allow us to provide operators with all examples from our training data that formed the local model (as we know which examples were matched by the rule's condition). This can help further the trust that the model's predictions are actually based on existing expertise. Going beyond traditional explaining by example [Bar+20], each example that influenced an individual rule's weights could hypothetically be listed, whereas in black-box models usually the entire sample influences every weight.

In online single-solution (Michigan-style) LCSs, each individual rule gets ascribed a quality measure (or multiple thereof in XCS(F)). This (or in case of multiple measures, at least one of them) represents the rule's fitness and is used to guide an evolutionary process. Moreover, we can utilize these measures to provide our operator with additional information on how exact and therefore useful a recommendation is. Rules with a low prediction quality and thus a high expected error might provide poor machine settings, while other rules in the model might actually provide very useful settings. This disparity in different parts of the feature space can also allow insights into where new sampling should take place [SMH17] and allows to differentiate the model further. Even if—viewed globally—the model is less than optimal, it can still be used within the SL agent and aid operators on problem instances where it is well fitted.

3.4 A Template to Assess Explainability Requirements and LCS Model Design

Following this examination of the applicability of LCSs as decision support systems for the parametrization of industrial machinery in a complex socio-technical environment, we want to raise several questions that—in our view—need to be answered on a case-by-case basis. We assume that some parallels will exist between applications, it seems, however, unlikely that general answers will hold for all or even the majority of cases. Note that we broaden the scope from our operators that interact directly or indirectly with the machine to all stakeholders that have a vested interest in the operation of the shopfloor, both digital and analogue. Thus, this could also include regulatory bodies, safety officers, engineering, management, customers, data scientists and others.

We hypothesize that the seven following questions allow those responsible for model development and deployment to gain valuable insights into what is actually requested by those affected by such a model. Ideally, the answers are so detailed that exact requirements could be made on the design of an LCS model which in turn allows the design of suitable training algorithms. But even if the answers are not detailed or specific enough to make those decisions directly, they might serve to decide which algorithms or model designs might be suitable or whether LCS are even the right choice. In cases where explainability is not deemed important, practitioners can default back to deep learning. In other cases, decision trees or simple linear models might be more appropriate. The questions should also serve to determine whether these different models are more fitting for the use case and can also give insights into their design requirements. Furthermore, we assume that those questions allow stakeholders to engage on what they have to expect from such a model and may later on be used to justify and explain certain limitations and trade-offs that have to be made without going into too much algorithmic detail. We intend for the questions to serve as a template for other practitioners in designing their own studies, if needed by adding further questions or detailing and adjusting some of the existing ones. However, based on our experience (among others those detailed in Section 3.5), we assume they are fitting for many situations.

Q1: To what extent does a stakeholder request explanations? This can have numerous dimensions, such as depth, frequency or diversity of explanations. Someone that operates the machine directly might prefer examples of past experiences while quality assurance personnel might prefer visualizations or vice versa. In this question we assume that stakeholders may seek explanations that go beyond regulatory requirements, although a potential answer may be that they are not interested in further/deeper explanations. This raises another aspect: How important is explainability deemed if prediction quality potentially suffers?

Q2: What are the differences within a type of stakeholder? Tying directly into the previous question, we assume that the diverse stakeholders of a given type will answer questions regarding explainability differently. Individual stakeholders may also hold different understandings of the machine itself, so explanations would need to accommodate specific levels of prior knowledge. Furthermore, diversity between individual operators might be substantial and warrant personalization approaches.

Q3: How many rules may the served model contain before being considered too large? Smaller rule sets are easier to generate a general understanding on, while larger rule sets can provide a more diverse coverage of the input space and, therefore, potentially more accurate predictions. In some cases, like explanations for specific decisions, the entirety of the rule set might not even be of interest and stakeholders may prefer explanations to be limited to the rules whose conditions matched the situation.

Q4: What form can conditions take before they are too complex to be understood? Many rule representations have been proposed in the past and while ellipsoids or neural networks can provide improved results, hyperrectangles (simple interval for each input dimension) might be easier to comprehend. Typical decision trees and random forests use hyperrectangular conditions with non overlapping feature space partitions. Beyond those options, there exists the LCS-specific concept of Code Fragments, a program tree–inspired way of capturing non-linear decision boundaries [IBZ14b]. They are fairly human-readable, albeit less than hyperrectangles, in comparison to neural networks. This should also probe whether the exact condition is even considered relevant or if operators are content with knowing that it applies to a certain instance. However, counterfactual-based explanations might be a worthwhile effort enabled by clear decision boundaries that can be understood by humans, i.e. if the situation was slightly different another rule might apply, changing the model's prediction.

Q5: How important are explanations of why the decision boundary of a rule is placed a certain way? In LCSs, the model structure (and decision boundary of each rule) is optimized using a metaheuristic to localize the rules in a way that they fit the data well. Within this question, we want to ascertain how important insights into this process are to operators.

Q6: What form can local models take before they are too complex to be understood? While linear models are widely regarded as easily comprehensible, more complex models might yield better results and typical explanations, such as feature importance analysis, may satisfy the stakeholders' want for understanding the decision making process. This also translates to the usage of mixing models (where multiple rules are used to construct a prediction) and the comprehension thereof.

Q7: What information do stakeholders request about the training process? This question aims towards the training in general and what steps are performed in the process towards deriving a model rather than at an analysis of the utilized model. An important aspect of this can be the gathering, cleaning and selection of data and responsibilities therein.

Regarding the specific model (and algorithmic) design decisions practitioners can base on the outcome of those questions, we want to highlight Section 3.5.6. In general, questions Q1 and Q2 serve mostly to determine the trade-offs of explainability and performance and the differences between stakeholders and individuals. They might tell us that we should train and deploy individual models for optimal acceptance and stakeholder satisfaction. They might also highlight the need for different visualization and analysis tools for the models or the form in which explanations should be given, regardless of model design. Q3-Q6 are more specific for rule-based systems such as LCS and decision trees. These also give the most insights into what the deployed model should look like. Q3 gives the relevance of rule set size in the optimization process (for batch multi-solution (Pittsburgh-style) systems directly during training, for online single-solution (Michigan-style) systems in post-training compaction). Q4 answers directly what condition scheme rules should use, whereas Q6 answers the same for the rules' predictive model. Both decisions are usually made before training is started. Q5 determines whether the training algorithm itself should be explainable (or to what degree this is needed). There are some approaches into making the stochastic optimization of evolutionary algorithms explainable [Bac+22], albeit not specifically focussed on LCS training. Q7 is again more general and not focussed on LCS model design but rather on the conditions surrounding training, e.g. who (individuals, departments) was involved or what data was used (and, importantly, what data was not).

3.5 Case Study: Assisting Operators in a Chemical Industry Plant

To demonstrate a potential use of our proposed template of questions to determine the requirements for a self-explaining socio-technical system that supports operators in their day-to-day tasks while also satisfying other stakeholders needs, we performed an interview-based case study. In this case study we interviewed a variety of different stakeholders about their individual as well as their colleagues' and subordinates' needs for such a system before its final design and implementation. Note that this study serves as an example into how to apply our proposed questions and its answers will likely be very use case–specific and might not be transferable to other use cases. This issue of non generalizing answers is very typical for similar studies regarding explainability needs [BP21; Her+22]. Therefore, we have to work with small sample sizes (as few individuals in a specific stakeholder role exist) and can not apply many of the quantitative analysis tools that might be available for large scale studies. The envisioned operator assistance system (OAS) is to be employed in an international chemical industry company, the REHAU SE [3]. REHAU plans on piloting it in a German plant of their interior solutions branch, which is the main focus of our case study, where so called *edge bands* are produced. However, we also interviewed a stakeholder from a plant of their window solutions branch to broaden the scope, potentially find differences even between branches of a single company and, hopefully, find some answers that can be applied to other branches in the future as well.

3.5.1 Operator Assistance System

The primary motivation behind the operator assistance system (OAS) is to disencumber operators and reduce their overall workload, which currently is substantial. This is to be done through increased automation of, currently manual, routine adjustments and by providing operators with more insights into disturbances and with potential solutions. Overall, this increases the robustness of the production and reduces material and energy waste.

In the line control an OAS-like system assists operators at manual configuration of individual machines in the line or overarching parameters and partially automates it. Its components are largely well understood from a chemical engineering point of view. While arguably some level of explanations to operators could always be beneficial, these algorithms do not employ any form of ML component and therefore fall out of scope for our study, where the focus is an SL agent operating as one of the, potentially many, systems forming the overall OAS. Another component currently in production is a tool that aggregates existing knowledge in an easy to navigate tree-like structure. When encountering some issue, e.g. a quality defect, the operator navigates via web interface from broad areas to specific defects/disturbances where individual stages are described both textually and visually. Once the issue is narrowed down, the operator is presented with common solutions to the problem and an estimate on how successful these have been in the past. After the issue is resolved, the operator is asked to give

[3]https://www.rehau.com

feedback whether the provided suggestion was helpful and correct, promoting this suggestion for the next operator that encounters this issue. These paths through the tree-like structure can be reformulated as rules. Those rules can potentially in turn be used inside an LCS, either as an initial population before training or by manual insertion into the trained model, where they serve to cover areas of the problem space where training examples were too scarce to create sufficiently accurate rules. Additionally, these rules can be used for potential explanations of evolved rules, as they should—due to their crowd-sourced nature—be deemed more reliable by operators than some rather high-level and maybe opaque ML process. For simplicities sake, we refer to the envisioned SL-based agent as part of the OAS as the *agent* in the remainder of this text and primarily consider its specific requirements without limiting other components.

Depending on its maturity, predictive power and stakeholder trust, the agent can be employed at different levels:

1. Predict the quality of a machine parametrization selected by the operator,
2. actively make suggestions for possible parametrizations and their predicted product quality to the operator,
3. set a single parametrization and prompt the operator to confirm and
4. regulate the process parameters fully automatically, e.g. when product quality or process stability indicators drop, with the operator only acting as a supervisor.

These levels also change the operator's role in our socio-technical system of machine, agent and operator in that the higher levels lessen the mental load of trying to come up with possible solutions and transform the operator to an executor of physical adjustments and tasks while keeping them in a position of supervisory responsibility. Likely, different settings in which the agent is to be used will allow higher levels of operation earlier. In less crucial (i.e. not sensitive or prone to significant damage) parts of a production line, the agent will be able to choose from a wider range of still sufficient parametrizations while facing less scrutiny by different stakeholders. The same holds for areas with different data availability and quality. Ultimately, any SL prediction is dependent on diverse and correct data for training. Machines of a line that have long been digitized and fitted with well calibrated sensors will more likely offer such data than machines that have until recently been controlled by analogue means. For these newly digitized machines, it might even be unknown what sensors are missing to make meaningful predictions and they might not yet have been online long enough to gather sufficient data or even to allow the determination of what noise is to be expected during operation, e.g. the impact of seasonal changes.

Regardless of the specific scenarios, it is clear that to get such an agent into production, relevant stakeholders have to be on board from the early stages of its design process. This was also reflected by those stakeholders in early talks about potential use cases. In these talks they first raised the, albeit expected, issue of transparency of such an agent and its decisions as central towards generating enough trust to employ it.

3.5.2 Extrusion: An Example Application of the OAS and its Agent

In the production of plastics, a typical first part of a production line is the melting of synthetic granulates (or powders) and subsequent form-giving extrusion of the heated semi-fluid mass. The correct pressure—and for many products also the temperature—is crucial to ensure sufficient dimensional accuracy and therefore product quality. The exact values are primarily dependent on size, shape and material type, but from a process engineering point of view it is very much possible to find a range of values that can be considered sufficiently optimal to guarantee the desired product quality. Operators will control for this measurable parameter rather than shape and size itself, as process engineering guarantees desired dimensionality whenever the correct pressure was applied. This also has one key advantage for prediction: The resulting learning task is a regression for which sensor readings are comparatively easy to obtain, whereas the control of a multidimensional shape and size vector for which complicated and highly accurate laser scans would be needed is much less straightforward.

In REHAU's interior solutions branch, specifically edge band production, extrusion pressure is regulated by eight adjustable parameters. Additionally, a multitude of additional sensor readings, primarily temperatures in different sections of the extruder, are available. The adjustable parameters show highly non-linear relationships with the target, warranting sophisticated self-learning and—due to the requirements on transparency—self-explaining systems.

3.5.3 Study Design

One critical issue to be solved to actually get the agent into use in a scenario similar to the one presented in Section 3.5.2 is stakeholder acceptance. This acceptance needs to be nurtured from the early design stages by making choices according to the wishes (and worries) of the various stakeholders. From early preliminary talks with R&D and different management levels, we already knew that whatever the exact embedding system design would be, the self-explainability of the employed agent is likely central. This already hinted towards an LCS being a very plausible choice for the learning algorithm. Thus, we use the template raised in Section 3.4 with relevant stakeholders to determine if the assumptions that explainability is very important are even correct and, if so, how the resulting LCS model should likely be designed. This serves a second purpose, as discussing these issues in the form of the questions with the stakeholders allows them early participation in the design process and can be used to develop the OAS according to their requirements. This reinforces the perception of holding a stake rather than the feeling that some ill-suited system was forced onto them. In another direction but complementary to the described goals this also facilitates a test of the validity and applicability of the questions raised and whether they even allow meaningful insights. This is an important consideration for potential future applications of the template (or if they turn out to be suboptimal for a reformulated version).

To validate the applicability of the questions and to gain some perspective on what answers we can expect and where additional clarifications or input might be warranted, we conducted a pilot interview with the Director of Smart Factory and selected members of his department, which is responsible for machine automation, data science, IoT, assistance systems and sensors. We found that the questions can be used as proposed in Section 3.4, but some more explanations, especially into the specific nature of LCS models, are beneficial to get more useful answers for the LCS-specific questions. Importantly, we found that explanations are definitely desired on many levels. More results are discussed in Section 3.5.4.

Our main study is conducted in individual interviews with stakeholders of about 45 minutes. As all participants were German and few work with English on a daily basis, these interviews were conducted in German. Interviews began with the interviewee prompted to give some information about themselves and their current job as well as job history at REHAU. After a short introduction into the general topic, possible levels on which the agent can operate and an example use case based on the extruder (cf. Section 3.5.2), the stakeholders were presented with the seven questions and some additional explanations, examples and follow-ups. The questions were also reformulated into German and technical (machine learning) jargon was—where possible—kept to a minimum. As LCS (and other ML model types) were unknown to most participants, an example of a 1-D task solution and an eight-dimensional example rule were also presented before question 3, where the number of rules is discussed. Interviewees were strongly encouraged to ask for clarifications if some point of a question was unclear and received some additional context or details if they expressed trouble answering. The interview was aided by a set of slides, so interviewees could read along and reread the question if needed. These slides can be found at https://doi.org/10.5281/zenodo.6505010.

The relevant archetypical stakeholder roles can be summarized as follows:

- *Operators* operate the machine to manufacture a product. Typically, operation takes place in a one to one ratio in the interior solutions branch and sometimes in a one to many ratio in the window solutions branch. They interact with the agent throughout their shift and, as they are responsible for smooth production, rely heavily on its capabilities. Especially (comparatively) inexperienced operators often need assistance, whereas seasoned (10+ years of experience) operators will rarely be in situations where they consult others.
- *Team Leaders* supervise a group of operators on the shop floor within a given shift. For troubleshooting, team leaders are the subsequent responder when the colleague on the next line was unable to assist an operator. Therefore, they interact with the line control (and thus the OAS and the agent) on a frequent basis. If even some of the operators' questions and issues get resolved by an agent, the team leaders' job becomes considerably less stressful, while if the agent gives poor advise or confuses the operator, their job might become more difficult.
- *Production Managers* are ultimately responsible for the entire plants production and are thus very interested in past and projected manufacturing capabilities.
- *Process Engineers* have the deepest knowledge of the underlying process. They have deep foundational understanding towards maintaining process stability, which they are also constantly trying to improve. They operate either closely to/within the plant, where they

are second in line for troubleshooting when operators and their supervisors could not fix the issue at hand by themselves, or in more centralized process engineering departments, where they determine set parameters and machine configurations for new material compositions and product types and perform other developmental steps towards machine improvement and innovation.

- *Data Scientists* are expected to maintain, improve and expand the capabilities and possible applications of the agent (and other ML methods in use). They select which data is to be used, what new sensors are needed and validate the correctness of readings. From a model perspective, it is likely that the agent encapsulates multiple models that are directly trained to predict on this machine (model) or even for a specific product, rather than a singular generalized model that solves all tasks, although generalization is overall desirable as fewer models are easier to maintain. Data scientists would thus need to determine which machines and products can share a model and for which combinations other models are needed. Ultimately, a badly performing agent is the responsibility of the data science team.

We want to add an important disclaimer for this specific study that is, however, very likely also the case for most similar studies: These stakeholder archetypes are often not clearly distinguished in a single person and their view on certain aspects might be heavily influenced by their (job) history so that despite their current position, they still express views we can clearly attribute to another archetype. This constitutes a form of bias which needs to be accounted for when drawing qualitative conclusions based on such studies. With large enough sample sizes, this might average out. However, in many typical industrial settings the number of individuals in a given position may often be too small [BP21]. Team leaders are often trained process engineers that have been operators at REHAU before undergoing additional education. In-plant process engineers often have a management role as well, with responsibilities for sections of the plant. Although, in this archetype specifically, the exact position of a person between R&D responsibilities, where university graduates are more common, and day-to-day operations widely varies. We still chose to present these as one archetype as the general questions they ask of the agent are similar. The interview partners available for this study were selected to allow an overview of all roles and interviewees were asked to distinguish between the different roles they might find themselves in or have held in the past for their answers. They were also requested to separately answer for operators and based on their perception on operator's requirements.

3.5.4 Interview Findings

From the conducted interviews we find that there is a need for self-explainability of the envisioned agent and that simpler models are generally preferable. More detailed descriptions of the answers to the seven questions are shown in Section Pilot Interview for our pilot interview and Section Main Interviews for the main interviews that were conducted afterwards. Reassuringly, we also found that the agent is indeed wanted.

Pilot Interview

In this first interview we primarily aimed at validating the applicability of the proposed questions to gain insights regarding the envisioned scenario. It was conducted with the Director of Smart Factory at REHAU in the presence of some members of his department, who were involved in the already existing parts of the OAS and a variety of data science applications. The concept of a SL-based agent to assist operators and its various levels of application were well established previously. Answers, as given by the Director regarding his perception of various stakeholders' requirements, were recorded and are stated in the following:

Q1: To what extent does a stakeholder request explanations? For operators, this primarily depends on the autonomy of the agent. The more autonomous the agent acts, the less will the individual operator request explanations. In contrast, the process engineer will always want in-depth explanations. This requirement will likely increase with agent autonomy, e.g. when debugging potential issues, as the operator will have less insights into what was configured and why. Team leaders will require more explanations and more depth than operators. Data scientists will want maximal transparency and self-explainability.

Q2: What are the differences within a type of stakeholder? For operators the frequency and depth of explanations will highly depend on their experience. Experienced operators will probably disregard the agent completely and use their own knowledge to solve upcoming issues. Thus, they will also not request any explanations. For other stakeholders, experience might matter for simple tasks, e.g. if the prediction aligns with their mental model, they will not request an explanation, but overall explanations will be requested by all personnel in these roles.

Q3: How many rules may the served model contain before being too large? As LCS models and their structure's implications were not completely clear, we presented a small ad-hoc visual aid what an LCS model might look like, which we then also kept for the main interviews. Data scientists may be the only stakeholders that might want to analyse the model in its entirety. Other stakeholders, specifically operators and team leaders, will be primarily interested in the model's situation-specific predictions. Therefore, explanations of the given mixing model will be more relevant and the global model can contain a large quantity of rules as long as it can still be experimentally or statistically verified, i.e. through well-chosen test data. The mixing model should also contain few rules. This question also brought up a point about local model: They should be trained in a way as to directly determine the most important features/parameters for a given prediction, e.g. by forcing 2-3 coefficients to be considerably larger than others in a linear model.

Q4: What form can conditions take before they are too complex to be understood? Interval-based rule conditions are strongly preferred over other options for both regression and classification tasks.

Q5: How important are explanations of why the decision boundary of a rule is placed a certain way? Operators and team leaders will probably not have this question and take the conditions as is. Some trace-back to the training sample might be interesting for data scientists but is not needed.

Q6: What form can local models take before they are too complex to be understood? This question was deemed impossible to answer without taking the model's task and performance into account. In general, simpler local models are preferred.

Q7: What information do stakeholders request about the training process? The Director was unable to confidently provide deeper insights into this question. Likely, information is of interest but the exact levels would need to be answered by the respective stakeholders.

Main Interviews

Following the findings of the pilot interview, an expanded introduction into both the possible application of the agent as well as LCS was prepared. Additionally, the seven questions were translated into German and, where applicable, follow-up questions based on answers given in the pilot interview were formulated. After that, four interviews were conducted. As this group of stakeholders was quite heterogeneous with different perspectives on the questions as well as operators' views, we attribute the (paraphrased) statements to the respective interviewees (A through D).

- **A** is currently a process engineer and supervisor with administrative responsibilities in edge band production. They started in the company as an operator and then became team leader before the promotion into the current position. They supervise and interact with operators and machines throughout a normal work day.
- **B** is from the window solutions branch and head of recycling and plant optimization. They started as an operator before training as a process engineer and receiving various promotions up to plant management. Therefore, they have a good perception on all relevant in-plant roles and might already give some perspective if the answers can be re-used for a similar manufacturing process for a different product at another plant.
- **C** is head of the data lab—a department responsible for all data management, analysis and science. They have a strong statistics background and have been working with various stakeholders from multiple plants for years. This includes directly interacting with operators at the machines over long periods of time.

- **D** is a member of the data lab. Originally part of R&D, they have subsequently joined the IT department and then—with its foundation—the data lab. They are primarily responsible for keeping data-related systems, like the envisioned agent, running and up-to-date.

In this section, the answers of the interviewees regarding the various stakeholder archetypes are presented in a question-wise manner. Where conflicting answers were given we present both.

Q1: To what extent does a stakeholder request explanations?

- *Operator*: New operators are thankful for all assistance, including explanations (A). Explanations also enable them to fix issues on their own (A, B). In general, short textual explanations of 2-3 sentences are preferred (A, B). Probabilities of success of a proposed parametrization and rule quality could be useful but are not mandatory as long as the model itself is not guessing (B). Explanations should be offered on request rather than by default on every prediction/re-parametrization (B). They could be enriched with images of issues that may arise from the suboptimal parametrization or other information about past production (A, D). Graphs and dashboards are not useful for operators (A, B). Neither are mathematical formulae (A, B, C, D). As long as the performance is on some generous level of practical equivalence, transparent models are preferred over better performing ones (A).
- *Team Leader*: In addition to textual explanations, graphs can be useful to understand and improve the manufacturing process (A). However, as long as production proceeds as scheduled, team leaders might not care for explanations (B). Not-as-explainable models with better performance can still be useful (A).
- *Production Manager*: The main interest is with keeping production up and efficient (B). Understanding why errors are occurring is of deep interest as to prevent them in future (B). In addition to textual explanations, which are likely too low level for most situations in which management is involved, high-level dashboards and graphs allow them to understand their production (B). Model transparency is more important to them, but in the end pragmatism reigns (B).
- *Process Engineer*: Being tasked with both ad-hoc debugging and long-term improvements, process engineers have a deep interest in understanding the manufacturing process (A). ML models that may infer connections from data that are unknown or at least unquantified by humans are of great relevance to achieve their goals (A). However, to be analysed these models need to be as transparent as possible (A). Diverse tools for in-depth explanations are very important (A). Process engineers might not analyse every decision but all that went wrong, as well as the general model (A).
- *Data Scientist*: Ideally, the model would be a complete white-box as transparency and explanations are preferable (D). However, a substantially worse white-box model should be replaced with a gray- or black-box system that undergoes a rigorous statistical analysis (C, D). A well-validated model that can be inspected via graphs and dashboards could be

deployed even without inherent transparency (C). Depending on the task, transparency could also be approximated via post-hoc analyses, although this would make the usability for other stakeholders questionable, depending on the correlation between the original black-box and its transfer learned pendant created through post-hoc analysis, such as LIME [RSG16] (C). Explanations should be in-depth and may include mathematical formulae (C, D).

Q2: What are the differences within a type of stakeholder? For all stakeholder archetypes, substantial experience will result in some predictions and decisions being obvious, thus, not requiring explanation (A, B, C, D). Data scientists might still want to understand how the model inferred this from data but this would not warrant a self-explaining model (C). Less experienced stakeholders will often require more or more in-depth explanations than those with average experience, although on the other hand, very experienced stakeholders might in turn require more depth to be convinced or to understand how the model found something they did not (A, B, C). Whether or not explanations are requested is mostly dependent on attitude and motivation rather than experience (A, B). The broadest spectrum is shown within the operator role (A, B). For inexperienced operators, consulting the system replaces disturbing their colleagues and/or supervisors to ask for their help, which will increase the agent's acceptance (B). Personalization of explanations might be good for individual operators but would greatly complicate the team leaders' and process engineers' user experience whenever they are called for assistance (A).

Q3: How many rules may the served model contain before being too large?

- *Operator* and *Team Leader*: The overall models number of rules can be as complex as needed, however, in a given situation only few (up to 4 (A)) may match and be included in the mixing model (A, B, C, D). Additionally, rules should ideally be limited in a way as to promote high weights for only 3-4 features at most, with other features having considerably smaller weights (A, D).

- *Process Engineer*: A process engineer will often analyse the full model and therefore requires it to be small (A). However, the exact size is problem dependent (A). For the extrusion problem, 15-20 rules should be an upper limit (A, D).

- *Data Scientist*: Matching rules are more important than the totality of rules (C, D). Intense validation of a subset of rules will likely allow data scientists to trust the other rules as long as they share performance metrics (C). Overall, rule similarity is also important in that many dissimilar rules are more acceptable than high overlaps (C). However, upon further probing, sizes of 30 to 100 rules were deemed as highly complex models for successful analysis (C, D).

Q4: What form can conditions take before they are too complex to be understood?

- *Operator*: For operators the specific condition does not need to be analysed as long as we can assure that this rule does apply (A, B). However, when interacting with operators to explain certain decision making processes, complex models might make this more difficult (D).
- *Team Leader* and *Process Engineer*: Easier to analyse is preferable (B) . They should not be more complex than intervals (A).
- *Data Scientist*: More complex conditions should be possible as long as they undergo post-hoc analysis, e.g., LIME (C). If the LCS has proven to produce well-placed decision boundaries for similar problems, not all rules of every model would need to be analysed in future applications (C). For practically equivalent performance, easier conditions are strongly preferred (D). With a higher degree of automation, analysing the condition becomes more important (D).

Q5: How important are explanations of why the decision boundary of a rule is placed a certain way? The interviewees were in agreement that there is no need for explanations why the trained model exhibits certain decision boundaries and how the optimizer found these. The data scientist might have an interest into the process from a scientific point of view but for machine operation and operator assistance through a trained model, this is not relevant (C, D).

Q6: What form can local models take before they are too complex to be understood?

- *Operator*: Operators will likely not value model specifics as long as a textual explanation for the central aspects (e.g., feature importance / influence of individual features on the prediction) is given (B). Models should be linear (A). An analysis of the mixing of the currently matching rules is sufficient (A).
- *Team Leader* and *Process Engineer*: Local models should be kept as simple as feasible (B). Ideally, local models are linear (A). In addition to an in-production use, process engineers will also want to analyse the model(s) to improve the process itself, e.g. through changes in hardware, and for this the models need to be understandable to them (D).
- *Production Manager*: Individual predictions are less important than overall system performance (D).
- *Data Scientist*: The usage of more complex local models should be possible (C, D), although simpler models are always preferable (C). If complex models are used, they would need to undergo rigorous individual testing and analysis (C). However, for better performing local models, this would be worth it (C, D).

Q7: What information do stakeholders request about the training process?

- *Operator*: The more information is available, the higher will be the operators' trust in the predictions (A). They care about which lines and which products were used for information gathering and by whom features were selected and models were build (A). As long as predictions are correct, operators will take the suggestions as is and not further request such information (B).
- *Team Leader*: More detailed information than for operators as well as some form of involvement in the design process is requested (A).
- *Production Manager*: Some information on a high abstraction level is sufficient, e.g. where did the responsibilities lie (B). Deployment time, lifetime performance and possible adaptations based on products and performance are relevant (D).
- *Process Engineer*: To determine if model performance is in line with the current understanding of the process, and to subsequently improve process stability on the basis of the models production, engineers require as much information as available (A, B, C). They should also be involved early on to avoid model biases from possible correlations without causation within the data (C).
- *Data Scientist*: While multiple stakeholder archetypes will request all information available, more than anyone else data scientists will want to do statistical testing and analysis of the models (C). They will analyse train-test-splits in detail (C). With the model in production, they employ statistical measures to detect possible concept or sensor drifts (C).

3.5.5 Summary

We find that stakeholder archetypes have—at times substantially—diverging requirements towards the explainability of the model. All stakeholders would prefer transparent models as long as performance is practically equivalent. However, should this not be the case, it highly depends on both the archetype as well as the individual person.

Within the group of operators, some might not ever consult the agent and many might not care for its explanations as long as predictions—or derived parametrizations—are correct. Regardless, substantial numbers of operators will both follow the agent's suggestions and check its explanations. These explanations can serve two purposes. On one hand, they help operators check for plausibility of a decision based on their own mental model and therefore increase trust in the agent. On the other hand, they may update the operator's mental model, which is especially important for newer and inexperienced operators that would otherwise need to rely on a colleague's or supervisor's assistance. Regarding LCSs, operators tend to only want to analyse currently matching rules. These should be few in number and kept as simple as possible. Operators want explanations primarily in a short textual form, ideally, directly generated from those rules. With this role especially, we found staunch differences between the two plants, where interior solutions' operators want explanations much more frequently than their window solutions' counterparts, where explanations are likely only requested in case of production issues and defects. While these might be attributable to the interviewees, it is very plausible that differences in the manufacturing process and how machines are interfaced with within the socio-technical system are the root of diverging answers, e.g. the fact that within window solutions multiple lines are operated by one operator.

Team leaders largely follow the trends set by their direct subordinates, the operators. However, due to their increased responsibilities, they require more, deeper and more diversely represented explanations from easy to analyse models. Again, the two plants seem to differ with regards to frequency of explanations for this role, although the trend is less substantial.

Production managers will less frequently interact with the agent and primarily require information about its performance and if that is poor, will request more information into likely reasons. The agent could, for example, explain its poor performance in certain areas of the feature space with poor sampling, high noise or unexpectedly complex parameter-target relationships. Individual decisions are unlikely to be analysed by production managers. However, depending on their background, they might be quite interested in what is running in their plant and how it works from a personal motivation.

The process engineer requires the most in-depth and diverse explanations and general model analysis capabilities. From our interviews, we found a second aspect of usage for the models besides their application within our operator assistance setting. Namely, to analyse the models (or the agent in its entirety) to deduce process improvements that go beyond a parametrization. This can range from the hardware setup itself to chemical mixtures of line inputs to hydraulic valve switching. The simplest models are strongly preferred for both aspects. Process engineers are less diverse in their requirements, both from an individual as well as a plant-wise perspective. Contrastingly to the perception expressed by the Director of Smart Factory, not only data scientist but also process engineers will want to analyse the full model.

Data scientists were overall relatively open to deploying gray-box or even black-box models as long as they had undergone substantial statistical verification or have been made explainable through post-hoc analyses. However, transparency is preferable as statistical verification of a black-box model can be sufficient if one deeply understands the statistical decision making process and possible fallacies therein but is hard to convey towards stakeholders that do not have such knowledge and training.

By gaining an understanding of the various stakeholders' requirements through this study we also validated that the seven questions are useful to determine them. We found that differences between the two plants seemingly exist for some but not all of our identified stakeholder archetypes. Likely, different domains and companies will also yield slightly different answers. Thus, these questions should serve as a template on how to design specific studies. Additionally, we found that potential users not only want to be included in the agent design process but also have important uses for the agent and its models that are not included in the originally envisioned case that can, however, also be solved without a differently or separately designed system and did not come up in any other previous discussion.

3.5.6 Consequences for Model Design

While the last section focussed on highlighting the most important interview findings on both the likely interactions with the agents and their respective models and the corresponding high-level explainability requirements, e.g. how to present explanations to operators versus team leaders, we want to focus on some more technical aspects regarding model (and algorithmic) design in this section. We want to again stress that the template of questions primarily focusses on the returned model after training completes.

With Q5, we inquired about the training process itself and found that—in the presented use case—it is largely irrelevant for the comprehensibility of an OAS agent. Therefore, the LCS's algorithmic design can be freely chosen. As long as the resulting model fulfils certain criteria, it is irrelevant whether practitioners use a batch or online, single- or multi-solution (Michigan-style, Pittsburgh-style, or hybrid) approach. The same goes for the deeper details, e.g. how model updates or credit assignments are made, the fitness function, when to use subsumption, the specific compaction technique, how to cover or what evolutionary algorithm to apply. None of the stakeholders expressed deeper interest in detailed knowledge about those mechanisms for the use case in the OAS. Data scientists are interested in this from a professional point of view (and are likely the ones ultimately making those decisions) but also stressed the importance of the model rather than the training process. Not having to derive explanations of the behaviour of evolutionary algorithms is also greatly advantageous for practitioners as this is a challenging open issue [cf. Bac+22; Zho+24].

While the algorithmic side of training was not confined by requirements, the expected structure of a trained model is. This does of course have indirect repercussions on the algorithmic properties of the LCS and how it should be configured to best arrive at such a model. The more straight forward aspects are the wishes for interval-based (hyperrectangular) conditions and linear models (for regression). Both aspects are typically defined before training. For the linear models needed for regression we found that some confinements should be made regarding their coefficients as stakeholders would prefer it if few (i.e. about 3) of those were large at a time to easier pin-point influences. Coefficient control is a very common requirement for ML models and typically solved by some form of regularization, e.g. the well-known Tikhonov regularization ("Ridge Regression") to keep coefficients small. How to design this component in an LCS depends on how the updates are made. In online-learning (e.g. Michigan-style) systems this is less straightforward than for batch learning–based systems (e.g. Pittsburgh-style) as in those systems the traditional regularizations can be used directly. An easy to implement option for online single-solutions systems would be to use the returned rules' matching functions but retrain their linear model's coefficients in the same way. A very important aspect for some stakeholders lies in the quantity of rules, both the total number of rules as well as how many rules partake in a prediction. The total number of rules is controlled via subsumption and compaction in online single-solution (Michigan-style) systems and directly in the fitness function and by pruning techniques in batch multi-solution (Pittsburgh-style) systems. For fitness-based control a practitioner could assign rule sets over a certain size an arbitrarily low fitness. For compaction-based control (which is applied after training completes) an algorithm has to be chosen that offers an option to shift the performance/size-trade-off in a certain direction. One straightforward option would be to use a binary genetic algorithm for that and again head towards fitness-based control. How many rules partake in a prediction is defined by the mixing-model. A very common mixing-model approach is to use a fitness-based weighted average of the prediction. Here, it would be quite simple to adapt the mixing-model to only take the top n (e.g. three) rules according to the fitness into account. As this might change where rules should be placed to make smoother decision boundaries, we strongly recommend to use such mixing restrictions already in training rather than afterwards.

Results like these almost automatically raise the question whether they generalize. The answer to this is complex and layered but I think general trends are similar in many domains. First, I am quite confident that they generalize to machines producing different products (but still plastics that are extruded) within the same manufacturing plant as where the OAS will be deployed. They very likely also generalize to different plants within the same company with plastic extrusion–based products. However, we might already encounter different setups, e.g. more automation on the shopfloor, that shift the patterns of interaction between human and machine, possibly removing the requirement for explainability as stakeholders are already comfortable with complex approaches governing their production. On the other hand, stakeholders at less advanced sites might be happy with any model that reduces their defect rate and feel like the potential advantages are so relevant that losing explainability is acceptable. I suspect that in this case the answers of what they would like to see would be similar but that it would rather be a nice-to-have feature than a hard requirement. Stakeholders from other producing industries I would expect to give similar trends as in this use case but I would strongly recommend to exercise through the same process and collect potential requirements without any bias. Applications of intelligent agents outside of the manufacturing domain could show similar trends if they are currently performed by human expertise with significant potential losses in money or other factors. Medical applications, e.g., for systemic decisions like treatment plans, would probably be less restrictive as doctors would take the time to verify the system in great detail and then have greater trust in other doctors that confirmed the results, however, I expect such applications to have hard requirements for both model and individual decision explainability in general.

Overall, although we find that practitioners still have large amounts of algorithmic freedom to train their models, we can make some clear restrictions into the expected final model structure.

4 The Supervised Rule-based Learning System

The primary contribution of this thesis is a new approach to train Learning Classifier System (LCS) models.[1] Chapter 2 presented thoroughly how this family of algorithms operates and what the trained models look like. We find that the designers of major LCSs do consider neither their intended model structure nor the optimizers that arrive at that structure as the primary motivator for design choices. This leads to a situation where the control over the resulting structure is more limited. However, we find from the case study presented in Chapter 3—and especially Section 3.5.6—that the amount of control existing major LCSs offer us over relevant optimization targets—most importantly the model size—is too limited for similar applications.[2] Section 1.1 did provide a thorough description of the research gaps that effectively arise from *C1* and *C2*.

This chapter introduces a new LCS: The Supervised Rule-based Learning System (SupRB). Its design was conceptualized based on the increasing need for explainable machine learning models. While most LCS models are typically showing some form of inherent interpretability and transparency, SupRB can control these directly by means of its split optimizer(s) which is novel in the field. Critically, SupRB does not try to solve multiple different optimization tasks (cf. Section 2.1.3 for those tasks) with multiple competing algorithmic heuristics but splits the tasks clearly and assigns each optimizer one of them. This disentanglement of model selection (and fitting) tasks makes the system easier to follow and gives it a clearer design structure which can not only help explain it to stakeholders, but also when troubleshooting or trying to improve the system, e.g. by building a specialised solution for a specific use case. Moreover, we will find in Chapter 5 that this allows SupRB to find much smaller models than competing LCS and rule-based ML approaches while staying competitive in terms of errors. Together with Chapter 5 and Chapter 6, where I will discuss possible improvements of the algorithm or its models, this chapter composes *C3* as defined by Section 1.1.

4.1 General Description

This section is a culmination of all previous presentations of SupRB, beginning with the initial concept paper [Hei+22c], over numerous papers offering extensions or additional stud-

[1]LCS are also known as a rule set learning and evolutionary rule-based machine learning algorithms.

[2]I would argue that most real world applications—including outside of manufacturing domains—will exhibit similar issues of requiring more control about model sizes.

ies [Hei+22b; Wur+22; Hei+22a; Hei+23c; Hei+24] to its most thorough description in the Journal of Applied Soft Computing [Hei+23d]. The latter serves as the direct basis for the following text.

The Supervised Rule-based Learning System (SupRB)[3] is a new type of LCS, with alternating phases of rule discovery and solution composition [Hei+22c]. In the first phase, rule conditions are optimized independently of other rules. Thereby, a pool of diverse rules with convenient localizations is discovered. The subsequent second phase aims at composing an accurate but small solution to the learning task. To this end, the system utilizes an optimization algorithm to select a subset of all available rules, i.e. all rules that are in the pool after rule discovery. Therefore, SupRB diverges from other LCSs, as the *model selection* objectives—identifying a number of well positioned rules and choosing a set of these for the resulting model—are separated (cf. Figure 4.1). Note that the quality of the positioning of rules depends on a trade-off between the local prediction error and the matched volume. Altogether, this enables the prediction of any input with minimal error, but the smallest possible set of rules, maintaining high transparency and interpretability. How many rules are required exactly to compose a good solution, however, is difficult to determine beforehand. Therefore, the two phases are applied in turns until a termination criterion is reached, i.e. a specified number of iterations (cf. Figure 4.2 and Algorithm 1). The alternating phases provide a unique advantage: subsequent rule discovery stages can be guided towards less explored regions, which were up until then only covered by no or only imperfectly placed rules. To achieve this, SupRB incorporates in-sample performance information from the last solution composition phase. Note, however, that a rule, once it has been put into the pool, is fixed and will never be removed during the training, which stands in contrast to all major batch multi-solution LCSs. Overall, according to the categorization system from Section 2.3.1 SupRB falls into the multi-model batch learning category.

Algorithm 1 SupRB's main loop [Hei+22b]

```
1: pool ← $\emptyset$
2: elitist ← $\emptyset$
3: **for** $i \leftarrow 1$, n_iter **do**
4:     pool ← pool $\cup$ DISCOVER RULES(elitist)
5:     elitist ← COMPOSE SOLUTION(pool, elitist)
6: **end for**
7: **return** elitist
```

Figure 4.1 can be thought of as a single (or more specifically, the first) cycle through Figure 4.2. From data sampled from some given function (for presentation sake, this is kept one dimensional) a set of rules is discovered by selecting different partitions of the feature space and fitting a linear model in them. Then, some of these rules are used to build a good model from while others are discarded. This model is using a mixing model where the prediction of the

[3]The newest version of SupRB's implementation is always found at https://github.com/heidmic/suprb. The version used for all experiments in this dissertation is long-term archived at https://doi.org/10.5281/zenodo.14181292.

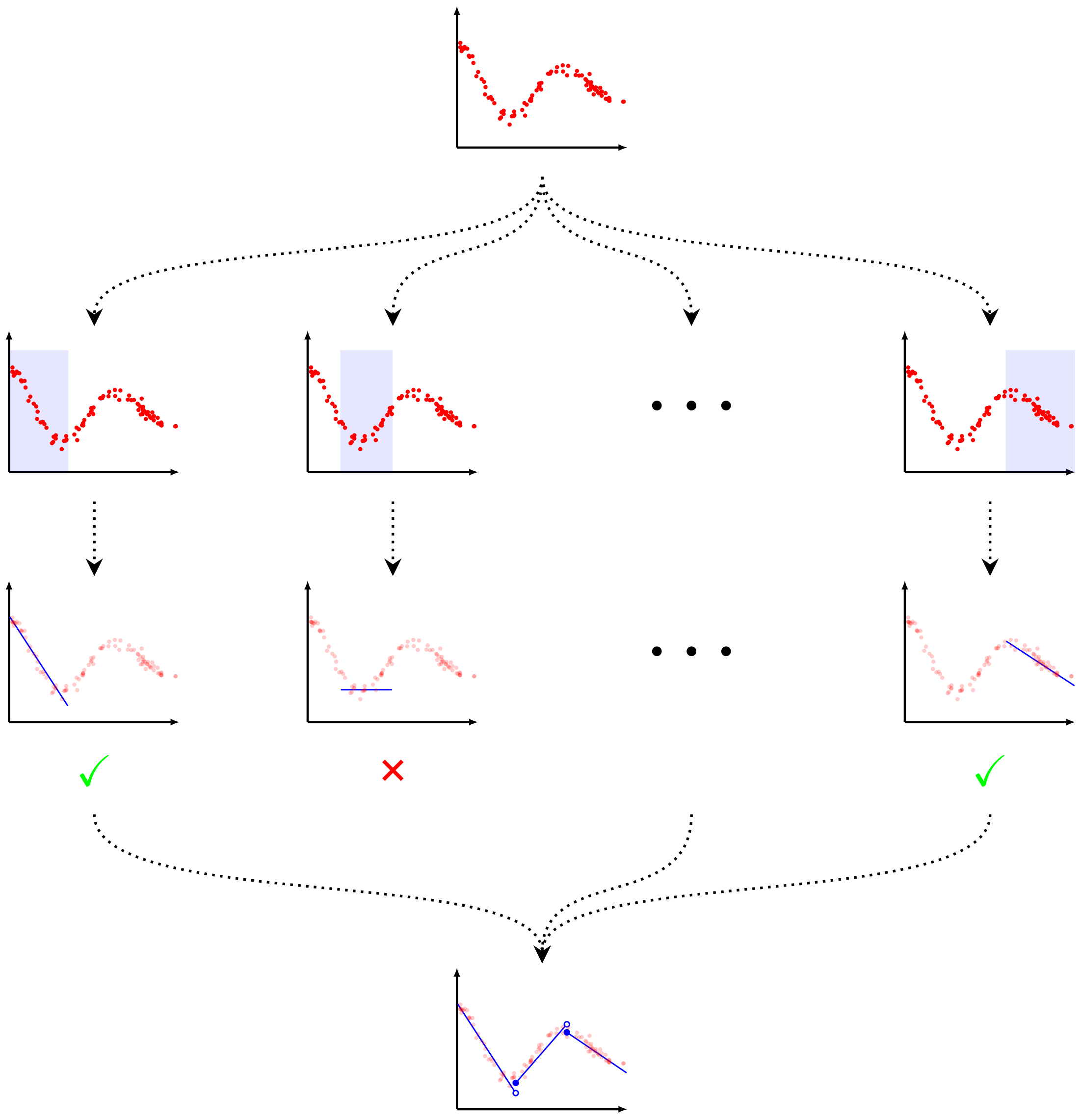


Figure 4.1: Extracting a rule-based model from data by discovering appropriate areas of the input space to place linear models in, creating individual rules. Afterwards, a subset of these rules is selected (denoted by cross and checkmark) to compose the output model from. In the example, we utilize three rules and use a mixing model where only the fittest rule per area is used for predictions. This figure first appeared in [Hei+23d].

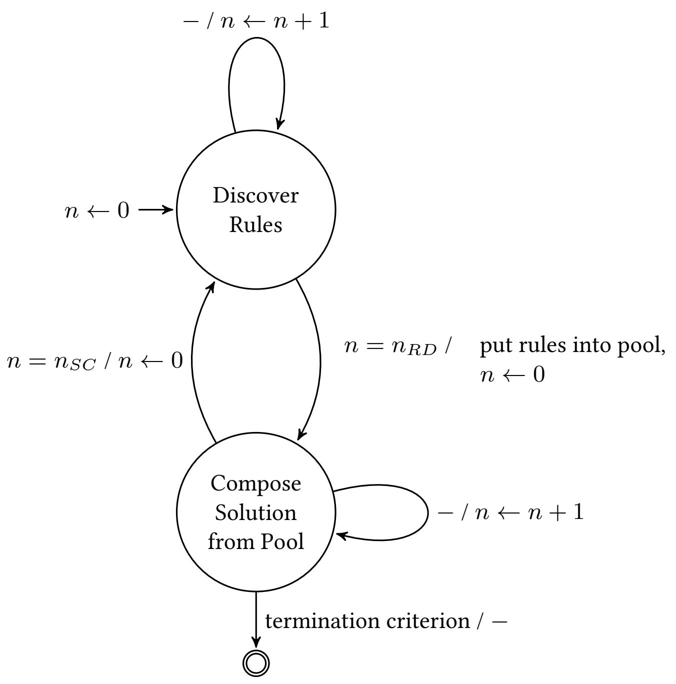

$- \,/\, n \leftarrow n+1$
$n \leftarrow 0$
Discover Rules
$n = n_{RD}$ / put rules into pool, $n \leftarrow 0$
$n = n_{SC} \,/\, n \leftarrow 0$
Compose Solution from Pool
$- \,/\, n \leftarrow n+1$
termination criterion / $-$

Figure 4.2: Rule discovery and solution composition alternation in SupRB [Hei+22c]. n_{SC} is the number of iterations for the solution composing optimizer and n_{RD} is the number of iterations performed for rule discovery.

best (according to experience and error) local model (sometimes also called a submodel) is used in regions of overlap. For a more detailed analysis of this and other mixing models, cf. Section 6.3. In general, this mixing model is not ideal for the type of function LCSs are meant to approximate and it should be preferred to use, for example, weighted averages. However, this type of mixing makes illustrations easier to read and better to understand for LCS novices. After this elitist (cf. line 5 of Algorithm 1) has been composed, termination is checked and the process begins anew, although for the next time a solution is composed not only the newly generated but also all previously generated rules will be available.

SupRB is strongly aimed at providing insights into model predictions. Therefore, attaining simplicity and interpretability of its model is essential [Hei+22c], which led to the following choices being made:

1. Rules' conditions use interval-based matching: A rule k applies for example x iff $x_i \in [l_{k,i}, u_{k,i}]\ \forall i$ with l being the lower and u the upper bounds.

2. Rules' local models $f_k(x)$ are linear. They are fit using linear least squares with a l2-norm regularization (Ridge Regression) on the subsample matched by the respective rule.

3. When mixing multiple rules to make a prediction, a rule's experience (the number of examples matched during training and therefore included in fitting the local model) and

Algorithm 2 SupRB's Rule Discovery [Hei+22b]

```
procedure DISCOVER RULES(elitist)
    rules ← $\emptyset$
    for $i \leftarrow 1$, n_rules do                        ▷ $(1, \lambda)$-ES for each new rule
        candidate, proponent ← INIT RULE(elitist)
        repeat
            children ← $\emptyset$
            for $k \leftarrow 1, \lambda$ do
                children ← children $\cup$ MUTATE(proponent)
            end for
            proponent ← child with highest fitness
            if candidate's fitness $<$ proponent's fitness then
                candidate ← proponent
                $j \leftarrow 0$
            else
                $j \leftarrow j + 1$
            end if
        until $j = \delta$
        rules ← rules $\cup$ candidate
    end for
    return rules
end procedure
```

in-sample error (the error on training data) are used in a weighted average. A rules mixing weight τ is defined by:

$$\tau_k = \frac{1}{\text{error}_k} * \text{experience}_k = \frac{\text{experience}_k}{\text{error}_k} \tag{4.1}$$

Using this weight in a weighted average mixing model is based on the inverse variance heuristic mixing [Dru07, Section 6.2.2].

Rule discovery (RD), as displayed in Algorithm 2, can be accomplished by many different methods. While the standard version of SupRB uses an evolution strategy (ES), we also have examined different variations of novelty search [Hei+22a; Hei+23c] (cf. also Section 6.4). A single rule discovery phase generates multiple rules to be added. Therefore, multiple runs of the $(1, \lambda)$-ES are performed independently, enabling parallelization. For each of those, an initial candidate, i.e. parent rule, is placed around a training example which is chosen by applying roulette-wheel selection based on the in-sample error of the current *elitist* (or intermediate solution). Thus, higher probabilities of being selected are assigned to examples where the prediction exhibited a high error. The next step is the creation of λ children by repeatedly applying the non-adaptive mutation operator. It increases the distance of the parent's lower and upper bounds to the center by adding values sampled from a halfnormal distribution to each. The fittest child individual—based on its in-sample error and the matched feature space volume—is selected as the new parent. Additionally, if its fitness is higher than that of the current candidate, it replaces this individual. In rule discovery, the fitness is calculated as

$$F(o_1, o_2) = \frac{(1+\alpha^2)\cdot o_1 \cdot o_2}{\alpha^2 \cdot o_1 + o_2}\,, \tag{4.2}$$

with

$$o_1 = \text{PACC} = \exp(-\text{MSE}\cdot\beta)\,, \tag{4.3}$$

and

$$o_2 = V = \prod_i \frac{u_i - l_i}{\min_{x\in\mathcal{X}} x_i - \max_{x\in\mathcal{X}} x_i}\,. \tag{4.4}$$

This metric (cf. Equation (4.2)) was adapted from [Wu+19], who utilized it to combine two objectives for a feature selection problem. The Pseudo-Accuracy (PACC), Equation (4.3), compresses the Mean Squared Error (MSE) of a rule's prediction into a $(0,\ 1]$ range, while the volume share $V \in [0, 1]$ (cf. Equation (4.4)) of its bounds is used as a generality measure. The parameter β controls the slope of the PACC and α weighs the importance of the objectives o_1 and o_2 against each other. We opted for a scalarization approach to solve this multi-objective optimization problem to keep the optimizer simpler and more straightforward. Especially with RD, we do not seek the globally best rule but a diverse set of options but we also do not really need to approximate the Pareto front, as any decision on choosing from the front automatically would in turn be based on some a-priori set weights, so we can just optimize based on this weight directly. Multiple values for β were tested and we found $\beta \in [1,\ 2]$ to be suitable defaults and selected a beta of 1 for this thesis' runs. For α, 0.05 can be used in many problems but, ultimately, the value should always depend on the model size requirements, which are task dependent. Within the experiments of this thesis, we tune $\alpha \in [0.01,\ 0.2]$. The optimization process terminates if the candidate was not replaced within the last δ generations, adding this specific elitist to the pool. Overall, the process of discovering and adding new rules is repeated until a predefined number of rules is reached. Note that the optimizer does not aim at finding a single globally optimal rule—as would be the case in most optimization scenarios—but the goal is to find many optimally placed rules that allow a more accurate prediction for all inputs than a trivial model, i.e. simply returning the mean of all data, would produce. To this end, independent evolution is advantageous.

The second phase, solution composition (SC), utilizes a genetic algorithm (GA) to select the subset of rules from the pool to form a new solution. Naturally, this phase can also utilize any optimizer, and in a previous experiment in [Wur+22] we compared several options and found that the GA is an appropriate choice. An extended version of this experiment is found in Section 6.5. For indicating whether a rule from the pool is included in the solution, a bit string representation is used. The pool is treated as a list with bits corresponding to specific indices. For the GA, pairs of parents are selected with tournament selection before being recombined utilizing n-point crossover, where the crossover probability has a default setting of 90%. The resulting children are mutated by probabilistically flipping bits, controlled by the

Algorithm 3 SupRB's Solution Composition [Hei+22b]

```
 1: procedure COMPOSE SOLUTION(pool, elitist)
 2:     population ← elitist
 3:     for $i \leftarrow 1, \text{pop\_size}$ do
 4:         population ← population ∪ INIT SOLUTION()
 5:     end for
 6:     for $i \leftarrow 1, \text{generations}$ do
 7:         elitists ← SELECT ELITISTS(population)
 8:         parents ← TOURNAMENT SELECTION(population)
 9:         children ← CROSSOVER(parents)          ▷ 90% probability n-point
10:         population ← MUTATE(children)          ▷ probabilistic bitflip
11:         population ← population ∪ elitists
12:     end for
13:     return best solution from population
14: end procedure
```

mutation rate. For the next generation, the children and some of the fittest parents (*elitism*) are kept. While the number of elitists depends on the population size, we found that, in our experiments, 5 or 6 elitists are most suitable for a population with 32 individuals. The fitness is also calculated based on Equation (4.2). In this case, the objectives are the solution's in-sample mean squared error and the number of rules selected (the *complexity*). Altogether, each individual corresponds to a subset of the pool. Therefore, the effective goal of SC is to find the composition of rules which offers the best balance (according to α in Equation (4.2)) between an accurate and a compact model, discarding all those subsets of rules which are Pareto-dominated. We still opted for a scalarization of objectives rather than approximating the Pareto front during training as we want to use one elitist to inform the search in RD and would thus have to choose from the front based on some weights anyhow. The choice of $\alpha = 0.3$ was adopted from [Wu+19] and observed to allow selecting few rules for easy and many for harder problems, all while maintaining a sensible error. However, the influence of α is something that should be investigated in more detail in future research. In contrast to other rule-based learning systems, especially batch multi-solution (Pittsburgh-style) LCSs, and attributable to the two phases of rule discovery and solution composition, rules in the pool remain unchanged by the optimizer of the SC phase and only rules within the pool can form a solution candidate.

In general, SupRB is intended—and therefore conceptualized and designed—as a regressor. This is reflected in the system's description, as well as the experimental setup and evaluation of the next chapters. Nevertheless, adapting the system so that it is able to solve classification problems is quite straightforward: An appropriate classifier, e.g. a constant model, logistic regression, or any more complex model (depending on the explainability requirements), can replace the linear local models. In addition, the fitness function needs to be based on accuracy (or an appropriate metric should the data be imbalanced) instead of PACC (and therefore MSE).

4.2 Computational Complexity

The computational complexity is a key quality feature of any algorithm. While compute might become cheaper and more accessible with time, it is still very prudent to design algorithms in a way that they operate resource friendly. In this section, an attempt at theoretically analysing SupRB's computational complexity will be made. However, it should be stressed that—due to their highly stochastic nature—it is difficult to predict evolutionary algorithms' (and other metaheuristics') computational requirements convincingly. Regardless, their main influences will be discussed as well.

At the heart of training with SupRB is the model fitting component (cf. Section 2.1.3 for what this entails for LCSs). As discussed in Section 4.1, SupRB is currently performing regression using linear local models and uses a constant mixing weight per rule which is based on the number of training examples matched and the in-sample error:

- The linear local models used in SupRB are trained with Ridge Regression. Ridge Regression as implemented in *scikit-learn* [Ped+11] (further details on SupRB's implementation are found in Section 4.1) has a training complexity of $O(n_{\text{examples}} * n_{\text{features}}^2)$ for $n_{\text{examples}} \geq n_{\text{features}}$.
- The mixing weight is computed in $O(n_{\text{examples}} * n_{\text{features}})$ time. To determine the mixing weight, we need to check each example (so a total of n_{examples}) whether it matches and then make a prediction for each example that does (at worst this is also n_{examples}), where each prediction is linear with the number of input features.

Overall, model fitting will thus cost $O(n_{\text{examples}} * n_{\text{features}}^2)$ per rule. Note that this happens many times during the training of SupRB. Even for models that in the end only have a low model complexity (the number of rules in the model) a large number of rules might have been fitted during the model selection process. Typically, a single rule discovery phase will test hundreds (or more) rule locations and (in the worst case) would need to fit a new local model for each adjustment of the matching function (the rule's condition as represented by intervals). Given how often rule discovery will be repeated (e.g. 32 times in the experiments of Chapter 5), this can quickly amount to high 4 and up to low 5 digit numbers of local models needed to be fitted per training.

This leads to the question of how expensive the model selection process of SupRB is. In SupRB, we decompose this problem into solution composition (SC; combining rules from a pool of rules) and rule discovery (RD; filling the pool with good candidates). For both of these tasks, we use evolutionary algorithms in SupRB, i.e. GA and ES, respectively, in the standard version of SupRB (for more options see Chapter 6). It is very difficult to predict the runtime of such metaheuristic black-box optimizers. Theoretically, with regards to the inputs of SupRB training (a matrix built from, typically multi-dimensional, data points) and under the assumption that model fitting is done (or not factored in) the computational complexity of both SC and RD is $O(1)$. However, this is not really reflective of practical considerations. In general, for real-world data, it is safe to assume that there is a correlation between the number of features in a data set and its complexity to learn. A weaker correlation should exist between the sample size and the complexity, however, this will be less pronounced. Note that of course there are exceptions to this and many large and multi-featured data sets exist which are easier to solve then some with few examples and features. Often, these smaller datasets are hard exactly because too few data points were gathered while for others there might be more than enough.

Much more important than the size/shape of the inputs for RD's and SC's runtime is the complexity of the data (or rather, its underlying correlations and interconnections) itself. A very rugged, high-amplitude (i.e. highly multi-modal with relevant depth of local optima) landscape will make it very difficult to place good linear models into. Thus, we will likely need more complex (utilizing more rules) models which also requires us to discover more individual rules. The problem SC solves scales exponentially as there are $2^{n_{\text{rules}}}$ possible models that can be created from the n_{rules} rules in the pool. While the GA probably scales better, this is an issue to consider. Although we assume that rules in the pool are meaningful, this can still lead to issues down the line.

RD's computational complexity is primarily dependent on its hyperparameters, i.e. how many rules to generate in total, how many generations per search, and how many individuals should be created each generation. Otherwise, its own runtime is clearly $O(1)$ for the training inputs and it is linear with regards to these parameters. Given that in RD each local model is fitted (rather than assuming all possible models are already fit), this leads to a true computational complexity of $O(n_{\text{examples}} * n_{\text{features}}^2)$.

We can gain a real-world speedup of both RD and SC by buffering predictions made by local models for the training data. With that, all known models only need to make the $O(n_{\text{examples}} * n_{\text{features}})$ prediction once, rather than each time that they are called. Especially for the time SC will require to compute, this speed-up can be significant if the lookup used is efficient.

Predictions on new data points, e.g. from test data, made by a model trained using SupRB are much more straightforward. These will always cost $O(n_{\text{examples}} * n_{\text{features}} * K)$ if we consider the size of our model K (see also Equation (2.1)) as well. This cost is made up of the computational complexity of each of the K rules for matching ($O(n_{\text{features}})$) and prediction ($O(n_{\text{examples}} * n_{\text{features}})$). This makes inference quite fast compared to many state-of-the-art ML models where still hundreds (or millions) of matrix multiplications have to be performed.

Overall, while we can say that training ($O(n_{\text{examples}} * n_{\text{features}}^2)$) and inference ($O(n_{\text{examples}} * n_{\text{features}})$) are scaling well with regard to the input shapes of data, it needs to be stressed that the true runtime is very dependent on the complexity to solve the rule discovery and solution composition tasks whose complexities are in-turn highly learning task dependent.

4.3 Towards More Explainable Models

In Chapter 3, a template on how to determine the explainability requirements for ML models on a case-by-case basis has been discussed and its application demonstrated in a real-world scenario. This section shows how SupRB can be used to fulfil the requirements laid out by the user study in Section 3.5 and how the consequences for model design (Section 3.5.6) can be facilitated by SupRB's approach to model construction. Note that, despite some of this section's examples being often tied to the user study, SupRB is still a general function approximator and can be used for a multitude of other use cases. Therefore, the deliberations made in this section are to be understood in a general sense. SupRB's clear focus remains on constructing accurate yet explainable models for any situations where this is a relevant feat and SupRB is not specific to the use case presented in Section 3.5.

4.3.1 SupRB's Training

In our stakeholder study, we found that how SupRB (or another LCS) trains its models is of secondary (or even lower) importance to the interviewees and their peers. This is insofar beneficial as it is, at least currently, very hard to achieve explainability of evolutionary (or other black-box) optimization [cf. e.g. Bac+22; Zho+24; SHH23]. While we can generally explain relatively well how these optimizers work on a surface level due to their underlying metaphor(s), these can also be misleading as their biological accuracy is somewhat limited and it is usually advisable to think outside of the metaphors [Ara+22; CA23]. In the future, we hope to improve this aspect specifically for SupRB and maybe to derive techniques to explain to operators or other stakeholders why a rule looks the way it does. But for now, SupRB does not make its model selection and fitting optimizers explicitly explainable.

I would, however, argue that SupRB's model selection (cf. Section 4.1) is easier to explain than that of the most popular LCS, the XCS classifier system (XCS).[4] In XCS, a *steady-state evolutionary algorithm (EA)* introduces new rules by slightly adjusting existing rules from the action set (the set of matching rules that proposed the value given as the system prediction) by using crossover and mutation. These new rules are added to the population up to a certain maximum size. If that size is reached a *deletion strategy*, e.g. fully random, fitness-proportionately random, age-proportionately random, highest age, lowest experience, lowest fitness, or others, determines some rules from the population to be deleted. Another mechanism that adds new rules is *covering*. If an input example is matched by few (or no) rules or only low fitness rules that mechanism creates a new rule (or potentially more rules) which matches that example. This rule is added in the same way as the EA's new rules followed by the same deletion strategy if needed. In addition to these influences on the population, there are mechanisms like *subsumption* (only adding rules if there is no rule with the same "action" that is strictly more general; note that there are two fully independent subsumption mechanisms in XCS that follow the same goal) [Wil98] and *absumption* (counteracting very general rules to reduce

[4]Note that despite it regularly being employed for (supervised) batch learning tasks [e.g. Ste19] XCS is originally designed for online (reinforcement) learning which obviously has an impact on its comprehensibility as it has some—albeit arguably convoluted—mechanics to enable the example-wise input and to avoid getting stuck in the environment. So this is less to say that XCS was ill-designed on account of its model selection process but more to say that there was (or is) a fundamental research gap within most LCSs where SupRB does better in—compared to XCS—when it comes to supervised batch learning applications.

competition) [LBX19] which both impose influences on the model selection process. Especially subsumption is essential for XCS's generalization [Ste19], and therefore learning, but is not easy to comprehend when investigating the training process. Finally, the population at the end of training will—at least when supervised offline learning was performed—often also be reduced in size to increase its explainability. *Compaction* [LBX21a] is a mechanism that removes unnecessary rules from the population, e.g. by filtering it according to some criterion. It is discussed various times throughout this thesis, e.g. in Section 6.5. A variety of compaction algorithms has been proposed over the years [LBX21a] but their core is always to remove as many rules as possible without loosing accuracy.

All these algorithms or separate heuristics have some degree of influence on the model selection in XCS. However, they are interacting with each other in a way that makes it difficult for users to grasp what the result of training will be. Even when tracing training step-by-step it is not always easy to understand why certain rules have been chosen or discarded or why they look the way they do. This is largely due to different component interactions but also the interaction of rules with each other, for example due to fitness sharing within niches and the areas of matched space (during subsumption).

By contrast, SupRB's model selection first involves the independent discovery of rules where each rule is discovered without being aware of the existence of other rules and by having its own independent fitness assigned (rather than a fitness based on a certain niche of the input space or a fitness representing the quality of the rule set as a whole). This discovery of rules constitutes Subtask II as defined in Section 2.1.3. In SupRB, only a single mechanism is responsible for adding new rules rather than a multitude of mechanics as in other LCSs. This makes it much easier to trace why a specifically shaped rule ends up in a model (or at least as a candidate for it). However, truly understanding it is still limited due to the explainability limits of evolutionary methods (as discussed above). Afterwards, solution composition selects a subset of rules from the pool of rules, thereby solving Subtask I. This is somewhat similar to compaction as it can be seen as reducing the number of rules but is different in that it is directly driven by a more advanced metaheuristic optimizer rather than a more simplistic heuristic. While the more basic compaction techniques are probably easier to follow than SupRB's GA, the more efficient techniques (e.g. RCR3 [LBX21a] are on-par in difficulty with the GA or might even be more difficult to follow as most stakeholders should be familiar with the concept of iterative optimization even if they don't have a technical or mathematical background.

SupRB's local model fitting (Subtask IV in Section 2.1.3) is quite similar to XCS's in terms of comprehensibility. Where XCS often uses recursive least squares to fit local models example-by-example, SupRB uses linear least squares to perform batch learning on matched data. Linear least squares as an algorithm is well understood and has been around for a long time. Comprehending its process is an issue for all linear models and therefore not directly in the scope of improving explainability of LCSs but it should of course be kept in mind that this could, theoretically, be a question of non-experts.

Neither SupRB nor XCS explicitly fit their mixing weights (Subtask III in Section 2.1.3). XCS's mixing weights are typically tied to its fitness which is niche-based and therefore a rules weight is subject to nearby rules' performance. In contrast, SupRB's mixing weights are defined on rule level independently of other rules as they are based on that rule's training performance and experience. Arguably, that makes it easier to understand why a specific rule outweighs another than it is in XCS because the neighbourhood does not have to be observed as well.

4.3.2 SupRB's Model

Following the results of the user study from Chapter 3 and the lack of discussion in relevant ML literature, i.e. a complete absence in some of the most-discussed current articles in the field [e.g. BP21; LPK21; Bar+20], the training process' explainability is of secondary importance to the extent of the created model's explainability.[5] Accordingly, the model must offer the option to be thoroughly inspected by humans and be as interpretable and transparent as possible. For the usage of SupRB—and indeed other LCSs—we assume cases in which explainability requirements go beyond what universal post-hoc techniques can achieve—especially on complex black-box models such as very deep neural networks—as is confirmed by the stakeholder interviews. Regardless, this does not necessarily contradict the usefulness of techniques such as e.g., *explanations by example*[6], to compliment explanations based on the model's transparent structure. Some stakeholders might specifically request that type of insight on top of deeper model explanations as it relates to how humans explain things [Bar+20] and, yet, others might already be satisfied with this more surface level approach. We can create those examples for almost any type of model. However, with LCS models a complex synthesis may be circumvented in a quite intuitive way[7] as one can select an example from the training/test data that was matched by the rule(s) in question or at least plausible other points within the matched space of a "fit" rule.

Barredo Arrieta et al. [Bar+20] even go so far as stating that post-hoc techniques are not needed with rule-based learners, albeit with some caveats regarding the model structure we have to take into consideration for SupRB as well. Some limits, such as how the data was preprocessed and if users can understand the processes there, fall a bit out of the scope of judging SupRB on its models' explainability but it might of course be relevant to keep that in mind when implementing a SupRB-based predictor. Another model or another internal structure within a SupRB model might need more (or less) preprocessing in the dataset and more (or less) transformations of features, thus, making it harder to understand what is put into the model. So, even if the model itself was easier to follow, the entire process might not be. Other limits

[5]In general, I think it is reasonable to assume that it is more important for the model that is selected for deployment to be explainable than for the training process to be, however, there are probably numerous use cases, e.g. in medicine or in safety- or security-critical situations, where the ability to analyse and comprehend the training is relevant as well.

[6]*Explanations by example* typically involves the synthesis and presentation of data points representative for a result made by a model. This technique answers which data points also lead to this (or a very similar prediction) to give a stakeholder an idea what to look out for in their own data.

[7]In a black box model, we might need to specifically filter training/testing examples for the prediction we want to investigate, e.g. a specific class, or at least a data point with similar—which is non-trivial to define—predictions in case of regression.

imposed on explainability of rule-based learners, according to Barredo Arrieta et al. [Bar+20], include the number of rules and their overall structure (cf. Section 3.1 for a bit more context about how this applies to LCSs in general). For SupRB specifically, we control the number of rules via our two optimization objectives of rule count and model error, attempting to balance them against each other and find a good middle ground on the Pareto-front, while rule structure and mixing (which is not mentioned by Barredo Arrieta et al. [Bar+20] as mixing is rather specific to LCSs and not present in many other rule-based machine learning approaches) are fixed by design before application.

SupRB's rules are kept as simple as possible. Therefore, they only use hyperrectangular conditions and linear models for prediction.[8] Hyperrectangular conditions are easily human readable.[9] However, very large feature spaces (very high-dimensional data) can make it more difficult to mentally process such rules and might make aforementioned feature transformations beneficial. More complex conditions, which are sometimes used in other LCSs ([e.g. BLW08; BO02; BH03; HBL09; IBZ14a]; cf. also Section 6.1), are much harder to interpret and thus undesirable according to our findings from Section 3.5.6 and thus not considered for SupRB. Complex local models, e.g. neural networks as used by Lanzi and Loiacono [LL06], are not considered following the same line of arguments. Rules' bookkeeping parameters like their fitness, error, and experience are independent of other rules and therefore can be examined in a standalone fashion. These choices already keep a trained models' rules rather readable for a human user but even with those readable rules a large number of them can still make an unmanageable model.

The number of rules is controlled indirectly through SupRB's solution composition (SC) process. Before training, the optimal number of rules to achieve a certain error is unknown. Usually, training is expected to achieve the lowest possible error which—if used as a standalone objective—can quickly lead to large numbers of rules being used (bloat), which can also lead to overfitting and is, of course, making the model harder to understand and interpret. Therefore, it is a good choice to balance both objectives against each other and find a middle ground solution. We can reasonably expect that any SupRB model that has finished training (where the training process showed clear convergence) has selected the minimal number of rules from the pool possible to achieve the given error and, thereby, a balance between error and model complexity. This should ensure that SupRB's models are also reasonably well-explainable for the given task at hand. If before training some level of error can be determined as an acceptable threshold, i.e. models better than this are deemed equally good, or a maximum amount of rules is given (or a lower bound at which rule counts are equally good can be defined) this can and should be used in training to not only limit the search space and speed up convergence but also to find more optimally tailored solutions. Such potential bounds are heavily influenced by specific applications and explainability needs but even given a specific situation it is unlikely that stakeholders are able to express these definitively enough.

[8]For regression tasks, constant models are less likely to fit the problem well if only few rules are to be used, but for classification, constant models could be an option to achieve further simplification in case logistic regression or similar approaches are still deemed to complex by or for stakeholders.

[9]An example of a rule trained on a real-world data set is presented in Table 5.4.

Lastly, mixing is an important feature of LCSs and can, of course, make explainability of specific decisions or the model as a whole more complex. This starts by the participation of multiple rules (albeit much fewer than in random forests) to form a prediction, but is also a crucial factor towards understanding how the rules come together. For SupRB, we attempted to make this as easy as possible, especially considering approaches like Mixture of Experts (MoE) models (cf. Section 2.2.2). We used a weighted sum of training error and experience of a rule which is relatively easy to relay to stakeholders when discussing the model: *The highest impact towards a decision should be based on the individual rule that has the most experience while being closest to the correct prediction in the past.* Contrastingly, those individuals which are inexperienced or often incorrect should be given less impact. The weight is constant once a rule has been fitted and is independent of other rules (which is often different in other LCSs). Theoretically, one could fit that mixing model as well, similar to MoE models. However, if we tried to explain these fitted mixing weights (even without going into details about the—likely complex—training process) we found a more challenging task ahead as the weights values would be based on model performance and which samples are matched. Therefore, a fitted weight will likely not be easy to grasp—especially without extensive LCS experience—but also impossible to ignore due to its high impact on the actual prediction of the model. Thus, a weighted sum with weights that were set once based on training data seems like the most feasible approach if explainability is the core goal.

Assuming the active usage of SupRB to make predictions about product quality and the interactions of a machine operator, as defined in Chapter 3, we could create examples in the style of: "*The rules that inform this decision are based on a similar situation encountered by your colleague Mrs Smith on the 25th of March last year. In this situation, the task was to manufacture <product> and she selected configuration <set values> while encountering sensor readings of <environmental and other readings> achieving product quality <quality parameters>.*" The placeholders in this snippet can link to other pages in the UI, open pop-ups, display information on hover/mouse-over, or, in case this is limited to very few values, be directly replaced by the relevant data. The exact method would depend on the overall design of the UI and the intended UX. Personally, I would recommend limiting the data that is displayed directly but make it available on request, ideally including context information like sensor readings before and after the specific data point. In contrast to synthetic examples, this type of example achieves direct relatability on an interpersonal level. While it is of course also possible to demonstrate to users what predictions are made with real examples within complex black box models, with SupRB and other LCSs we can use the structure and transparency of the model to great advantage. Here, we can already limit our explanations to a subset of rules (those that match the example) rather than having to explain the entire model. Additionally, given the low complexity of local models, it is likely easier to observe the interactions within features and make users comprehend predictions. In most cases, SupRB's rules will also be limited to much smaller subsets of the training data that influenced one rule than would be the case for neural networks, for example, where we assume that each weight was influence by every training example. Admittedly, there is a lot of research to be done into the specifics of this. Open questions include but are not limited to:

- *How can we effectively present a rule to a user?* This includes possible visualizations of both conditions and local models but also possibly extends to the list of training data this rule has seen and other aspects. Visualizing n-dimensional data is notoriously difficult for $n \geq 4$ but most interesting applications of these models will involve higher dimensionalities.
- *How do we present multiple rules at once and explain their mixing-based interactions?*
- *How interactive should this interface be?* Should we, e.g., allow "what-if"-scenarios by adjusting sliders or similar?

More questions arise when considering the model as a whole rather than the subset responsible for a given situation. What is needed is stakeholder dependent. Most operators in our scenario from Section 3.5 will probably mostly interact with the model (and therefore desire it to explain its predictions) on a case-by-case basis. Whereas, process engineers and data scientists might want to inspect the model as a whole, process engineers to learn from it and what types of, possibly unknown, interactions it found within the data and data scientists to decide if it is ready for deployment and to gather insights into its likely performance beyond statistical measures on the test data set.

5 Comparing SupRB with other RBML methods

After the previous chapter introduced the basics of SupRB, the following chapter will test its performance as a predictive model while comparing and contrasting it to more established rule-based machine learning (RBML) approaches.[1] The main contents of this chapter have previously been published in the proceedings of the BIOMA conference series [Hei+22b], as well as the Journal of Applied Soft Computing [Hei+23d], where the latter extended the former's comparison of SupRB to the most common LCS for regression tasks, XCSF [Wil02a], by additionally benchmarking against Decision Trees (DTs) and Random Forests (RFs). We selected these three methods as they are relatively widespread RBML approaches that should be the strongest competition for use cases where SupRB can be used effectively. This is also the reason why we do not compare with neural networks. As they are—at least at the moment—not interpretable, they are best suited for different applications as SupRB. We can expect them to show lower errors but unfit for the explainability requirement we assume for the use of SupRB.

We find that, as hypothesised, SupRB performs competitively to XCSF with respect to prediction error across datasets, while producing substantially more compact models directly. SupRB achieves better mean errors than DT while still constructing smaller models and worse errors than RF, albeit, with vastly smaller models than RF.

5.1 Revisiting Explainability in Rule-based Machine Learning

To ease the reader's understanding of the core discussions around the results from the evaluation presented in this chapter, this section reiterates some core concepts around what makes Rule-based Machine Learning (RBML) models explainable that have been discussed in more detail in most of the previous chapters of this thesis.

A well-known representative of LCSs is the XCS Classifier System (XCS) (cf. Chapter 2), which can be utilized to solve all the major learning tasks through its many derivatives and extensions [UM09]. To compare it to SupRB and other rule-based learning systems on a set of supervised regression tasks with real-valued domains, a specific derivative and configuration

[1]As introduced in Chapter 4, where I gave a general description of SupRB, the two chapters form *C3* as defined by Section 1.1 in combination with Chapter 6, where I will discuss possible improvements of the algorithm or its models.

is required. XCSF provides the basis for this, being designed specifically for supervised function approximation by replacing the constant predicted payoff with a linear function [Wil02a]. Additionally, the binary matching function is replaced with interval-based ones to enable applicability on real-valued problem domains [Wil02b]. There are more elaborate variants for replacing matching functions and linear models, which aim at improving the performance, but, at the same time, reduce the overall model transparency [BO02; LL06].

RBML systems and, therefore, also LCSs, are considered transparent or interpretable by design due to their natural relation to human behaviour. This means they do not necessitate the use of complex post-hoc methods, e.g. visualization or model transformation, to be explainable to humans. There are, however, factors that can reduce the capabilities of these models, even when they are inherently transparent. These include for LCS, among others, the applied encodings, the number of rules in the model and the complexity resulting from the complicated matching functions or local models in the individual rules [Bar+20]. For a more in-depth view, see also the previous discussions on this in Chapter 3 and Section 4.3.2.

While it is common to control these limitations in LCSs by-design (cf. Chapter 4 for the case of SupRB), it is also possible to utilize specialized post-hoc methods. Especially visualization techniques for rules, but also other strategies, aim at improving the model's interpretability [UGM12; LBX19; LBX21b]. Nevertheless, controlling interpretability by-design can be advantageous, as it does not require to devise designated post-hoc methods or adapt existing ones to the specific needs of the problem at hand, as well as the model itself, which requires not only time but also sufficient expertise. The design must therefore focus on understandable matching functions and local models, but still keep the predictive power acceptable. Other factors, for example problem-dependent complex variables/features, can reduce the interpretability and are additionally hard to influence, which is why a thoroughly thought through design is important.

The interpretability of LCS and of rule-based learning systems in general is also highly dependent on the resulting rule set sizes. For example, visual inspections are much easier for smaller sets and the extensiveness of subsequent analysis is reduced. For LCSs, the control of the rule set size is different for batch multi-solution (Pittsburgh-style) and online single-solution (Michigan-style) systems. In batch multi-solution LCSs, the optimization algorithm's fitness function often combines several objectives, most commonly the accuracy and the number of rules, therefore enabling rule set size control. In GAssist [Bac04], for example, the accuracy and the minimum description length are combined into a single objective, while applying an additional penalty for rule sets with too few rules. For online single-solution LCSs, large populations are generally beneficial during the training process and therefore they do not control the rule set size using the fitness function. However, during training, two rules can be merged if one fully encloses the other, a mechanism which is called subsumption. Additionally, compaction can be invoked as a post-hoc method to decrease the number of rules in the set [Wil02b; LBX21a]. It removes redundant rules, i.e. ones whose deletion does not decrease prediction accuracy. Compaction methods are, however, commonly designed for classification tasks only.

In contrast to LCSs, Decision Trees (DTs)—the most well known RBML method—are composed of a set of non-overlapping rules, whereas in LCSs rules may overlap which enables a higher degree of generalization. Models based on multiple DTs, such as Random Forests (RFs) using bagging or XGBoost using boosting, are typically less interpretable than DTs and even LCSs. LCSs are build from overlapping rules, while RFs and XGBoost models are build from aggregations of trees which are in turn build from non-overlapping rules. For an extensive discussion on how the concepts of bagging, boosting, and mixing relate to and differ from LCSs see [Hei+23a] and Chapter 2.

5.2 Evaluation

To evaluate SupRB as described in Section 4.1, we compare it to a recent implementation of XCSF[2] [Wil02a; PP21] with hyperrectangular conditions and linear local models (trained by recursive least squares updates [Lan+06]). This configuration very closely corresponds to the conditions and local models used in SupRB. While we want to specifically acknowledge that some better-performing conditions, e.g. hyperellipsoids [But05], have been proposed for XCSF, we consider them substantially less interpretable for average users, which worsens with increasing input space dimension. We expect XCSF to perform somewhat on-par with SupRB. Usually, errors should be lower while the number of rules in the model (after model compression/compaction) is higher. Additionally, we compare SupRB (and XCSF) with Decision Trees (DTs) and Random Forests (RFs)[3]. As LCSs can be considered as one option to generalize DTs (by dropping the assumption of non-overlapping rules (cf. [Hei+23a] and Chapter 2)), a comparison to them is very intuitive. We hypothesize that DTs show worse errors than the other ML approaches while producing somewhat easier-to-interpret models. Their interpretability is strongly supported by rules not overlapping. However, this usually also makes for less precisely approximated functions (assuming equal number of rules before mixing), increasing the error of the overall model. RFs combine a multitude of trees which, at first glance, might look similar to LCSs since these trees are mixed together in order to produce predictions, whereas, in LCSs, *rules* are mixed. As each individual tree that is part of the RF consists of rules that fully cover the input space, predictions are performed by mixing multiple rules (exactly one rule per tree since there are no overlapping rules within each tree). However, due to how RF are trained, the training steps that lead to these rules are much harder to trace. Furthermore, RFs typically use large numbers of trees, resulting in mixing many more rules than we expect to see mixed for each prediction in SupRB. We hypothesize that their errors will be lower than SupRB's due to higher model complexities being allowed and, probably, also lower than XCSF's. The interpretation of the models generated by RFs, on the other hand, can be expected to be more difficult due to the high number of interacting trees.

[2]https://github.com/rpreen/xcsf https://doi.org/10.5281/zenodo.5806708

[3]We use them as implemented in the Python package *scikit-learn* [Ped+11] v1.0.1.

5.2.1 Experiment Design

SupRB is implemented[4] in Python 3.9, adhering to *scikit-learn* [Ped+11] conventions. We standardize the target and transform input features into the range $[-1, 1]$. These transformations improve SupRB's training process as they remove the need to tune error coefficients in fitness calculations, i.e. balancing complexity versus error for each dataset, and help prevent rules from being placed in regions where no samples exist (and therefore none would be matched). While other optimizers have been proposed for rule discovery (RD) [Hei+22a] and solution composition (SC) [Wur+22] in SupRB, the results have largely been inconclusive, with ES and GA, respectively, probably being a good (or even the best) choice for the optimizers. Therefore, the experiments of this chapter[5] , which correspond to [Hei+23d], follow the setup of [Hei+22b] on which they were originally based and use these optimizers as well as the remaining general configuration[6]: We assume that for the datasets under investigation discovering a total of 128 rules is sufficient to compose good models from. Hence, 32 cycles of alternating rule discovery and solution composition are performed, generating 4 rules per cycle. For the ES we selected a λ (size of the offspring population) of 20. The GA performs 32 iterations with a population size of 32. In general, all of these parameters can be chosen quite freely as long as they are "large enough" (facilitating convergence), with larger values being neither considerably beneficial nor detrimental to model performance but may impose considerable additional runtime cost during training. There is a clear and observable difference in regards to learning tasks, although they do not call for specific tuning. In future work, where new tasks will be investigated, which are likely more complex in function type, number of data points, dimensionality, et cetera, they could need to be increased for better performance. To tune some of the more sensitive parameters, we perform a hyperparameter search using a Tree-structured Parzen Estimator in the Optuna framework [Aki+19] that optimizes average solution fitness on 4-fold cross-validation. We tune the datasets independently for 1000 iterations or a maximum of 360 core hours per tuning process. For XCSF we follow the same process, selecting typical default values[7] [PP21] and tuning the remaining parameters independently on the datasets. This same setup is also used for DT and RF.

The final evaluation, for which we report results in Section 5.2.2, uses 8-split Monte-Carlo cross-validation, each with 25 % of samples reserved as a validation set. Each learning algorithm is evaluated with 8 different random seeds for each 8-split cross-validation, resulting in a total of 64 runs per algorithm per dataset.

We evaluate the four approaches on datasets which are taken from the UCI Machine Learning Repository [DG17]. An overview of these is presented in Table 5.1. The Combined Cycle Power Plant (CCPP) [KT12; Tüf14] dataset shows an almost linear relation between features

[4]The newest version is always found at https://github.com/heidmic/suprb. The version used for all experiments in this dissertation is long-term archived at https://doi.org/10.5281/zenodo.14181292.

[5]All scripts to reproduce the results of all experiments within this thesis—including the other chapters—can be found at https://github.com/heidmic/suprb-experimentation.

[6]Note that the results of [Hei+22b] were chronologically achieved first, even if they saw final publication later than [Wur+22] and [Hei+22a] due to different publication cycles.

[7]https://github.com/rpreen/xcsf/wiki/Python-Library-Usage

Table 5.1: Overview of regression datasets SupRB, XCSF, DT, and RF are compared on.

Name (Abbreviation)	n_{dim}	n_{sample}
Combined Cycle Power Plant (CCPP) [KT12; Tüf14]	4	9568
Airfoil Self-Noise (ASN) [BPM89]	5	1503
Concrete Strength (CS) [Yeh98]	8	1030
Energy Efficiency Cooling (EEC) [TX12]	8	768

and targets and can be acceptably accurately predicted using a single rule. Airfoil Self-Noise (ASN) [BPM89] and Concrete Strength (CS) [Yeh98] are both highly non-linear and will likely need more rules to predict the target sufficiently. The CS dataset has more input features than ASN but is easier to predict overall. Energy Efficiency Cooling (EEC) [TX12] is another rather linear dataset, but has a much higher input features to samples ratio compared to CCPP. It should similarly be possible to model it using only few rules.

5.2.2 Results

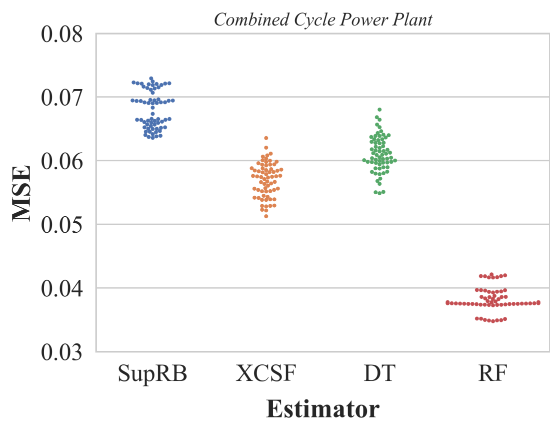
In this section, we present and discuss the results of our experiments. We first focus on MSEs, then discuss solution complexities, and close with a rigorous statistical analysis.

Mean Squared Errors

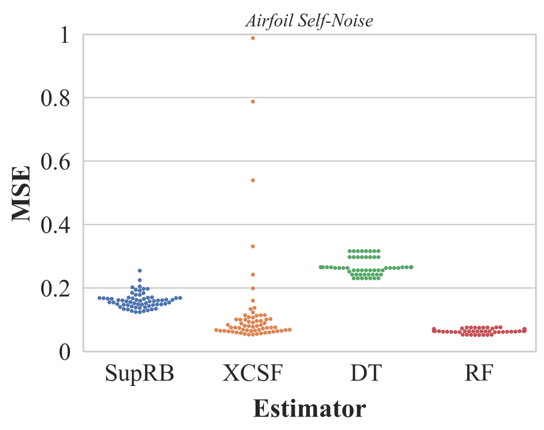
Table 5.2: Overview of the experimental test data results of 64 runs per dataset rounded to two decimals. The columns give the mean error and its standard deviation over the runs per algorithm. The errors are calculated in a standardized target space, meaning the data was transformed to zero mean and unit variance beforehand. If the data is normally distributed and the sample size is large enough, then a very trivial model that predicts the mean of the output training data for any input is able to achieve an MSE of about 1 in standardized target space. Highlighted in bold are the models where the mean performance was stronger than the other models'.

	CCPP	ASN	CS	EEC
SupRB	0.07 ±0.00	0.16 ±0.02	0.14 ±0.03	0.03 ±0.01
XCSF	0.06 ±0.00	0.12 ±0.16	0.17 ±0.13	**0.02 ±0.02**
DT	0.06 ±0.00	0.27 ±0.03	0.17 ±0.03	0.08 ±0.02
RF	**0.04 ±0.00**	**0.06 ±0.01**	**0.09 ±0.02**	0.04 ±0.01

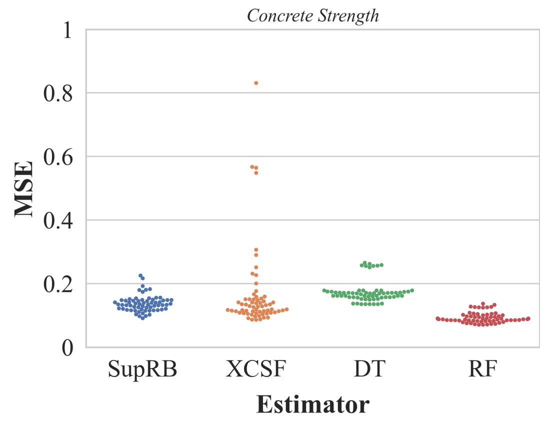
In our experiments, we find that—on average—XCSF performs slightly better than SupRB, which in turn performs slightly better than DT with regards to mean squared errors (MSEs). RF is competitive or better on all datasets and does show the lowest variance over the different



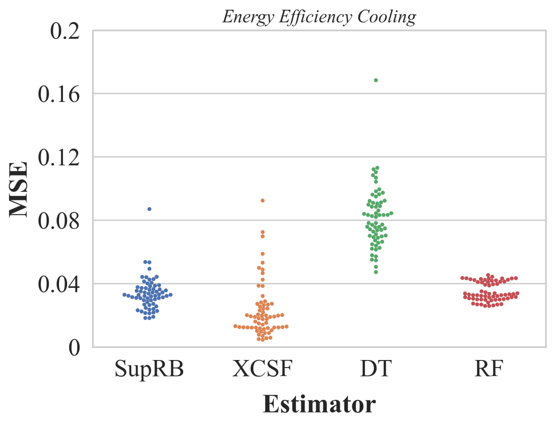
(a) Distribution of runs on CCPP



(b) Distribution of runs on ASN



(c) Distribution of runs on CS



(d) Distribution of runs on EEC

Figure 5.1: Distribution of runs' errors. All datasets are standardized with unit variance, therefore, a trivial model should achieve an error of about 1.0. Note that the plots are on different scales, reflecting the varying difficulty of the learning tasks.

runs. This follows our previous hypotheses. While RF is performing worse than XCSF and SupRB on EEC and—at least visually—vastly better than all on CCPP (cf. Figure 5.1a), we want to stress the very low difference in actual value of the runs on these two datasets, which is likely not practically relevant. Plots of all runs are found in Figure 5.1, while we report mean (across runs) MSEs and their standard deviations in Table 5.2[8]. The presented results are based on the standardized (zero mean and unit variance) versions of the targets of the respective datasets, facilitating an easier across-dataset comparison. However, a comparison with results from other publications that used the same datasets can be more difficult since oftentimes met-

[8]Note that the presented values (taken from [Hei+23d]) marginally differ from those in [Hei+22b]. We did adjust some of the available hyperparameter ranges for our hyperparameter tuning between these two articles. For SupRB, the differences were very small, only visible on ASN with the given two decimal precision. However, XCSF did perform slightly better, falling less frequently into cover-delete cycles (a common problem with XCS-based systems, cf. [SB10]).

rics are reported on non-standardized-outputs scales as they are made with different intentions. As we do not attempt to find the overall best solution to those specific learning tasks but want to compare and contrast SupRB to three other RBML techniques, we deem this an appropriate compromise.

Analysing the distributions of errors more closely, we find that the algorithms do show different behaviours. Clearly, runs are not actually normally distributed for any algorithm. Presumably, the fact that small real-world datasets are used in this analysis has an influence on this behaviour, but the algorithms do also not converge towards the exact same model per split. No clear relations between random seeds or data splits and errors can be seen from the experiment data itself.

SupRB, XCSF, and DT all did show at least one outlier, although, for XCSF, these were much further away and substantially more frequent. Cover-delete-cycles, a known issue in online single-solution LCSs where experienced rules are removed from the population by error and new rules are only shortly part of the population [SB10], seem to have affected many of XCSF's runs. In online single-solution LCSs, covering is a rule generating mechanism that is triggered when few rules match a given input and then creates a new rule that is guaranteed to match this input. The deletion mechanism of XCSF randomly, but fitness-proportionately, removes rules when the population has reached its hyperparameter-imposed maximum population size. During tuning, we allowed our tuner to choose both the number of training steps and the maximum population size (among the many other parameters of XCSF). Regardless, we find that post-training population sizes almost always reach the population size limit (not accounting for numerosity). The tuner seems to have found settings that counteract the cover-delete-cycle somewhat well. We find that XCSF's hyperparameter tuning chose much larger populations than what would be expected following the typical rule of thumb of using ten times as many rules as should be needed for a good problem solution (based on domain knowledge or prior modelling experience) [UB17]. Upon further investigation, we also found that rules contained in the final models were usually only discovered late during training, although the system error had remained stagnant (at least not showing practically significant changes) for quite some time beforehand. Subsumption in the EA—a typical method to combat cover-delete-cycles—was utilized. This important mechanism checks whether a newly discovered rule is fully contained within a parent rule, in which case the new rule is not added to the population and the respective parent's numerosity parameter gets increased by one. A numerosity of n means that this rule is in the population n times and thus counts as n simple rules (with a numerosity of 1; also known as micro-classifiers rather than macro-classifiers which have a numerosity > 1) towards the population size limit. Numerosity is also used in the fitness-proportionate delete mechanism. A rule selected for deletion gets its numerosity reduced by one and is only fully removed if that parameter hits 0. Theoretically, subsumption is thus decreasing the number of macro-classifiers in the population. Regardless, it seems that in our runs cover-delete-cycles could not be prevented as we found most rules contained in the final model have only been part of the population for relatively short amounts of time and mostly had numerosities of 1.

Overall, SupRB's training seems to converge more reliably than that of the other LCS, XCSF. Some variance based on random seeds is strongly expected, as this is a common occurrence in (evolutionary) stochastic optimization, cf. e.g. [Ver+22]. RF—a non-evolutionary method—suffers less from this phenomenon, however, the overall variance, especially within one data split, is not substantially different from SupRB (and DT) when practical significance gets taken into account as well.

Model Complexity

Table 5.3: Overview of the solution complexities (number of rules in the solution proposed by SupRB, the final macro-classifier count in an XCSF population after compaction and the number of leaf nodes in a DT, respectively) across 64 runs per dataset rounded to two decimals. We report mean and standard deviation. Note that we omit RF here, which used 194, 228, 91, and 122 trees of—on average—similar sizes to the individual DT models, respectively, and is therefore vastly out of scale in terms of number of rules.

	CCPP	ASN	CS	EEC
SupRB	2.97 ±0.67	32.67 ±2.8	28.47 ±2.68	11.84 ±1.85
XCSF	1922.71 ±390.7	1617.58 ±413.12	481.68 ±336.33	707.94 ±282.16
DT	556.44 ±18.38	67.92 ±2.59	110.62 ±6.37	26.50 ±0.71

A main concern of SupRB is to produce (better) explainable models while maintaining competitive results in terms of error. Figure 5.1 and Table 5.2 demonstrated that SupRB is able to learn from complex datasets similarly well as established methods. Table 5.3 gives the model sizes[9] of SupRB, XCSF, and DT. We report their mean over 64 runs and their standard deviation. Model size (number of rules) is one of the key parameters that hinders interpretability and therefore explainability of RBML models [Bar+20].

Due to the frequent cover-delete-cycles discussed extensively in the previous section, XCSF's trained models are usually very close to the population size limit set during tuning. We then perform compaction (cf. Section 5.1) to remove as many rules as possible, which resulted in about 10% of rules being removed. However, the final model size still far exceeds the one of SupRB. Admittedly, XCSF training does not explicitly promote small macro-classifier counts; however, many mechanisms are present that should reduce it (in particular: set subsumption, EA subsumption, and, most importantly, compaction).

DT did show a tendency towards surprisingly large and complex trees for CCPP. Its models for ASN and EEC were only twice the size of SupRBs, which is easily explained by LCSs ability to have overlapping rules which DTs lack. However, the errors of DT on these two datasets

[9]Note that the presented values differ from those in [Hei+22b]. These differences are the result of adjusting some of the available hyperparameter ranges for our tuning. For SupRB, this results in larger models, while, for XCSF, it reduced model size compared with [Hei+22b].

were also notably higher than SupRB's. Rule overlaps could explain the roughly four times as large model for CS as well, where DT performed only slightly worse than SupRB in terms of error. For the training of DTs we used a slightly modified version of the well-known CART algorithm, as implemented in scikit-learn [Ped+11]. This algorithm controls tree size on the basis of some configurable hyperparameters like maximum depth and minimal sample count in a leaf node and does also feature a pruning mechanism that balances prediction error with tree complexity directly.

For RF, we tuned the number of trees to be bagged, which resulted in 194, 228, 91, and 122 trees for CCPP, ASN, CS, and EEC, respectively. The individual trees were of roughly the same size as the ones used by DT, making RF's models substantially larger than those of the other three algorithms'. Models of that size are theoretically interpretable but practically it is not realistic to gain substantial insights in an appropriate time. SupRB's models can, however, be read rather easily and analysed even without very deep LCS knowledge. Our results on the complexity metric fit with our hypotheses made at the beginning of Section 5.2.

Model Inspection

Table 5.4: Exemplary rule generated by SupRB on CS dataset. The target is the concrete compressive strength in MPa. The original space intervals denote the area matched by the rule in terms of the original variable scales, while the intervals in feature spaces are scaled into $[-1, 1]$ and help perceiving rule generality at a glance. Coefficients denote the weight vector used for the linear model. This rule was also presented in our previous study [Hei+22b].

	Original Space	Feature Space σ	
input variable	interval	interval	coef
Cement [kg/m^3]	$[104.72, 516.78]$	$[-0.99, 0.89]$	2.38
Blast Furnace Slag [kg/m^3]	$[0, 359.40]$	$[-1.00, 1.00]$	2.29
Fly Ash [kg/m^3]	$[13.45, 200]$	$[-0.87, 1.00]$	0.68
Water [kg/m^3]	$[122.64, 244.80]$	$[-0.99, 0.96]$	-1.26
Superplasticizer [kg/m^3]	$[6.02, 24.80]$	$[-0.63, 0.54]$	-0.67
Coarse Aggregate [kg/m^3]	$[950.16, 1145]$	$[-0.13, 1.00]$	0.71
Fine Aggregate [kg/m^3]	$[756.14, 992.60]$	$[-0.19, 1.00]$	0.60
Age [days]	$[18.36, 365]$	$[-0.90, 1.00]$	2.07
		intercept$_\sigma = 3.9160$	
In-sample MSE$_{\text{orig}}$ 1.5310	In-sample MSE$_\sigma$ 0.0917	Experience 84	

An example rule extracted from a model trained on the CS dataset is presented in Table 5.4. The data set consists of eight input features (and a target). For each dimension of the input, Table 5.4 gives the name (and SI unit), what portion of that dimension is matched, and how influential the value in a dimension is towards our target (denoted by the coefficient of the localized linear model). For the matching interval (we always match hyperrectangular subspaces), Table 5.4 contains the absolute (original/unscaled) and the normalized (min-max-scaled features) bounds. The normalized bounds are what SupRB operated on and make it easier to grasp what part of the populated (with samples) space is being matched, whereas the original space representation helps at understanding proportions and relates stronger to the real world. While we think that both representations are important for XAI, we can assume that algorithmic behaviour is better understood/analysed on the normalized space, while the original space is more helpful for domain experts tasked to operate with the rule's predictions. Additionally, Table 5.4 gives the constant intercept of the linear local model and the in-sample (encountered during training) error of the local model, as well as how many examples from the training data were matched and used for its training.

During training, the rule in Table 5.4 matched 84 examples (its experience is 84) and it did match another 31 examples from our test set. The model we extracted it from consisted of a total of 23 rules, with experiences of 7 to 240. Rules were either rather general or rather specific. The presented rule is more general than many others.

Close inspection reveals that it matches most of the available input space for five out of CS's eight dimensions. It is even maximally general for the "Blast Furnace Slag" input variable. For the "Superplasticizer" input variable, the rule matches slightly more than half of the possible values and is roughly positioned around the center of the range of values, while for "Coarse Aggregate" and "Fine Aggregate", the rule matches 55 to 60 % of the input range, oriented towards higher values. Note that despite these seemingly large intervals per dimension, the rule only matched about 16% of the entire feature space.

The linear local model's parameters are also based on the transformed feature space and the standardized version of the target (in this case, the concrete's compressive strength). While higher concentrations/values of "Cement", "Blash Furnace Slag", and "Age" of the mixture influence its compressive strength positively, high concentrations of "Superplasticizer" and "Water" have negative effects on this target. The other three variables have lower effects on the target but all seem to positively influence compressive strength of the concrete mixture in the matched areas. Note that this last distinction is very important and actually the great strength of a rule set learning approach. We are able to assign different parts of the inputs space different coefficients for the same features and keep that in an easily readable format.

One of the rule's matched examples is $x = (190.3, 0, 125.2, 161.9, 9.9, 1088.1, 802.6, 56)^T$ with an original $y = 38.56$. For this example, the rule proposes $\hat{y} = 38.39$. The prediction is clearly close to the ground truth. Providing this example to a user with domain experience can help them assess the model's predictive power (cf. Section 4.3.2). Moreover, the user can then extrapolate to other examples that would fit in the provided matching intervals, increasing their trust over other models, where such extrapolation is not possible from the models' structure.

The rule performed slightly below average compared to the other rules in this solution (mean in-sample MSE of 0.0751) on its training data, although this should be viewed critically as more specific rules tend to be able to fit their (noisy and non-linear) input data more easily. We find that rule inspection can provide critical insights into the model's predictive process and is easy to do due to the rule's design and the overall small number of rules per model.

Statistical Analysis

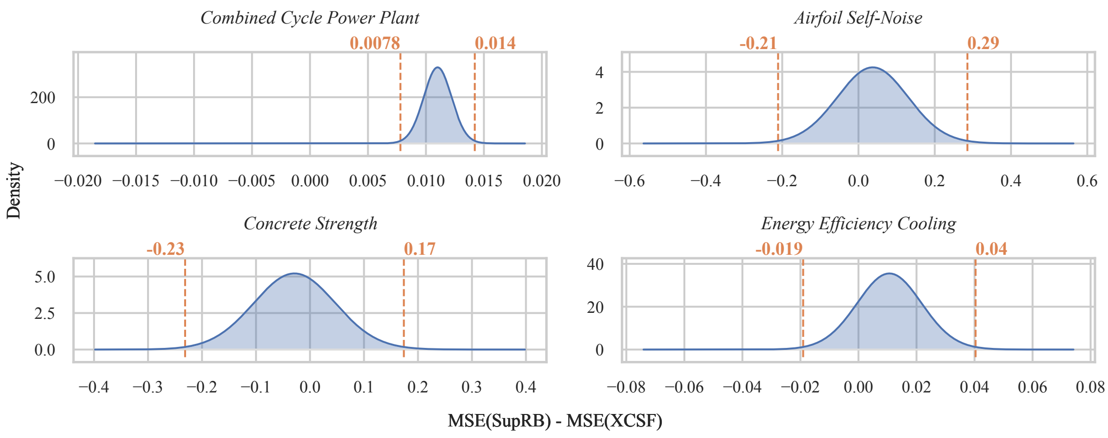


Figure 5.2: Density plot of the posterior distribution obtained from Corani and Benavoli's Bayesian correlated t-test [CB15] applied to the difference in MSE between SupRB and XCSF. Orange dashed lines and numbers indicate the 99 % HPDI (i.e. 99 % of probability mass lies within these bounds). HPDI bounds rounded to two significant figures. Effectively, this indicates how likely a specific difference between one run of each RBML algorithm will be (practically, probability mass above zero equates to SupRB having a higher (worse) value than XCSF).

In this section, we compare SupRB with the three established RBML algorithms using a Bayesian model comparison approach. For this, we apply Corani and Benavoli's Bayesian correlated t-test [CB15][10] to determine probability distributions over the differences in performance between the algorithms.[11] Despite their remaining popularity, we deliberately avoid the use of null-hypothesis significance testing due to their many flaws and possible pitfalls [e.g. Ben+17]. The resulting posteriors (given in Figures 5.2, 5.3, and 5.4 for MSEs including 99 % high posterior density intervals (HPDIs)) are the *distribution of the difference between the considered metric for SupRB and the considered metric for the other algorithm* (practically, this equates to values above zero indicating SupRB having a higher (worse) metric value than the other algorithm).

Figure 5.2 compares SupRB's MSEs with those of XCSF. We see that in 99% of runs, we should expect SupRB to perform worse on CCPP and almost even on the other three datasets. It should be noted that on CCPP the actual difference in expected errors is quite small and a difference

[10]We use the implementation provided by https://github.com/dpaetzel/cmpbayes.

[11]Note that the results differ from [Hei+23d] due to the fix of an—at the time of writing that article undiscovered—bug in the underlying library, cf. https://github.com/dpaetzel/cmpbayes/pull/1.

in this range is unlikely to be practically significant. For ASN and CS, the median difference is expected to be very close to zero with XCSF and SupRB each having a slight edge on one of the datasets. The distributions are comparatively wide in terms of absolute errors indicating that, when performing piecewise comparisons of individual runs of the two LCSs, some combinations can have vastly different errors despite the majority of combinations likely yielding small differences. For EEC, the distribution does look similarly wide, however, it should be noted that this is on a scale one order of magnitude smaller. XCSF might have a small edge in the case of some EEC runs, however, it is doubtful that this difference is practically significant. As explained, the practical significance of differences in expected performance is likely quite low and we assume that, in terms of error, XCSF and SupRB should be considered to perform somewhat equally or at least that no clear decision can be made. In terms of model complexity, the result is very clear that SupRB performs considerably better and does not require further statistical analysis.

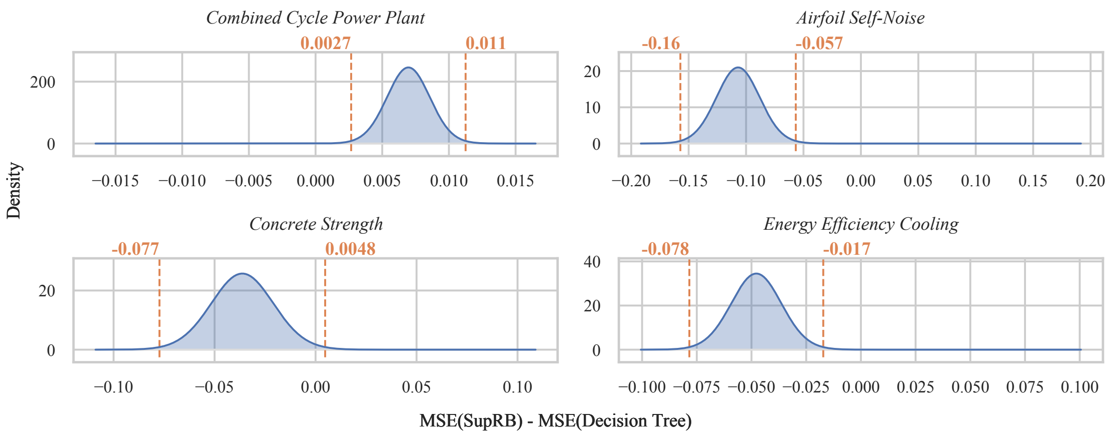


Figure 5.3: Density plot of the posterior distribution similar to Figure 5.2 but between SupRB and DT.

In Figure 5.3, we compare SupRB's MSEs with those of DT. DT might perform very slightly better on CCPP, although this difference would again likely not be practically significant. On the other three datasets, SupRB expectedly will perform better with smaller distributions of differences than against XCSF. For ASN and EEC, we expect that more than 99% of runs will have SupRB showing better errors. For CS, only a very small amount of probability mass favours the DT's runs. As with XCSF, the model complexity analysis clearly shows SupRB to return models with substantially less rules but with the caveat of DT's rules not overlapping, making it slightly easier to explain.

The distributions in Figure 5.4 show the expected distribution of differences in MSE between SupRB and RF. RF performs better than SupRB on CCPP, ASN, and CS, and does so rather consistently. For EEC, the test expects SupRB to perform—at most—very slightly better, however, this is clearly not a practically significant difference and I would argue that the result of the test puts SupRB and the RF at equal performance for EEC.

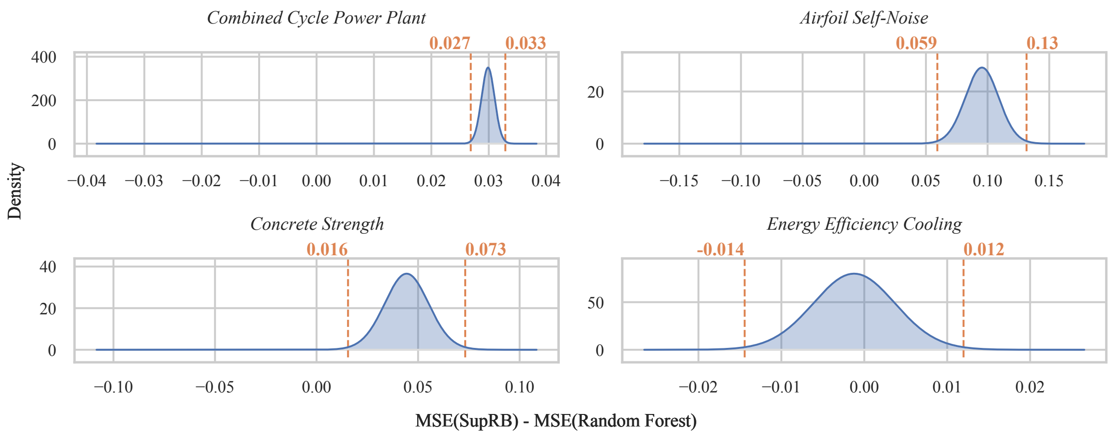


Figure 5.4: Density plot of the posterior distribution similar to Figure 5.2 but between SupRB and RF.

Overall, we can conclude from our tests that, on MSEs, SupRB is probably outperformed by RF or performs equally to it, outperforms DT, and is only slightly outperformed by XCSF in the majority of cases. This fits well with our hypotheses outlined at the beginning of Section 5.2. In terms of model complexities, SupRB does manage to produce substantially smaller and more interpretable models without loosing too much predictive power.

5.3 Summary of the Experiment

This chapter benchmarked SupRB and compared and contrasted it to more established RBML algorithms. It is based on [Hei+23d], which itself is an extension of [Hei+22b]. In those publications, we depicted how SupRB, as a novel RBML algorithm that uses two separate optimizers to place and select rules, ranks in terms of compact rule sets and prediction errors when compared to three well-established RBML algorithms.

In this comparison, we benchmarked SupRB, Decision Trees (DTs), Random Forests (RFs), and the XCSF classifier systems (XCSF) on four real-world regression datasets with different learning task complexities and dimensionalities. As both SupRB and XCSF should be considered Learning Classifier Systems, XCSF produces the most similar types of machine learning models. For both algorithms, we limited rules to use hyperrectangular conditions and linear local models, as the main advantage of these types of models (and motivation to use them) over other models is their inherent interpretability and transparency. We tuned the more sensitive hyperparameters of each algorithm for every dataset independently and then performed a total of 64 runs (8 random seeds and 8-fold Monte-Carlo cross-validation with 25% test data) per algorithm per dataset.

We expected SupRB and XCSF to perform similarly in terms of errors, with smaller models for SupRB. Furthermore, we assumed DT to perform worse than SupRB (maybe better regarding interpretability) and RF to perform better than SupRB in terms of error and substantially worse on model complexity. We found those hypotheses to largely hold. While there is some variation across datasets, general tendencies are quite clear. We did perform an extensive statistical analysis using Bayesian correlated t-tests to critically question these findings. The results of these tests did support the hypotheses and are in-part (where they were most interesting) presented in this chapter. Overall, we conclude that SupRB is a promising new RBML algorithm for creating predictive models with compact sets of human-readable rules.

It is reasonable to assume that SupRB could construct less erroneous models if the pressure to evolve small rule sets was lower. However, as explainability is the main reason to use RBML models, rather than e.g. neural networks, we think that our current models strike a well-acceptable balance. XCSF's models have been severely more complex while only being slightly better with respect to prediction error. Likely, XCSF, RF, and SupRB find themselves at different points on the Pareto front between error and complexity, whereas DT is possibly Pareto-dominated.

The model of SupRB—as introduced in Chapter 4 and benchmarked here—forms the basis for all subsequent investigations featured in this thesis. Chapter 6 will propose new or updated mechanics component-by-component, discuss their relevance given the assumptions about SupRB's goals, and test promising candidates for their merits. In Chapter 7, several approaches to investigate—and potentially improve—SupRB even further will be discussed.

6 Extending and Improving SupRB

After comparing SupRB with some of its closest competitors (cf. Chapter 5), finding that it does perform as hypothesized, the next logical step is to investigate possible improvements and/or extensions to the base formula (cf. Chapter 4).[1] In this chapter, possible adjustments are discussed and various experiments where such extensions have been performed will be presented. Importantly, this chapter also discusses why some of the more typical LCS extensions from past research might not be useful for the context of providing a highly explainable ML system. The sections of this chapter discuss extensions component-wise, starting with the two parts of the rule (condition (Section 6.1) and local model (Section 6.2)), discussing the mixing model used to combine multiple rules to a model (Section 6.3), and, finally, investigating the two metaheuristic optimization processes (rule discovery (Section 6.4) and solution composition (Section 6.5 and Section 6.6)).

Given the introduction of an entirely new system into a mature field, there are seemingly endless possibilities to adapt SupRB. The following tries to provide a well-rounded first investigation of some of the more immediate options, but we had to take sensible limitations to our experiments and focus on the most promising adaptations. In general, my mental approach was: *As someone with a deep knowledge of the literature of the LCS and wider evolutionary computation fields, what would be my "Did you already try X?" questions to a colleague asking for advice on how to improve various aspects of SupRB.* Then, I tried to answer as many of them as I could in a sensible prioritized order[2] and present a large number of other options in the respective sections of this chapter and, of course, Chapter 7, although even that is a non-exhaustive list. Additionally, some options were suggested to us by peer-reviewers of our SupRB-related publications or interested other researchers during conferences. We include them into the respective sections as well. Overall, I am confident that this thesis presents versions of SupRB that are convincing to others interested in implementing (or experimenting with) XAI and does not leave them thinking that this is an unfinished or unrefined system that is too insufficiently tested to even be considered for further investigation.

[1]As introduced in Chapter 4, where I gave a general description of SupRB, Chapters 4 to 6 constitute *C3* as defined by Section 1.1.

[2]My priority mainly took into account whether this is a large system change, e.g. switching whole metaheuristics as in Section 6.5, which should be done early as it might render other work void if the improvements are considerable, whether the actual gains are promising because there is an immediately plausible theoretical basis, e.g. including self-adapting properties as we did in Section 6.6, and whether they still maintain our assumptions about explainability or would just serve as a benchmark to compare to, e.g. we did not test neural network–based rule conditions or local models or even hyperellipsoidal conditions because these will most likely not be sufficiently explainable. However, we did investigate changes to the mixing model that might improve the explainability.

6.1 Rule Conditions / Matching Functions

One of the often discussed and investigated elements of LCSs, especially XCS(F), are the presentations of rules, especially, their conditions. The condition (or matching function) of a rule defines the subspace of the input feature space for which this rule can or should model the data. How a condition is encoded is therefore also problem dependent, in addition to the previously discussed considerations on explainability (cf. Chapter 3). For example, a binary input space would typically be encoded using ternary [Wil95] conditions while integer- and real-valued input spaces are modelled differently. One typical approach for these is to use interval-based conditions.

As SupRB is conceptualized as a regressor for real-world tasks, feature spaces solely made up of binary or categorical inputs are not considered. Similarly, hybrids, i.e. matching functions where some dimensions are matched with one type of condition while others are matched with others, were not yet considered, although that might provide an interesting avenue for future work for very specific real-world tasks. Such hybrids might then contain ternary matching in addition to, e.g., interval-based matching. In general, it is assumed that SupRB operates on real-valued inputs. If the original feature space is different, it should be transformed to real values. As explainability is hindered the more complex a matching function gets, there is a strong incentive to choose simpler matching functions. Following the experiments on interval-based matching presented in Section 6.1.1, Section 6.1.2 will present more complex approaches and will argue why it is reasonable that adopting some of the options is inappropriate for the tasks SupRB is meant to be used for, i.e. those where explainability of models is paramount.

6.1.1 Interval-based Matching in SupRB

Past research proposed four variations of interval-based conditions and a quite numerous number of other forms of conditions utilizing a variety of functions and hybrids thereof. This section will summarize our own experiments regarding matching functions in SupRB.

The easiest possible matching function on real values uses an interval for each dimension. Values inside the interval bounds are matched while those outside are not. In a multi-dimensional input space, this type of function matches *hyperrectangular* subspaces. It is, therefore, often referred to both interval-based and hyperrectangular matching in literature.

Four different ways to encode such hyperrectangular conditions inside a rule have been proposed in LCS literature:

- **OBR**: *Ordered Bound Representation* (sometimes also called MMR: Min Max Representation) is based on the classic way to display intervals, i.e. by directly using the upper and lower bounds. It was first introduced by Wilson [Wil01] for an XCS for data mining with integer inputs. OBR was also used in all SupRB experiments in this thesis. A rule k applies for example x iff $x_i \in [l_{k,i}, u_{k,i}]\ \forall i$ with l being the lower and u the upper bounds. This makes it relatively easy for a human reader to evaluate whether a data point is matched or not. Although, with high dimensionalities, humans can even struggle with this representation.

- **UBR**: *Unordered Bound Representation* is an adaptation of OBR based on the assumption that a potential reordering of the two bounds after mutation might have negative effects on the search behaviour of the evolutionary algorithm [SB03]. A rule k applies for example x iff $x_i \in [\min(p_{k,i}, q_{k,i}), \max(p_{k,i}, q_{k,i})]\,\forall i$ with p and q being stored in the rules unsorted. The potential negative effect on metaheuristic search behaviour due to sorting could occur in the following example: Assume a two dimensional input space and a rule k_1 that has the condition $([3, 7], [1, 5])$. k_1 matches x iff x_1 is between 3 and 7 and x_2 is between 1 and 5. If the mutation operator would now change upper bound of the first dimension from 7 to 2, OBR would require resorting to $([2, 3], [1, 5])$, while the new condition in UBR would be $([3, 2], [1, 5])$. Although mutation only operated on one of the four values, in OBR, two have been changed. While the same space is matched in both representations, the genotype locality of mutation is higher with UBR. Additionally, two genotypes can now produce the same phenotype which may be advantageous—as argued by Stone and Bull [SB05] as no swapping operator is needed—despite the loss of phenotype locality. Dam, Abbass, and Lokan [DAL05] argue that this goes against the building block hypothesis in GAs and may therefore be a disadvantage, leading them to propose MPR (see below). However, for explainability UBR should probably be parsed into OBR before being presented to the user.

- **CSR**: *Center Spread Representation* was the first hyperrectangular representation used in a real-valued LCS, originally proposed by Wilson [Wil00] for XCSR, an XCS adaptation for real-valued inputs. Rather than encoding upper and lower bounds of the intervals, CSR is made up of the center points of the values and the symmetrical distance in both directions from that point to the bounds. A rule k applies for example x iff $x_i \in [c_{k,i} - s_{k,i}, c_{k,i} + s_{k,i}]\,\forall i$ with c being the centres and s the spreads. It is very intuitive to imagine the size of a matched space (at least for individual dimensions) but it is a bit harder to ad hoc determine if a data point is matched. From an optimization perspective, there are disadvantages as well as advantages. For one, it is easier to control the size, e.g., pushing overly specific rules towards generality, or vice versa, without changing the general region that is matched. However, every change always changes both the upper and the lower bound. In contrast to OBR where both values of an interval have the same semantics, center and spread are sufficiently different to probably warrant the use of different mutation rates. This is especially true for SupRB with the Evolution Strategy–based rule discovery introduced in Section 4.1 as we probably want to keep the center relatively stable and increase the spread each generation. Stone and Bull [SB05] found that with CSR, XCS is biased towards more general intervals in its rules.

- **MPR** *Min Percentage Representation* was originally proposed by Dam, Abbass, and Lokan [DAL05] to fix the potential optimization issues of UBR as well as CSR by better adhering to the building block hypothesis, which is violated by switching which gene in the genotype expresses which behaviour in the phenotype in UBR. MPR uses a lower bound (just as OBR) and then encodes the upper bound based on what part of the total space (assuming that there are known bounds for the inputs) is also matched. A rule k applies for example x iff $x_i \in [l_{k,i}, l_{k,i} + (q_{k,i} * (u_{max,i} - l_{k,i})] \, \forall i$ with l being the lower bounds, u_{max} the highest available values per dimension, and q being the percentages of matched area between the lower bounds and the maximum values.

A typical notion for proposing different variants was, for example, a supposed benefit for the evolutionary operators in the search for good rules. As this is a central aspect of SupRB's rule discovery, Anton Huber did investigate this in his Bachelor's thesis under my supervision. The experimental setup was the same as in [Hei+22b; Hei+22a; Wur+22] (cf. Chapter 5, Section 6.4 and Section 6.5, respectively) but excluding the relatively linear, and therefore similar to CCPP in terms of potential knowledge gained, EEC dataset and replacing it with two additional data sets available in the UCI ML repository (namely, Physicochemical Properties of Protein Tertiary Structure (PPTS)[3] and Parkinsons Telemonitoring (PT)[4] [Tsa+09]). SupRB was used with the ES-based rule discovery and the GA-based solution composition as described in Section 4.1 and benchmarked in Chapter 5.

The evaluation showed that UBR and MPR did not perform well within SupRB, while CSR and OBR performed almost on-par with each other, with OBR having a slight potential edge. For most use cases, OBR[5] regularly produced smaller rule sets during individual runs and often also performed slightly better on errors than CSR.

For UBR, the complexities of produced rule sets were similar to those of OBR, albeit with significantly higher average MSEs. The only exception was on PT where the 64 evaluation runs showed considerably worse values than OBR for both metrics. Typically, MPR produced a smaller mean model complexity although it was firmly beaten in terms of errors. A closer look into the individual rules making up UBR and MPR models hinted that they are somewhat incompatible with the rule discovery process of SupRB. Although those are the representations that were developed to make rule discovery easier in previous LCS, UBR and MPR should, in my opinion, not be further considered for practical usage.

To further investigate whether OBR or CSR should probably be used, we performed a Bayesian statistical analysis[6] to compare the approaches:

We first use the Plackett-Luce model–based approach as described by Calvo et al.[CCL18; Cal+19]. Considering all four variants, OBR was expected by the statistical model to perform on rank one for errors with a probability of 40.84% and for complexities with 22.88%. CSR showed probabilities of 42.44% and 13.83%, respectively. The remaining probability mass was favouring MPR (which has the highest probability to be the best in complexity with 35.42% but only 14.48% on error), however, it is too likely that it will underperform on error to be considered in use. These results further confirm that we can drop MPR and UBR from consideration.

[3]https://archive.ics.uci.edu/ml/datasets/Physicochemical+Properties+of+Protein+Tertiary+Structure

[4]https://archive.ics.uci.edu/ml/datasets/Parkinsons+Telemonitoring

[5]For the sake of brevity, I simplify "SupRB utilizing OBR for its rules" to "OBR" in the remaining text. This is analogous for the other variants.

[6]See [Ben+17] on why this is more appropriate than null-hypothesis testing despite its prevalence in ML research.

A direct comparison on errors between CSR and OBR, using the model introduced by Benavoli et al. [Ben+17] and assuming a region of practical equivalence (rope) of $0.005 \cdot \sigma_{\text{dataset}}$, showed:

$$\begin{aligned} p(\text{OBR} \gg \text{CSR}) &\approx 51.96\,\% \\ p(\text{OBR} \equiv \text{CSR}) &\approx \ 18.48\,\% \\ p(\text{OBR} \ll \text{CSR}) &\approx 29.56\,\% \end{aligned}$$

where:

- $p(\text{OBR} \gg \text{CSR})$ denotes the probability that OBR performs better (achieving a lower MSE on test data),
- $p(\text{OBR} \equiv \text{CSR})$ denotes the probability that both representations achieve practically equivalent results and
- $p(\text{OBR} \ll \text{CSR})$ denotes the probability that OBR performs worse (achieving a higher MSE on test data)

When adjusting the rope to $0.01 \cdot \sigma_{\text{dataset}}$ the model yielded a probability distribution of approximately (0.28% | 0.41% | 0.30%), while setting the rope to 0 resulted in the probabilities (0.58% | 0.00% | 0.42%). Regardless of the chosen rope, neither of the two representations managed to reach a threshold of 80%, as suggested by Benavoli et al. [Ben+17], to make any automated decision in this regard. Following those results (similar to our decisions about comparing XCSF and SupRB in [Hei+22b]), we assume that OBR and CSR have specific use cases (datasets) in which they can be applied more favourably but that, in general, especially when considering the raw values in the results, OBR is a good all-rounder and can be chosen for all SupRB experiments going forward.

6.1.2 More Complex Matching Functions

Other than the now extensively introduced interval-based matching functions, which have a clear advantage for explainability, several other matching functions have been proposed in the past for use as a rule's condition.

A popular option is the use of hyperellipsoids (rather than hyperrectangles) as the matching function [BLW08]. It replaces the rectangular with an ellipsoidal shape, which has a clear advantage that is most obvious when comparing a hypercube with a hypersphere: for a spherical shape, data points on the surface of matched space are all at the same distance to the centre. However, for the cube, some points on the surface are closer to the centre than others. Assuming that rules in an RSL model represent areas of similar expected behaviour (regardless of whether we are performing classification, regression, or even reinforcement learning/agent control), these "corners" of the hypercubes/hyperrectangles are not very intuitive and could lead to rules matching data points they should not match. On the other hand, full coverage of all inputs but little rule overlap is easier to achieve with rectangular partitioning. Importantly, the intervals are also easier to analyse and explain to stakeholders than the ellipsoidal shapes which usually are implemented as Gaussian functions with hard cutoffs. This follows the ability to separate individual dimensions by conceptualizing the intervals one at a time, whereas, for ellipsoids the most effective strategy would probably be to think of *"How far is a certain [non-axis parallel] point in space from the centre of the rule condition"*, which is non-trivial for n-dimensional space.

More complex models had also been proposed in the past but have mostly not seen wide adoption. The use of neural networks as rule conditions has been explored by, e.g., Bull and O'Hara [BO02], Bull and Hurst [BH03], and Howard, Bull, and Lanzi [HBL09]. While these enable very complex decision boundaries during matching which can theoretically enhance performance, they are basically unexplainable at this time. Moreover, they are harder to optimize as they contain more parameters. Iqbal, Browne, and Zhang [IBZ14a] introduced GP-like code fragment graphs as rule conditions. They can also model highly complex functions and should show a better explainability than neural networks. Furthermore, partial structures in those trees can likely be exploited to learn different but similar tasks much more quickly (and potentially with less data) than a new training process would require. A first step towards this was made by Nguyen, Browne, and Zhang [NBZ20]. It might be interesting for future work to explore the use of code fragment conditions in SupRB in real world settings and test their explainability in live scenarios. One example where this transferability of knowledge could be very useful is the production process at the basis of the case study presented in Chapter 3. For this product, a very high number of variations exists and new ones are introduced very regularly, which can challenge ML-based models as they will often be tasked with predicting the quality (or the optimal parameters) for a product that was not—or very rarely—produced before. This would require a very high generalization capability, but the use of code fragments might be especially useful here, regardless of the loss of explainability of rule matching which was not deemed essential by the interviewed group of stakeholders. However, in other cases, matching can be very important to stakeholders. This is particularly common when the focus is not on explaining individual decisions, but on analysing the model as a whole.

6.2 Rules' Local Models

Where a rule's condition operates on a view of the environment, i.e. the inputs that lead to a model returning some output, the rule's local model governs which specific output is derived, e.g. the action an agent should perform. This output can be quite diverse based on the model's tasks and integration into the agent. While some agents might allow the model to directly control a robotic arm, equating its output to certain movement of said arm, other models might be responsible to predict the quality of a product (cf. our case study in Chapter 3) which the agent then uses to make adaptations. While there is no "official" or universally agreed upon distinction between agent and model, I assume agents to be more complex systems that contain a model but also other components, e.g. drivers to parse model outputs into real world behaviour.

In general, rule set learning algorithms return a model appropriate for the learning task (cf. Chapter 2). They can be trained using supervised, unsupervised, or reinforcement learning. Outputs can be single values, e.g. a class (in binary and multiclass classification), a real value (in regression for prediction or forecasting), an action (for direct control), or vectors of values, e.g. for multilabel classification, regression of multiple values, estimating distributions over values, or proposing actions and including their expected rewards. For this thesis and the previous publications on the matter, I assume(d) SupRB to train models that perform regression of single values. We did, however, propose a way to extend support towards (multiclass) classification (cf. Chapter 4 and [Hei+22b; Hei+23d]).

In LCSs, models are made up of localized submodels (usually just called a *local model*) that operate on partitions of the input space (cf. Chapter 2). We commonly refer to this combination of local model and matching function as a *rule*. Section 6.1 already contained a discussion on different matching functions. Therefore, the remainder of this section will focus on the models contained in each rule.

The simplest possible local model to be used in SupRB is constant. These models can, theoretically, be used for both classification and regression. However, they are much more suited for assigning a single class label to some input instance than assigning real values as rules should cover as large as possible portions of the input space and—assuming the matched space is not also featuring constant outputs—-a constant value will have high errors for most of the space. Thus, we can expect that in a model geared towards small errors, this will probably lead to very large numbers of rules. In models attempting a balance of rule quantity and error size, this will often lead to unsatisfactory results on both metrics. When XCS is used for classification, it uses constant models [Lan01].

The LCS most commonly used for regression is XCSF [Wil02a]. In its original conception, XCSF used local models that were linearly approximating the function from which the data set was sampled. Each rule contains a linear model of the space it is responsible for. These local linear models are then mixed to make a prediction for a specific data point (cf. Section 2.1 and Section 6.3). While their training algorithms may vary between specific implementations, the usage of linear models is still the most common approach for regression in RSL/LCSs. For example, our experiments presented in Chapter 5 (based on [Hei+22b; Hei+23d]) use XCSF with recursive least squares to estimate the linear local models iteratively, as do [PHH23; PNH24]. In contrast, SupRB uses linear least squares to fit its linear models batch-wise (cf. Section 4.1).

Linear models are generally considered as among the easiest to explain [Bar+20; BP21] which is why our interviewees of the study presented in Chapter 3 expressed a strong preference for rules using linear models rather than more complex ones, even if this might harm performance slightly. Therefore, SupRB also uses linear local models. For SupRB, I experimented with different training approaches to create these models from a rule's matched data. First, ordinary least squares was tested. This deterministic method estimates a linear model from data by minimizing the so called residual sum of squares (the sum of the squared errors per data point). Given that for typical real world data sets (which are the most relevant to validate SupRB on) each rule will only be responsible for relatively few data points, ordinary least squares did very often strongly overfit the data, resulting in poor generalization and, thus, performance on validation/test data. Many approaches exist to prevent overfitting of linear models. One can use k-fold cross-validation, although this requires more data than we can typically expect for a linear local model. There are also numerous approaches that prepare the data to prevent both overfitting and underfitting, e.g. feature selection, feature transformation, or the removal of outliers from the data. However, whether these techniques are actually effective is strongly dependent on the dataset itself.[7] Therefore, for SupRB, a technique should be used that is applicable more generally.

Another option for the creation of less overfitting linear models is the use of regularization terms during training. Two regularizations are commonly applied here:

- L1-norm regularization (Lasso) tends to drive some coefficients to exactly zero, which can be advantageous for explainability purposes (cf. the results of our user study in Chapter 3) as this should select the most predictive features while discarding less relevant ones. However, this can also lead to poorer performance, especially if all features are important to make a correct prediction.

- L2-norm regularization (Ridge) reduces overall feature coefficient magnitudes. It drives all coefficients to zero without including the less relevant ones specifically. It is more robust towards noisy and very small datasets which is to be expected for our local models.

It is likely that due to that last property, Ridge did perform best in my experiments in SupRB. ElasticNet is a combination of Ridge and Lasso but this has not been tested yet.

I did experiment with the interchangeability of Ridge and Lasso and their impact on model structure and the preliminary results[8] are: When training a SupRB model using local models based on Lasso, the performance is slightly worse than when training with Ridge. When exchanging these local Lasso rules with Ridge models trained on the same data, the performance of the overall model does not differ significantly and is not reaching the performance of the model trained with Ridge in the first place. However, when replacing the local Ridge models of a SupRB model that optimized its model structure based on these rules with Lasso models, the system performance drops considerably. It seems like Ridge is allowing the training process of SupRB to make choices that are incompatible (and better performing) than Lasso would. However, as these results are very preliminary, I want to caution against making a definitive judgement based on them.

Generally, Ridge seems to be the choice that can be expected to perform best most often. However, for specific use cases (with varying requirements regarding the feature count) and datasets, appropriate regularization techniques (and their parameters) should be chosen to optimize model performance.

[7]A data scientist can (and should) of course always improve the data quality beforehand to further improve the model induction process.

[8]A student of mine is currently preparing a larger study as his Bachelor's thesis.

Of course, more complex local models than linear ones have been tested in various LCSs before. Of note are local models using higher order polynomials [Lan+05], radial basis functions [SMH18], or neural networks [LL06]. For reinforcement learning, where continuous actions are often important in real-world applications, various approaches to compute the action based on inputs exist. Tran et al. [Tra+07] use a linear model, whereas Howard et. al. [HBL09] use a single neural network to determine both matching and actions from the inputs. Iqbal et al. [IBZ12] dealt with continuous actions by computing them via code fragments (in this case a two branches deep binary tree similar to Genetic Programming). Naqvi and Browne [NB16] incorporated this approach to solve symbolic regression problems.

These more complex models are probably not desirable to use in SupRB as they come with a loss of explainability (cf. Barredo Arrieta et al. [Bar+20]'s analysis on the explainability of models). This is also reflected in the general rejection of non-linear models in our study in Chapter 3. However, it would be interesting to see in future work if more complex local models come with a significant boost in performance (or model complexity reduction) and to evaluate whether the prospective users of a SupRB-based agent actually find the loss in explainability relevant. It is not unlikely that in the way explanations of model predictions are presented to the user (which is an unsolved UI/UX question), sufficient abstraction took place so that the user is not exposed to the model itself, which would make its specific make-up less relevant. However, for now, it is assumed that the (local) models themselves are presented, explanations need to be generated directly from the models (which is harder to do with more complex models [Bar+20; BP21]), and that SupRB with linear models does not suffer from significant performance losses.

6.3 Rule Mixing

In RBML, a solution to the learning task, or simply *a model*, combines multiple individual rules. While in Decision Tree–based models the rules may not overlap and the set always covers the entire input space, LCS/RSL models often have numerous rules matching a single datapoint. Multiple tree-based models may be combined to form an ensemble, e.g. in Random Forests or XGBoost, cf. Chapter 2, which does create rule overlap in these models. The difference to LCSs is that the number of rules matching is fixed based on the number of trees and that these overlapping rules are all part of different models that are individually matching all inputs. Importantly, in LCSs, some inputs may not be matched by any rules or varying numbers of rules. The number of rules matching individual parts of the feature space can have implications on the comprehensibility of models as much as the total number of rules (cf. Chapter 3).

The process of making a singular prediction based on multiple rules is called *mixing* (cf. Chapter 2). Random Forests often use majority voting for classification and averaging for regression as their mixing model. XCS is using a mixing model which is tied to its niche-based fitness, which is what is also used for the evolutionary operator [Wil95]. Effectively, this mixing model is iteratively updated each time step as the individual fitness-based mixing weights are updated for all rules matching the specific input whose action was chosen. As rules usually match different input spaces even when proposing the same action, they are not guaranteed to update at the same pace (or even in the same direction) which can lead to different mixing results even when the same input is matched by the same rules at a later time during training. In XCSF [Wil02a], which is using a mixing model based on XCS's but updates each matching rule every time (as there is no action in XCSF) the issues with this are even clearer. In XCS, the rules mixing weights influence how likely their action is to be selected (together with the policy), whereas in XCSF, they actually directly influence the output by building a weighted average of all matching rules' predictions. A detailed analysis of XCS(F)'s mixing model has been performed by Drugowitsch [Dru07, Chapter 6].

In SupRB, a mixing model based on the inverse variance heuristic mixing [Dru07, Section 6.2.2] is used, although here, we weigh the prediction error and the experience of a trained rule (both determined based on all training data), cf. Equation (4.1). The other sections of this thesis are using this mixing model without any limitations or adjustments. In this section, several adjustments to that model are tested which limit which (and/or how many) rules may partake in the mixing (and thus the final prediction of the model) and how their individual mixing weight is calculated. The main motivation studying adjustments to the mixing model are the results of Section 3.5. Here, stakeholders informed us that they prefer few rules to be used to form an prediction, essentially proposing to impose some kind of upper bound on the number of rules that are mixed, assuming that this should increase explainability. Additionally, some of the adjustments could increase the performance of the model.

The implemented adjustments of the mixing weight calculation can be divided into three groups:

1. A weight to *weigh experience against error* instead of assuming both to be equally important to measure the quality of a rule, which in turn decides how important and impactful it should be for mixing.

2. A *limit to the number of rules partaking* in mixing. This hyperparameter (l in Equation (6.3)) is a direct result of Chapter 3 where stakeholders voiced that they would like mixing to be kept to about three rules. Thus, less rules would need to be investigated for a given prediction. Note that, of course, some predictions might contain less rules as fewer rules are matching the data point in question.

3. A *cap to the influence of experience.* Assuming an n-dimensional problem, it is quite obvious that (assuming linear local models) a rule with experience n should be considered more reliable than a rule with experience $n/2$. However, with the standard mixing model, we would also assign a rule with $10n$ experience substantially more impact on the final prediction than one with $2n$ experience. This can be balanced out by higher errors for bigger rules but there can be learning tasks where over-general rules might not perform that much, i.e. a factor of five, worse than ideally sized one. However, if a local model is strictly better, it should not be dominated by a very general rule just because of the smaller size. While the trend towards larger rule sizes dominating could yield easier to interpret models, it might also introduce overly complicated models as certain regions of the input space might get harder to approximate. One could argue

that the solution composition (SC) mechanism should select the optimal balance automatically due to its fitness pressures and that mixing is unimportant for that. On the contrary, a mixing model that is also fixing such imperfections might help SC to find even better rule combinations which reduce overlaps, thus being arguably quite useful for comprehensibility (as decision trees are doubtlessly easier to comprehend than RSL models). Therefore, the maximum mixing weight a rule can acquire due to its experience is capped to a fixed value (either a constant or a linear function of learning task dimensionality).

It is of course possible to execute all of these additional mechanisms at once, changing Equation (4.1) to the new mixing weight formula for rule k:

$$\tau'_k = \frac{1}{\text{error}_k} * \min(\text{experience}_k, \text{cap}) * \text{weight} \tag{6.1}$$

However, potentially there are fewer (at most l) rules that partake in the mixing than originally:

$$M_x = \{k_i \mid i \in \{1, \dots, K\}, m(\psi_{k_i}; x) = 1\}, \tag{6.2}$$

with M_x denoting the original set of matching rules and i being an index within the pool (list) of rules of size K as defined in Section 2.1.2.

We now form the set M'_x by using the function $f(\cdot)$, which receives the index of a rule within the pool and returns its fitness value so that it only contains the l fittest individuals.

$$M'_x = \{k_{i_1}, k_{i_2}, \dots, k_{i_l}\} \subseteq M_x \quad \text{such that} \quad f(i_1) \geq f(i_2) \geq \dots \geq f(i_l) \tag{6.3}$$

Thus, the model built by SupRB modifies from the generic LCS formula (Equation (2.1)) to:

$$\hat{f}_{\mathcal{M}}(\theta, x) = \frac{\sum_{k \in M'_x} \tau'_k \hat{f}_k(\theta_k; x)}{\max(\sum_{k \in M'_x} \tau'_k, 1)} \tag{6.4}$$

Note that—in contrast to Equation (2.1)—matching is now contained implicitly via the definition of M'_x, the subset of matching rules according to the predefined limit l. As before, if no rule matches, $\hat{f}(x) = 0$, which is the mean of the training data as we assume the outputs to be standardized.

The following experiments on mixing also investigated whether there is a notable effect when SupRB is not using the best l rules according to fitness, but uses l randomly selected rules or selects l rules based on a fitness-weighted roulette wheel selection. Note that any randomness in the creation of the mixing model can (if more than l rules match a data point) change the model's output between two predictions on an identical input. For inference in practical applications this is, of course, very undesirable. However, we assumed it could be interesting to check whether it has an impact on training or overall model performance or even helps the model to avoid rule overlap. Reduced rule overlap could then make explanations easier and to avoid the issue of non-deterministic predictions, the mixing model could be slightly adjusted to use Equation (6.3) again.

6.3.1 Experiments

To test the different modifications to the mixing model, especially with regards to the performance impact of limiting the number of participating rules, a set of experiments was performed. Setup, hyperparameters and datasets and the specific mixing model modifications and their parameters are presented in the next two sections.

Setup, General Hyperparameters, and Datasets

To test the impact of different modifications to our mixing model, we follow the experimental setup of Chapter 5 and [Hei+22b; Hei+23d]: While other optimizers have been proposed for rule discovery (RD) [Hei+22a] and solution composition (SC) [Wur+22] in SupRB, previous results have largely been inconclusive, with ES and GA, respectively, probably being a good (or even the best choice) for the optimizers.[9] Therefore, we use these optimizers as well as the remaining general configuration from Chapter 5:[10]

We standardize the target and transform input features into the range $[-1, 1]$. We assume that for the datasets under investigation discovering a total of 128 rules is sufficient to compose good models from. Hence, 32 cycles of alternating rule discovery and solution composition are performed, generating 4 rules per cycle. For the ES, we selected a λ (size of the offspring population) of 20. The GA performs 32 iterations with a population size of 32. To tune some of the more sensitive parameters, including some of the parameters introduced in the next section, we perform a hyperparameter search using a Tree-structured Parzen Estimator in the Optuna framework [Aki+19] that optimizes the average fitness on 4-fold cross-validation. We tune the datasets independently for 1000 iterations or a maximum of 120 core hours per tuning process.

The final evaluation, for which we report results in Section 6.3.2, uses 8-split Monte-Carlo cross-validation, each with 25 % of samples reserved as a validation set. Each learning algorithm is evaluated with 8 different random seeds for each 8-split cross-validation, resulting in a total of 64 runs per algorithm per dataset.

[9]Updated results on possible optimizer choices can be found in Sections 6.4 to 6.5 but the general conclusion of the ES+GA combination being generally good remains.

[10]The code to reproduce the experiments can be found in SupRB's experiment repository https://github.com/heidmic/suprb-experimentation/ while the specific code version is archived at https://doi.org/10.5281/zenodo.14181292.

We evaluate the four approaches on the same datasets as in [Hei+22b; Hei+23d] and Chapter 5 which are taken from the UCI Machine Learning Repository [DG17]:

- Combined Cycle Power Plant (CCPP) [KT12; Tüf14]
- Airfoil Self-Noise (ASN) [BPM89]
- Concrete Strength (CS) [Yeh98]
- Energy Efficiency Cooling (EEC) [TX12]

More information on the datasets can be found in Section 5.2.1 and Table 5.1.

Tested Mixing Modifications

For the second mixing weight calculation adjustment (cf. the list at the beginning of Section 6.3), values for l (cf. Equation (6.3)) are chosen from $\{1, 2, \ldots, 5\}$, which limits the rules that are allowed to be mixed. Given that the stakeholders of the case study in Chapter 3 voiced their preference for 3 rules or less, testing the values above and below for the limit seems reasonable. We assume that more than 5 rules are probably rarely relevant given probable redundancy for areas of such large overlaps. Additionally, considering the sizes of SupRB's entire model in the experiments from Chapter 5 (cf. Section 5.2.2), higher numbers of rules matching the same datapoint seem unlikely. Note that always using only a single rule in the mixing model equates to an overall model similar to the one shown in Figure 4.1.

In addition to selecting only the best rules according to their fitness (called "l Best" later on; cf. Equation (6.3)), two variations of this approach are included in the evaluation:

1. Instead of selecting the rules that exhibit the highest fitness, up to l matching rules are selected at random. We refer to this as "l Random".
2. Rather than selecting purely random, a fitness proportionate random selection similar to the well-known roulette wheel selection, common in population-based EA and XCS(F), is performed. We call this "RouletteWheel".

Both approaches are mainly meant as a sanity check (see the explanation earlier in Section 6.3). The most interesting aspect should remain the selection of the l best rules rather than all rules.

For the third adjustment, which caps the influence of experience on the mixing weight (cf. Equation (6.1)) with an upper bound, the cap can be set in two ways:

1. By setting the cap to a fixed value ("Experience Cap"). To reduce the computation load, we hyperparameter-tune cap $\in \{20, 50\}$ rather than testing all values.
2. By setting it to a value linearly dependent on the dimensionality of the learning task at hand ("Experience Cap (dim)"). We hyperparameter-tune cap $= c * \dim_x$ for $c \in \{2, 5\}$.

The optimal value for model complexity is highly task-dependent and is influenced by both the dimensionality and noise of the learning problem. For a noiseless linear task, it is well-established that a linear model can achieve perfect fitting when the sample size equals the number of parameters in the model. However, most real-world tasks, such as those encountered by SupRB, are non-linear and noisy, requiring more sample points to reduce variance in local models. While increasing sample size matched by a rule may help reduce variance, the non-linearity introduces higher bias due to linear approximations being less local.

SupRB aims to balance this bias-variance trade-off by maintaining smaller local models. By preventing larger models—those with more matched examples during training—from dominating predictions and introducing excessive variance, this proposed adjustment to mixing might lead to an overall better fitness. Since we lack reliable information about the noise level in many tasks but often have insights into their dimensionality, it makes sense to use dimensionality as a proxy when setting the threshold.

Regardless of choosing a direct value or something linearly-dependent on the dimensionality, we can assume that sufficiently extensive tuning finds a sensible cap. There is some hope that tuning can find a value for the cap that is “sufficiently good for most tasks” to not require future tuning of this value and I conjecture that if it exists, it is probably correlated with the dimensionality.

Theoretically, a lower experience bound, which removes rules from the mixing if they have been trained on less training data points than that bound, could be plausible. However, it is not intuitive how to set that value. While it is probably problem dependent, a value equal to the number of dimensions of the learning task is likely reasonable for most tasks. For tasks that are subject to a lot of noise—even restricted to some subspace of the task—any training algorithm will often require a larger sample to fit the linear model well on. In contrast, for subspaces where no noise is present and a perfectly approximating local model is constant (rather than linear or more complex), a single sample point and, therefore, an experience of 1, would be sufficient.

For the first adjustment, which weighs experience against error, we kept a default value of 1 for the experiments of this section. The main reasoning was that we assume this would introduce an additional pressure towards different population structures which works alongside the solution fitness. This would be undesirable from a configuration point of view as, ideally, each aspect is governed by a single parameter or at least by parameters that have predictable interactions, which a weight at this point would probably not show. Preliminary results hinted that this assumption might be correct. Thus, we decided to not invest the considerable additional compute.[11]

[11]Reasonably, one should test at least three values but five to ten or more would be better. Given that we already had twelve combinations of options each tested on rule counts of one through five, resulting in tuning parameters 240 times (due to the independent tuning on four datasets) and, based on the tuning results, 15360 runs to generate the plots in Section 6.3.2, this would drive the already considerable runtime up substantially without an obvious gain in insight.

Overall, we thus arrive at twelve combinations, four options (including the original one) of selecting which rules take part in mixing and three options (also including the original one) on how to calculate the mixing weight. We tested each of these on the five rule counts and four datasets.

6.3.2 Results

This section presents the results of our investigations on modifying the fitness function in SupRB to improve performance and explainability. Given the large number of different experiments (cf. Section 6.3.1), I cannot present them exhaustively here but will focus on the key insights that were generated.

In the following graphics and discussions, the errors and complexities presented and contrasted will have been normalized to an interval of $[0, 1]$ across all evaluation runs per dataset. Thus, we did select the best and worst error (or complexity) achieved by any mixing approach and set them to 1 and 0, respectively. For each dataset, we had 3840 evaluation runs from which we chose those values, and then normalized all other results on these intervals. We did this to bring the results on equal scales across datasets and to make general discussions a bit easier. Note that interpretations should be made carefully as the error distributions on these datasets are not identical (cf. Section 5.2.2).

Table 6.1: Overview of the metric results to which the normalized intervals of $[0, 1]$ correspond to per dataset.

	MSE		Complexity	
Dataset	Min	Max	Min	Max
Combined Cycle Power Plant	0.059	0.757	1	65
Airfoil Self Noise	0.092	1.035	1	63
Concrete Strength	0.080	1.017	1	67
Energy Efficiency Cooling	0.011	1.065	1	76

Table 6.1 shows which original result corresponds to the interval values on a per-dataset view. Especially interesting are the maximum errors and the minimum complexities. Across all datasets, there was always at least one run that only used a single rule as its final elitist, thus effectively becoming a single Ridge regression model. Additionally, ASN, CS, and EEC all showed runs where the errors after training (and therefore optimizing that metric specifically) were about what would be expected from a model that only returns the mean of the data.

The first question is of course: How did the SupRB baseline using the unmodified mixing formula fare under the normalization scheme? Overall, the complexities ranged from 0.05 to about 0.8 and, while there was a noticeable gap between about 0.3 and about 0.4, the runs were relatively uniformly distributed, showing no clear clustering otherwise. The results are slightly poorer than those from Table 5.3, which is the result of cutting the tuning budget of the runs to one third. On the other hand, a majority of errors of the runs from the baseline were between 0 and 0.1 with only about 5% between 0.1 and 0.22 and no runs above.

The best results achieved in this set of experiments were from the runs that did not restrict the number of rules in the mixing model but added an upper cap on the effect of experience. Here, making it dependent on the dimensionality of the input space seemed to be slightly better than directly controlling the number. Their distribution of complexities was about the same as that of the baseline, while the errors were a bit better, giving a tighter distribution and falling more commonly below 0.1. However, both showed outliers at 0.3, 0.42, 0.48 and 0.51 which did not occur in the baseline runs.

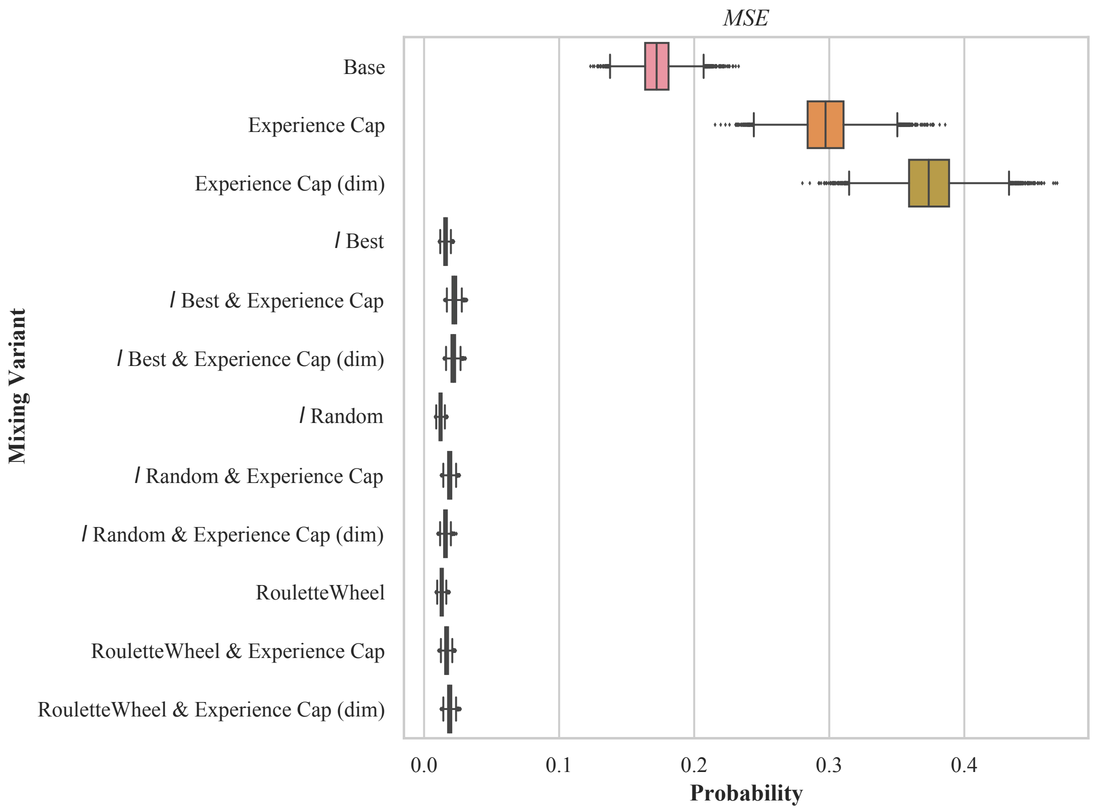


Figure 6.1: Box plot (with standard 1.5 IQR whiskers and outliers) of the posterior distribution obtained from the model by Calvo et al. [CCL18; Cal+19] applied to the normalized MSE data for $l = 3$. A mixing model having a probability value of q % says that the probability of that mixing model performing the best with respect to MSE is q %.

The dominance of these runs at achieving low error models is further illustrated when applying the Bayesian statistical analysis model proposed by Calvo et al. [CCL18; Cal+19][12] to our run

[12]We use the implementation provided by https://github.com/dpaetzel/cmpbayes using 10,000 steps in the MCMC process. We tested larger numbers for sampling, namely 20,000, 50,000 and 100,000, as well but found no

data, which, for each of the mixing models, provides us with the *posterior distribution over the probability of that mixing model performing best.* As we expect from our prior knowledge that using a maximum of three rules ($l = 3$) should be plausible, we show and discuss the test results for this value. We provide a box plot which shows the most relevant distribution statistics in Figure 6.1. The majority of options seems to be relatively equal, whereas the mixing models without the restriction on rule counts clearly perform better. Together they make up about 65-70% of the probability mass, with the baseline taking about 17% itself. A typical minimal threshold for automated decision making is 80 % (or usually even more) for a single algorithm/approach [Ben+17]. While we could not select any one approach based on this, we could discard all rule count–restricting models based on this if we were to only care about minimizing errors.

We performed some further statistical testing for MSE with $l = 3$ but only comparing the mixing approaches that actually use said l. The results can be found in Figure 6.2. Regardless on how the number of rules was restricted, a positive effect of using a cap on the experience seems to transfer. We can probably assume that, given the relatively small rule set sizes a SupRB model typically features, the case that restricting to three rules is not occurring as often, which would explain why a fully random selection is not as detrimental. This is likely further illustrated as fitness-weighted randomness or strictly fitness-based selection of the rules in the mixing model do not seem to have vastly different (albeit marginally better) results.

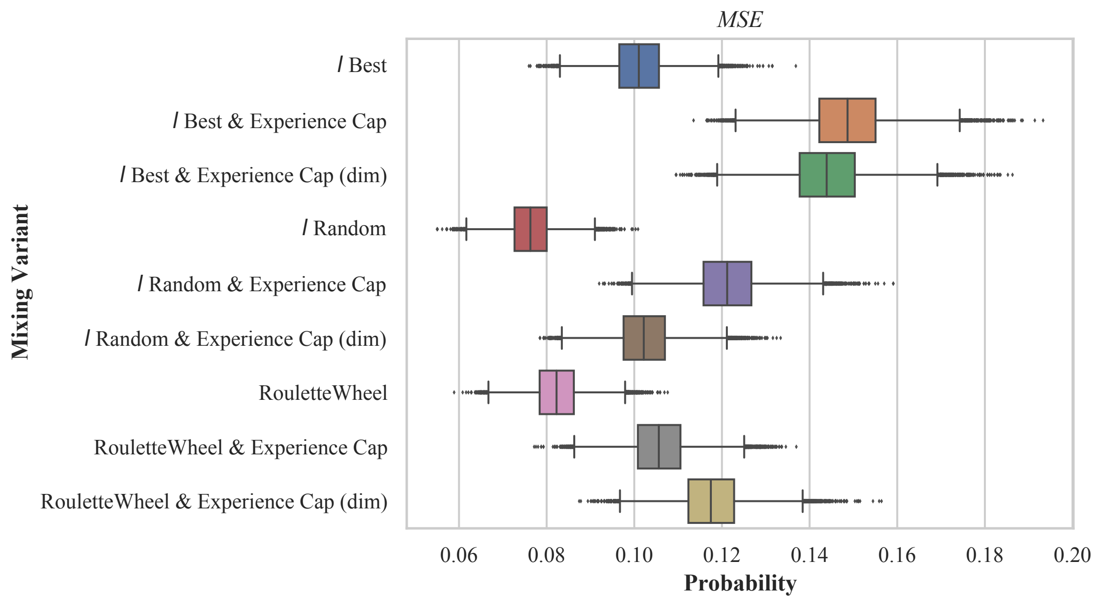


Figure 6.2: Box plot in the same style as Figure 6.1 but only computing the rank probabilities on the normalized MSE data for the mixing models that introduced a limit to at most 3 rules matching.

differences which is why the remainder of Calvo models in this thesis uses 10,000 steps unless specifically stated otherwise.

When setting l to other values, the overall results of the statistical analyses do not change significantly. With $l = 5$, the other options do indeed catch up marginally but it does not seem to have a large positive impact. However, setting $l = 1$ results in a clear decision against any of the rule count–restricting models. When observing complexity, $l = 1$ often leads to collapsing populations for these approaches, regularly only featuring a single rule in the model. This is especially pronounced when using the fitness-based selection within the mixing model, but also happens with both randomly selecting approaches. When using fitness, this could probably be explained by the fact that the fitness of a rule is, in part, based on its volume, giving higher volume rules a higher fitness. Whereas traditional SupRB is able to still discover useful small rules and effectively combine them, a substantial additional fitness pressure seems to lead to collapsing populations. Randomly choosing the rules also has an obvious effect on a guided search and likely interrupts any effective optimization during solution composition, which explains the collapsing populations there.

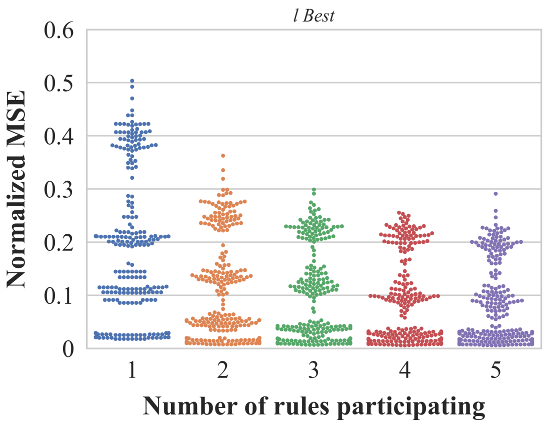
When looking a bit closer into the distributions of runs on the various values for l and the different mixing models, we find that larger values for l loose their effects, which reaffirms the results of the statistical testing for those values. A selection of these distributions is presented in Figure 6.3.[13] The other (non-shown) mixing models exhibited similar behaviours.[14] To contextualize, recall that the baseline's runs showed (normalized) errors that were about half the size of the ones exhibited here (see above). As expected, $l = 1$ is noticeably worse than higher values for l for all of the variants. However, the improvement of errors when using higher values for l is not as pronounced (across datasets and steps in the value) as one might expect. Selecting the best rules according to their fitness is clearly a more consistent approach towards building a good model but the differences were not especially sharp (this was of course also the result of our statistical testing).

A plausible reason for this is that only few rules were matching for most datapoints (recall the relatively small model sizes from Table 6.1). When l or less rules are matching, any restriction is essentially meaningless automatically. Not only that, it is also plausible that any selection scheme has a good chance of selecting the best l rules for $l + 1$ matching rules. And even if not the best l rules are selected in cases of more matching rules, the model usually only contains valuable rules to begin with and it is not too likely that rules with overlapping responsibilities exhibit vastly different local models as long as they are appropriately sized for this niche. Tied into this is also the possible explanation for a poorer performance than the baseline: Restrictions on the rules obscure the fitness signal. The optimizer is no longer seeing the potential performance of the model as a whole if not all matching rules are contributing. This harms the search process for a good model during the SC phase, which in turn leads to less optimally placed rules in the next RD phase(s).

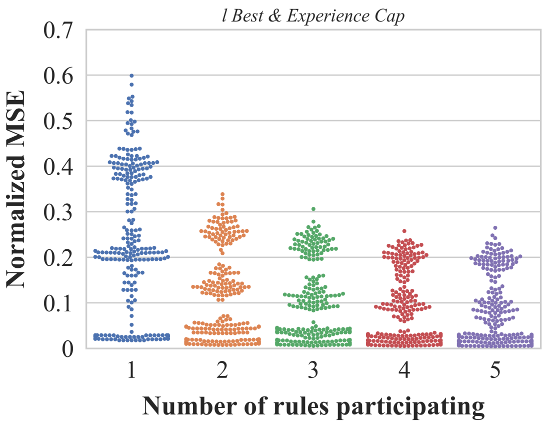
When analysing model complexity, the results are somewhat surprising. Capping the experience produces populations similar to the baseline but all other approaches were vastly different. l Best often leads to collapsing populations for $l = 1$, while it managed to always evolve larger

[13]Note the different scales of the individual plots which were fit to the runs for better readability.

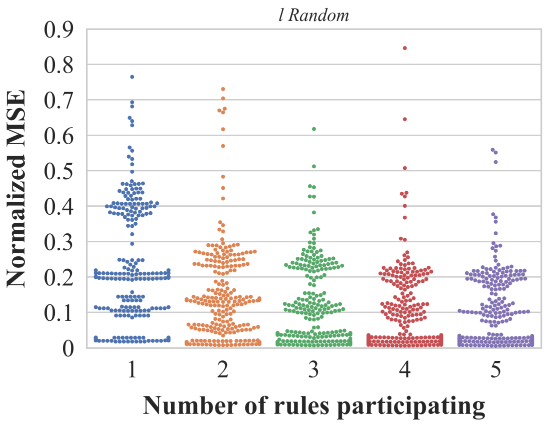
[14]The full set of plots can be found at https://github.com/heidmic/diss-graphs/tree/main/MIX. The t-tests and Calvo models in the repository are all for $l = 3$.



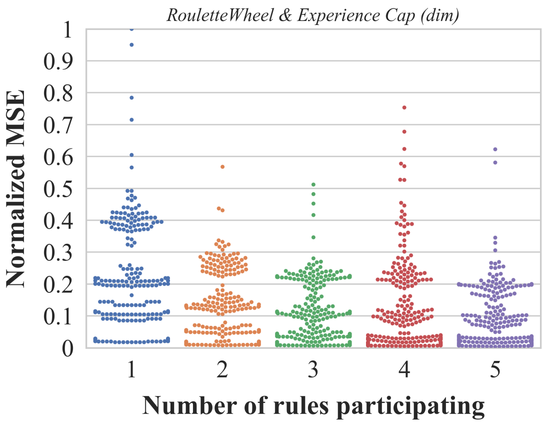
(a) Distribution of runs using a mixing model with the l best/fittest rules.



(b) Distribution of runs using a mixing model with the l fittest rules and a hyperparameter-tuned cap on the influence on experience.



(c) Distribution of runs using a mixing model with the l random rules.



(d) Distribution of runs using a mixing model with fitness-proportionately randomly selected l rules and a hyperparameter-tuned (in relation to the problems dimensionality) cap on the influence on experience.

Figure 6.3: Distribution of runs' normalized errors for various mixing models. Shown are 256 runs per mixing model and value for l. Due to clustering within the results, not all points may be present within the visualizations.

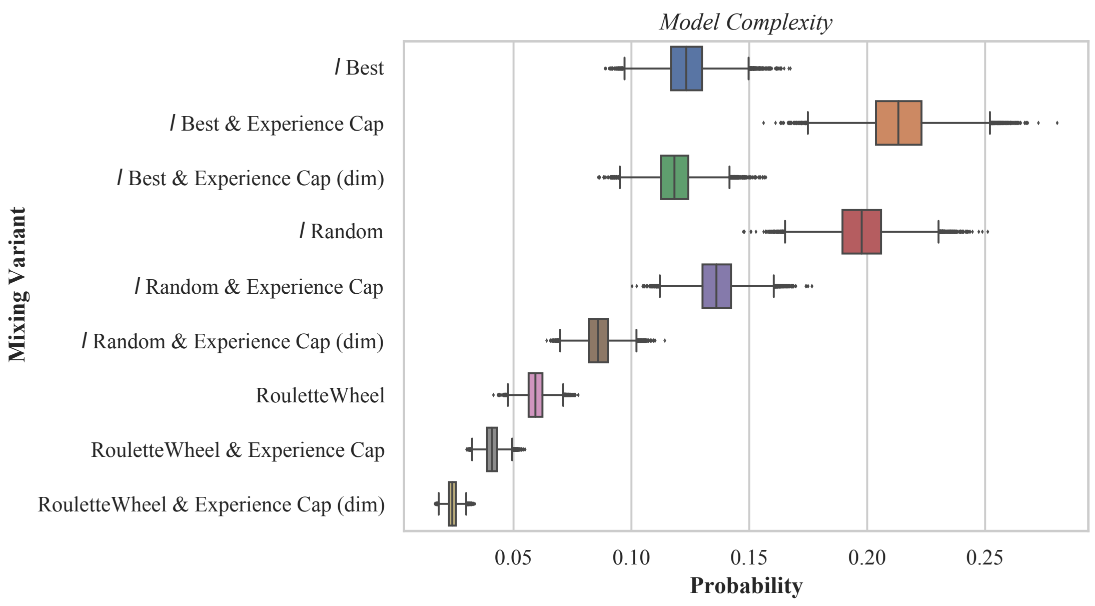


Figure 6.4: Box plot (with standard 1.5 IQR whiskers and outliers) of the posterior distribution obtained from the model by Calvo et al. [CCL18; Cal+19] applied to the complexity data for $l = 3$.

models for higher values of l. They were, however, much less distributed over the complexity range, tending towards small (but not collapsed) populations. With increasing l, the cluster where most of l Best's runs landed did also increase in model size, which indicates that the theory of additional pressure towards small models introduced by this could be correct. l Random and RouletteWheel also tended to similarly small models (and only collapsed for $l = 1$) but showed a few high outliers which l Best did not. Due to this, the statistical analysis on complexity in the style above is clearly showing that the non-restricting (thus, effectively having an infinitely large l) mixing models are expected to perform worst with negligible probabilities to be on rank one. However, this is somewhat misleading as an overall performance metric, as the very small populations are in-turn showing relatively bad results on errors (as expected for non-linear learning tasks but very few rules). Figure 6.4 shows the analysis results for $l = 3$ and only the rule count restricting mixing models. According to the analysis, RouletteWheel should be expected to perform worse than even full randomness. The cap on experience did in all but one (l Best & Experience Cap) cases lead to larger populations, with the cap based on dimensionality performing consistently “worse”. Full randomness did not evolve models that much smaller than selecting for high fitness, as this was hardly possible due to their very small size compared to the standard mixing model, which already is producing rule sets that are again very small compared to XCSF.

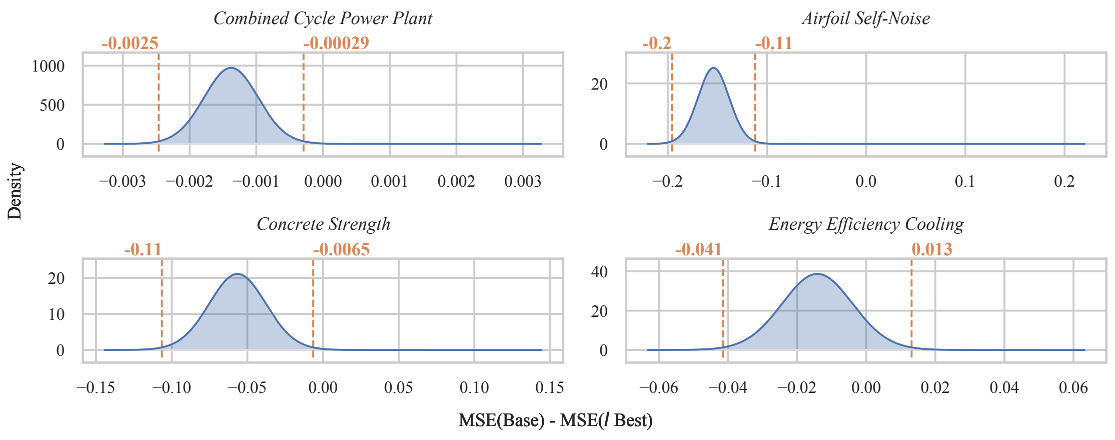


Figure 6.5: Density plot of the posterior distribution obtained from Corani and Benavoli's Bayesian correlated t-test [CB15] applied to the difference in MSE between the baseline and the l Best mixing model on each dataset for $l = 3$. As the MSEs are now given individually, they are no longer normalized over all datasets as before but are still standardized on the individual data. Orange dashed lines and numbers indicate the 99 % HPDI (i.e. 99 % of probability mass lies within these bounds). HPDI bounds rounded to two significant figures. Effectively, this indicates how likely a specific difference between one run with each mixing model will be (practically, probability mass above zero equates to the baseline having a higher (worse) value than l Best).

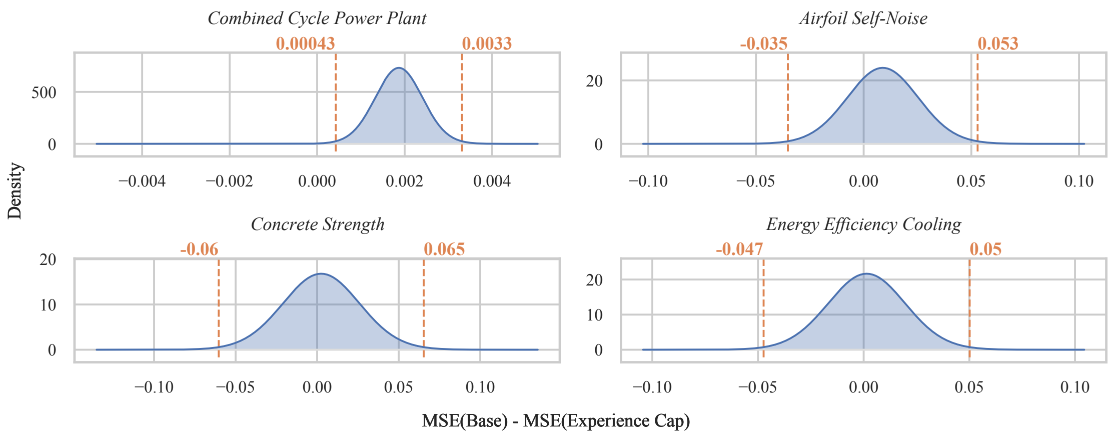


Figure 6.6: Density plot of the posterior distribution similar to Figure 6.5 but between the baseline and the Experience Cap mixing model.

We also performed Corani and Benavoli's Bayesian correlated t-tests [CB15][15] as we did in Section 5.2.2. The resulting posteriors for the actual (standardized) MSEs per dataset including 99 % high posterior density intervals (HPDIs) are the *distribution of the difference between the errors for each mixing model.* Due to the large number of experiments, we again focus on two key candidates: l Best and Experience Cap. Figure 6.5 shows that indeed l Best comes with some losses on all datasets. When comparing them to Figure 5.3, we find that these were not dissimilar to the difference between the results of SupRB and the Decision Trees (DTs). One should note, however, that based on these analyses, we cannot conclude that l Best would perform worse than DT. In Figure 6.6, we find that the actual difference in errors between Experience Cap and the baseline is very small and likely not practically significant, even if Figure 6.1 could have been interpreted towards a statistically significance in the difference of the achieved results. For EEC and CS, there seems to be no improvement. The improvement on CCPP is not practically relevant and the improvement on ASN is quite minor.

To summarize: The experiments showed that restricting the maximum impact of experience may have some minor beneficial effects on prediction errors (and can also have this effect on complexity, albeit less consistently). However, this effect is likely not especially pronounced on many datasets, although it might become more relevant on larger datasets, e.g. big data scenarios like Section 3.5. We found that this effect can be achieved without system knowledge by including that parameter in the hyperparameter tuning process. As a general recommendation, I would, however, suggest to not add another parameter to the search. We also showed that we can restrict the number of rules partaking in a mixing model, but that this does come with a measurable and likely practically significant loss in predictive power. Whether this restriction really adds to the explainability of the model remains to be proven. If there should be a measurable benefit, the next step would be to determine whether the increased errors are worth the trade-off. Interestingly, even randomly selecting the subset of matching rules to contribute to the global model's prediction does have less negative effects than expected (at least for moderate values for l). This could suggest that either all rules are close to equally good, or it could be interpreted as them having similar local models and would be merged if the matching functions would allow more complex geometrical shapes. Potentially, given that solution composition seems to not have been completely inhibited, even when choosing a fully random selection of rules during mixing, but there is still a measurable impact on the overall training (possibly due to the obscured fitness signal), this form of more restrictive mixing could be tested during inference-only on a model trained with the original mixing formula, which was originally not attempted as it was not deemed overly promising.

6.4 Rule Discovery

After determining, discussing, and testing possible extensions to the rules of SupRB and how they interact, the next step is to investigate whether more effective methods for model selection can be found. Recall that SupRB splits its model fitting and selection process into two

[15]We use the implementation provided by https://github.com/dpaetzel/cmpbayes.

alternating and repeatedly performed phases: rule discovery (RD) and solution composition (SC) (cf. Section 4.1). SupRB's models' practical transparency (or explainability in a broader context) is primarily influenced by the effective number of rules and their placement within the feature space, which are determined by the two phases (cf. Section 4.3.1 and Section 4.3.2 for a detailed discussion). Therefore, the—preferably near optimal—placement of rules should be a primary concern when designing an LCS algorithm with explainability requirements.

This section discusses possible extensions for discovering rules from data, while Section 6.5 follows a similar approach to determine more options for model building from these rules. Different methods to discover new rules that fit the data well are proposed and benchmarked against each other on a variety of real-world datasets. This section further extends our previous works on this topic. The first was presented at ECTA / IJCCI 2022 [Hei+22a] and was subsequently expanded with new RD methods for publication in the Springer Nature Computer Science journal [Hei+23c]. The extensions in this thesis feature investigations on additional datasets when compared to the previous publications.

6.4.1 Learning Classifier Systems and Rule Discovery

As previously introduced in Chapter 2, LCSs are a family of rule-based learning systems that—typically—construct their models using evolutionary algorithms. At the very least, the metaheuristics determine the shape of individual rules by adjusting the matched areas, but in some classes of LCSs their role does go beyond that. Recently, we made advances towards a modern classification system in [Hei+23a]—substituting the old differentiation in Michigan-style, Pittsburgh-style, and Hybrid systems—where batch (learning) versus online (learning) and single- versus multi-solution form the basis of classification (cf. Section 2.3.1). In that system, SupRB is classified as a batch multi-solution system. Due to those very different outsets, LCSs from one class approach rule discovery and improvement substantially differently from other classes.

For example, the most widespread online single-solution system, XCS [Wil95], features two mechanisms to determine new rules.[16] The first is the covering mechanism: Whenever the number of rules that match a new data point falls below a predetermined threshold or the fitness of the matching rules is too low, new rules are generated that match this data point. These rules are often randomly made more general than to just match this point specifically and therefore slightly differ when inserted into the population of rules. The second is the evolutionary algorithm which is invoked regularly on matching rules that also proposed the actually-chosen action. It utilizes crossover and mutation mechanisms appropriate for the types of input data and evolves the population in a steady-state manner.

In the established batch multi-solution systems, e.g. GAssist [Bac04; FKB13], new rules can be added directly to an individual of the population (a set of rules), shared between individuals, or be the product of a mutation (and, more rarely, rule-level crossover) operator. As the evolutionary algorithm of these systems operates on rule sets rather than rules directly, the

[16]XCS is also the most widespread "Michigan-style" system if we follow the old classification system.

fitness signal that guides evolution is not based on individual rule performance but rather the performance of a given set of rules, which can complicate rule discovery due to the added indirection.

6.4.2 Rule Discovery Mechanisms for SupRB

The main subject of this section is SupRB's RD phase's optimizer. In contrast to many typical optimization problems, we do not want to find a singular globally optimal rule but rather a set of localized rules that perform well in their particular feature space partition. Thus, we are attempting to find an unknown number of local optima without mapping the entire fitness landscape (or even all local optima). The discovered rules should be a diverse set to enable SC to compose a good overall model from them, which would be difficult from a set of very similar rules that do only cover a small part of the input space. However, as SC will select the subset of discovered rules most appropriate to solve the learning task, some heavily overlapping rules in the pool are—at least assuming the training works as intended—not an issue. In the following, the different optimization approaches for RD that will be compared in this section are presented. As we did previously [Hei+22a; Hei+23c], we build two baselines to compare the new algorithms against. For the first, we use the simple evolution strategy which is used in many other publications on SupRB (cf. [Hei+22c; Hei+23d] and Section 4.1). For the second, we employ an even less sophisticated random search that only exploits information from the SC phase but otherwise places rules randomly. We compare those with three different variants of novelty search, for each of which we test two substantially different ways of applying the archiving component (against which the novelty gets computed). For all heuristics, we utilize the same approach for calculating rule fitness by combination of two objectives based on the in-sample error and the matched feature space volume, respectively (cf. [Hei+22c] and Section 4.1).

Evolution Strategy

The traditional choice for an LCS's optimization processes is some form of evolutionary algorithm. Therefore, the first strategy employed for RD in SupRB, which was also used for the experiments in [Wur+22] and [Hei+22b], among others, is an *Evolution Strategy* (ES), specifically, a simplified $(1, \lambda)$-ES [Hei+22c].

The approach is summarized below and found in Section 4.1 as Algorithm 2. The ES's initial individual is generated by selecting a random example from the training data around which a rule is placed, preferring those examples exhibiting a high in-sample error in the intermediate global solution. This individual serves both as the initial candidate for addition to the pool and the parent of the next generation. From this parent, we generate λ children with a non-adaptive mutation operator, which moves the upper and lower bounds further outwards by adding random values sampled from a half-normal distribution. The child individual with the highest fitness becomes the parent for the next generation. If this individual showed a better fitness than the current candidate, it also becomes the new candidate. When for a fixed number of generations no new candidate has been found, the evolutionary search terminates. This ES produces one rule at a time, which allows it to be easily parallelized. One merit of this approach is in the comparatively high explainability of both the search procedure and the resulting pool. In general, rules that have fitnesses independent from other rules are easier to understand for most non-experts (and even experts for more complex models). Whereas, in most current LCSs, the fitness assigned to each rule is highly dependent on the other rules it is surrounded by. Beyond these fitness-based considerations on the understandability of our learning process, the ES itself is also an easy to follow search method: Expand the area (or hypervolume) an individual matches, evaluate the new individuals, choose the best new option and repeat.

Random Search

As an alternative to the strongly fitness-guided RD performed by the ES, we introduce a form of *Random Search* (RS). RS commonly serves as a baseline for testing the performance of other optimization algorithms throughout the metaheuristic literature due to its simplicity, which usually comes with some inefficiency. Therefore, any more sophisticated metaheuristic algorithm should be able to beat RS on complex problems with equal tuning budgets. Furthermore, with the ulterior motive of finding diverse rules to add to the pool, RS provides an interesting approach where the fitness only plays a role in the selection of the final candidate but not in the generation of new rules.

In SupRB, RS (cf. Algorithm 4), similarly to ES, randomly selects a fixed number of data points with the probability of selection being weighted by their respective in-sample prediction errors in the last solution candidate (produced by the previous SC phase). We then place random bounds around those points based on half-normal distributions (to ensure we always match the selected point). To balance the computational cost between RD approaches, we produce substantially more rules initially than we would in an ES generation but a number roughly similar to the total number of children present in the ES. We then greedily select the rule(s) with the highest fitness to become part of the pool.

Novelty Search

One of the central challenges of RD is that the optimizer's objective is to find multiple rules that partition the feature space and, in their individually matched hypervolume, predict data points well. Contrastingly, optimizers operating on many typical optimization problems are expected to find a single global optimum (or at least a point very close to it). The RD's primary objective can somewhat be viewed as the optimizer being tasked to map an unknown number of deeper local optima within the search space. In other words: With RD we aim at finding a diverse set of well-performing rules for all areas of the feature space.

Algorithm 4 Rule Discovery with RS

```
 1: procedure DISCOVER RULES(elitist)
 2:     rules ← $\emptyset$
 3:     for $i \leftarrow 1$, n_rules do
 4:         candidates ← $\emptyset$
 5:         for $k \leftarrow 1, \lambda$ do              ▷ Randomly generate a large number of rules
 6:             candidates ← candidates ∪ INIT RULE(elitist)
 7:         end for
 8:         rules ← rules ∪ candidate with highest fitness
 9:     end for
10:     return rules
11: end procedure
```

In this section, we describe a new approach towards discovering rules based on Lehman's *Novelty Search* (NS) [Leh12]. In this evolutionary search method, the optimizer tries to find individuals that exhibit new behaviour previously unknown within the population, rather than being guided (or at least not being fully guided) by the typical fitness function. In SupRB, behaviour of rules can be equated to what subsample of the training data they match. We want to find a rule that predicts an area of the feature space currently unmatched or only matched by more general rules with high errors in this area, thus, a rule displaying behaviour previously unknown.

For our adaptation, which is outlined in Algorithm 5, we base the NS on a (μ, λ)-ES with elitism and follow the extensive experimental findings laid out by [GMC15]. In each iteration, we select a list of λ parents out of the current population. These parents are paired, undergo a uniform crossover and a half-normal mutation (cf. Section 6.4.2). The resulting children are then fitted and the best performing μ children are selected for the next population. Additionally, we select a number of high-performing parents equal to the number of rules the NS is expected to produce within one RD phase as part of the new population (*elitism*). Performance of an individual can be based on novelty alone or on a combination of fitness (as used in the ES; cf. Section 6.4.2 and Section 4.1) and novelty, e.g. a linear combination.

For the novelty of a rule, we compare its match set and the match sets of the other rules. In this section, we experiment with two different approaches at the selection of those other rules. Previously [Hei+22a], we chose the rules in the pool and those in the current NS population, where we would compare with the other children for the selection or the parents for elitism, respectively. In the remainder of this section, we refer to this as *NS-P*, with the *P* indicating that a comparison of past populations is only based on rules in the pool or "*pool-only*". Additionally, we investigate another option called *NS-G*, which we first presented in [Hei+23c]. In this approach, we use a more traditional archiving technique and compare with this archive and the current population (either children or parents, as previously). When initially starting each RD phase, the entirety of the pool is copied into the archive. Therefore, NS-G automatically encompasses all comparisons made in NS-P. In each generation, ρ (which is subject to

Algorithm 5 Rule Discovery with NS

```
1: procedure DISCOVER RULES(elitist)                         ▷ (based on $\mu$, $\lambda$)-ES
2:     population ← $\emptyset$
3:     archive ← pool
4:     for $i \leftarrow 1, \mu$ do
5:         population ← population ∪ INIT RULE(elitist)
6:     end for
7:     for $i \leftarrow 1,$ n_iter do
8:         children ← $\emptyset$
9:         parents ← SELECTION(population, $\lambda$)                ▷ Select $\lambda$ parents
10:        for $k \leftarrow 1, \lambda/2$ do
11:            child_1, child_2 ← CROSSOVER(parents[$k$], parents[$k + \lambda/2$])
12:            children ← children ∪ MUTATE(child_1) ∪ MUTATE(child_2)
13:        end for
14:        EVALUATE_FITNESS_AND_NOVELTY(children)
15:        best_children ← SELECT_BEST_CHILDREN(children, $\mu$)
16:        if NS-G then
17:            archive ← archive ∪ SELECT_BEST_CHILDREN(children, $\rho$)
18:        end if
19:        best_parents ← SELECT_BEST_PARENTS(parents)                      ▷ Elitism
20:        population ← best_children ∪ best_parents
21:    end for
22:    return $n$ best rules from population
23: end procedure
```

hyperparameter tuning) rules are selected from the children and put into this archive, giving NS-G its name: "*generational*".[17] Thus, the archive grows with each generation and—in contrast to NS-P—NS-G is discouraged from exploring regions it has already explored in this RD phase. Note that, after an RD phase completes, the archive is reset to the then-current pool. This avoids a heavy computational load on one hand but is also more in line with the idea of independent optimization only based on the current solution to the problem. Would we account for previous RD phases, we might hinder the incorporation of information from this solution, as some region might have been touched but that rule was not added to the pool as it was unimportant at the time but did become beneficial to further explore now.

The novelty score assigned to a rule is the average Hamming-distance between its match set and its k nearest neighbours' (most similar rules) from the respective comparison set. A value typically encountered for k with other NS applications in literature—e.g. [LS10]—is 15, although we tune between 10 and 20.

[17]As the comparison is rather expensive, we cannot add all rules to the archive, but given that many children will be non-useful anyhow, a limitation should have little ill effects.

After a set number of iterations, we add a predefined number of rules from the current population to the pool and conclude this phase. Which rules get added can be randomized or based on the highest novelty(-fitness combination).

In addition to the basic NS, we implemented and experimented with two variants of NS: *Minimal Criteria Novelty Search* (MCNS) [LS10; Leh12] and *Novelty Search with Local Competition* (NSLC) [Leh12]. For both variants, we also applied and benchmarked the "generational" and "pool-only" options for the novelty calculation.

MCNS imposes additional pressure on the search to explore less vividly and focus more on rules that at least fulfil some minimal requirement. In our experiments, we set the minimal criterion to a minimum number (tuned between 5 and 15) of examples from the training data having to be matched by the rule to become viable. However, we did also impose that at most one fourth of the population should be removed because they missed the minimal criterion to prevent collapsing gene pools. We also use progressive minimal criteria novelty search [GUC12], itself based on MCNS, as an option for combining fitness and novelty as the objective and introducing increased fitness pressure by using a dynamic fitness threshold as an additional criterion. Here, all individuals that do exhibit a fitness worse than the median fitness are removed automatically in each iteration of the search. This approach is not tied to MCNS and can be used in all three variants.

NSLC introduces a localised fitness-based pressure on the new generation. The idea is that, within a neighbourhood of similar rules (based on their behaviour and not their position in the search space), the rules that exhibit high fitnesses should be chosen. A rule's novelty score gets increased by a factor of $\frac{b}{\kappa}$, where b is the number of individuals within the neighbourhood specified by κ that have a worse fitness than the rule currently evaluated. We tune κ in the same range as k, as this does also specify a neighbourhood of rules this rule is in competition with.

One potential disadvantage for the explainability of the NS-like approaches is that rule selection is no longer solely based on independent metrics (fitness) but rather on the independent fitness and the highly dependent (on other rules) novelty score.

6.4.3 Evaluation

To examine the differences between the rule discovery methods and to find the most versatile strategy, we evaluated those strategies within SupRB on several regression datasets.

Experiment Design

The experimental design of this section follows those of previous papers on SupRB [Wur+22; Hei+22b; Hei+22a; Hei+23c; Hei+23d; Hei+24] and—therefore of course—the other experiments in this thesis.[18] The target is standardized and input features are transformed into the range $[-1, 1]$. While these transformations are reversible, they improve SupRB's training process as they help preventing rules to be placed in regions where no sample could be matched and remove the need to tune error coefficients in fitness calculations, respectively. Based on our assumptions about the number of rules needed, 32 cycles of alternating rule discovery and solution composition are performed, generating four (or eight in case of NS variants) rules in each cycle for a total of 128 (256 for NS variants) rules. For NS variants, we opted for the generation of twice as many rules as the novelty pressure will by-design lead to unsuitable rules, e.g. overgeneral and overspecific ones, because they are substantially different from the actually useful rules that are well balanced. This of course makes the NS variants substantially slower in terms of run time, with a non-linear increase with the number of rules. Additionally, the GA is configured to perform 32 iterations with a population size of 32 and 5 elitists. To tune some of the more sensitive parameters, we performed a hyperparameter search using a Tree-structured Parzen Estimator in the Optuna framework [Aki+19] that optimizes *solution fitness* on 4-fold cross-validation. We tuned the optimizers on each dataset independently for a fixed tuning budget of 360 core hours. The final evaluation, for which we report results in the next subsection, uses 8-split Monte Carlo cross-validation, each with 25% of samples reserved as a validation set. Each learning algorithm is evaluated with 8 different random seeds for each 8-split cross-validation, resulting in a total of 64 runs per dataset and algorithm.

Table 6.2: Overview of the six regression datasets the eight rule discovery approaches for SupRB are compared on.

Name (Abbreviation)	n_{dim}	n_{sample}
Combined Cycle Power Plant (CCPP) [KT12; Tüf14]	4	9568
Airfoil Self-Noise (ASN) [BPM89]	5	1503
Concrete Strength (CS) [Yeh98]	8	1030
Energy Efficiency Cooling (EEC) [TX12]	8	768
Physicochemical Properties of Protein Tertiary Structure (PPPTS)	9	45739
Parkinsons Telemonitoring (PT) [Tsa+09]	18	5875

An overview of the used datasets in this set of experiments in given in Table 6.2, including sample size and dimensionality. As with the previous articles on RD [Hei+22a; Hei+23c], we evaluate on datasets part of the UCI Machine Learning Repository [DG17]. The Combined Cycle Power Plant (CCPP) [KT12; Tüf14] dataset shows an almost linear relation between features and targets and can be acceptably accurately predicted using a single rule. Airfoil Self-Noise (ASN) [BPM89] and Concrete Strength (CS) [Yeh98] are both highly non-linear and will likely need more rules to predict the target sufficiently. The CS dataset has more input features than ASN but is easier to predict overall. Energy Efficiency Cooling (EEC) [TX12] is another rather linear dataset, but has a much higher input features to samples ratio compared to CCPP. It should similarly be possible to model it using only few rules. In addition to these four,

[18]The code to reproduce the experiments can be found in SupRB's experiment repository: https://github.com/heidmic/suprb-experimentation/

which were present in [Hei+23c], we also investigated the performance on Physicochemical Properties of Protein Tertiary Structure (PPTS), which is highly non-linear, and Parkinsons Telemonitoring (PT) [Tsa+09], which is also non-linear and has the highest number of features of all investigated datasets. Overall, we expect these two additional datasets to be harder to approximate than the other four. Both have also already been used in our study on local models (cf. Section 6.1.1). We also included them here to hopefully make more conclusive statements about which optimizer to choose than were possible following [Hei+22a] and [Hei+23c].

Results

In the following, we abbreviate *SupRB using X as its RD method* simply by *X*, e.g. *SupRB using ES as its RD method* is just signified as *ES*. The other optimizers follow accordingly.

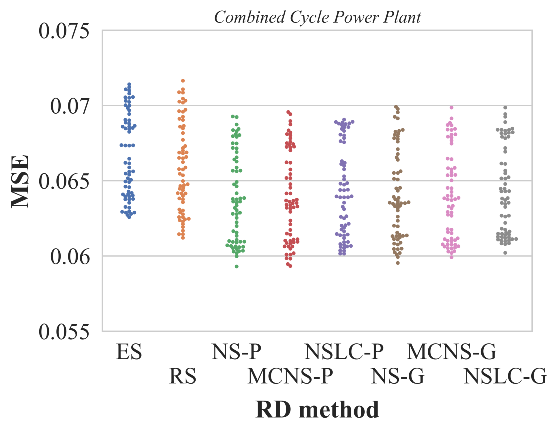
Tables 6.3 and 6.4 give the means and standard deviations of model performances represented by *mean squared errors* (MSEs) (measured using the standardized—individually per dataset—test data) and model *complexities* (measured by the number of rules in the final elitist) achieved by the eight RD approaches when evaluated—as described in the previous section—on the six real-world datasets.

At a first glance, on five of six datasets, RS shows the clearly worst performance in terms of mean MSE but the models it creates generally show a low(er) model complexity, being even the best one two datasets. ES tends towards smaller models but only has the best error on ASN and is a bit worse, albeit close to the best, on errors for all datasets but PPPTS. The other optimization approaches vary in their results between datasets with no clear tendencies being easily discernible based on the tables alone.

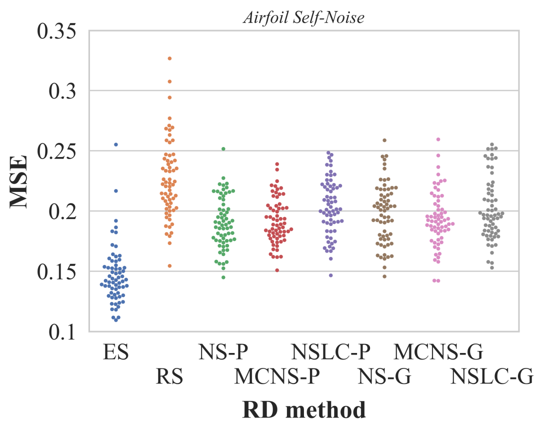
Table 6.3: Mean and standard deviation (over 64 runs, rounded to two decimal places) of *MSEs* achieved by the eight RD approaches on the six datasets. Best entry in each column (if one exists) marked in bold.

	CCPP	ASN	CS	EEC	PPPTS	PT
ES	0.07 ± 0.0	**0.15 ± 0.02**	0.15 ± 0.04	0.04 ± 0.03	0.63 ± 0.02	0.31 ± 0.03
RS	0.07 ± 0.0	0.23 ± 0.03	0.17 ± 0.05	0.06 ± 0.03	0.62 ± 0.01	0.45 ± 0.03
NS-P	0.06 ± 0.0	0.19 ± 0.02	0.14 ± 0.03	0.04 ± 0.01	0.6 ± 0.01	**0.3 ± 0.03**
MCNS-P	0.06 ± 0.0	0.19 ± 0.02	**0.13 ± 0.03**	0.04 ± 0.02	0.59 ± 0.02	0.3 ± 0.04
NSLC-P	0.06 ± 0.0	0.2 ± 0.02	0.15 ± 0.03	0.05 ± 0.02	0.62 ± 0.02	0.39 ± 0.03
NS-G	0.06 ± 0.0	0.2 ± 0.03	0.14 ± 0.03	0.04 ± 0.02	**0.59 ± 0.01**	0.35 ± 0.03
MCNS-G	0.06 ± 0.0	0.19 ± 0.02	0.14 ± 0.05	**0.03 ± 0.02**	0.59 ± 0.04	0.34 ± 0.03
NSLC-G	0.06 ± 0.0	0.2 ± 0.03	0.15 ± 0.03	0.04 ± 0.02	0.63 ± 0.03	0.38 ± 0.03

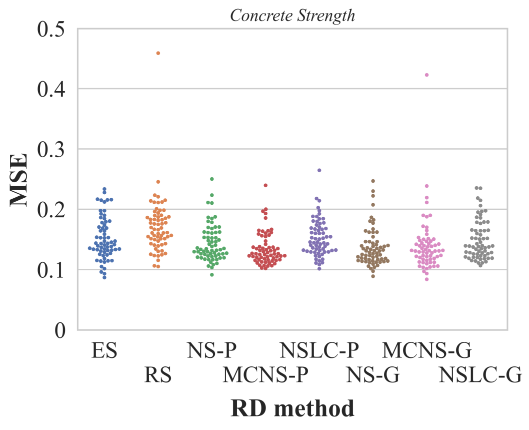
In order to get a better overview of the runs, as well as include more detailed information about the distributions, we created swarm plots to visualize the MSE (Figure 6.7) and model complexity (Figure 6.8) results. For CCPP (Figure 6.7a), we see only very slight differences



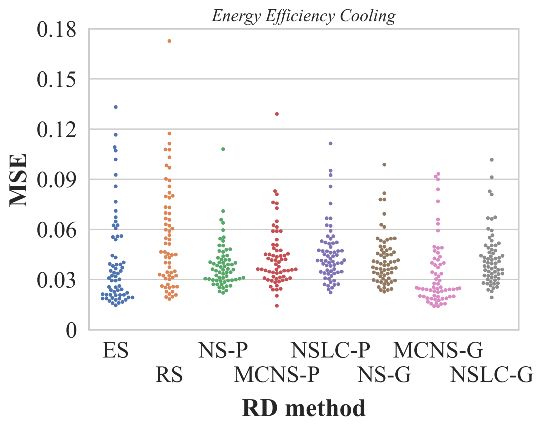
(a) Distribution of runs on CCPP



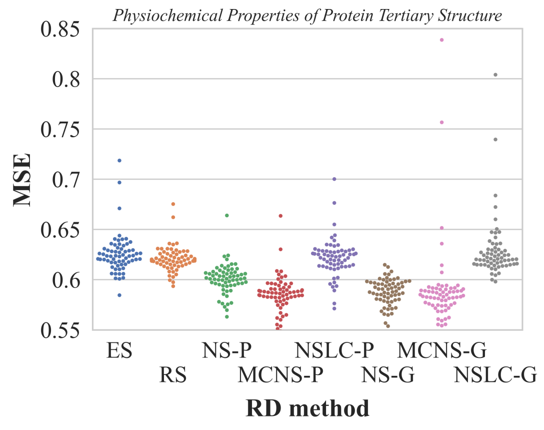
(b) Distribution of runs on ASN



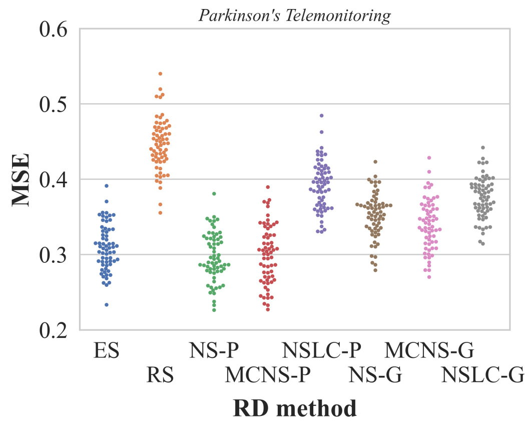
(c) Distribution of runs on CS



(d) Distribution of runs on EEC

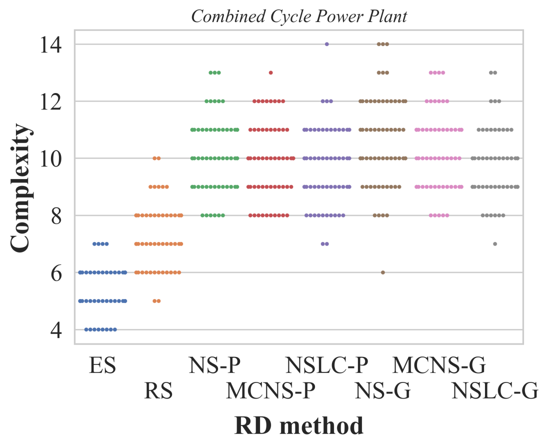


(e) Distribution of runs on PPPTS

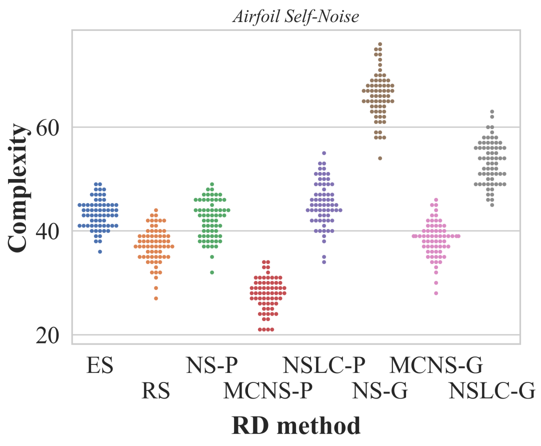


(f) Distribution of runs on PT

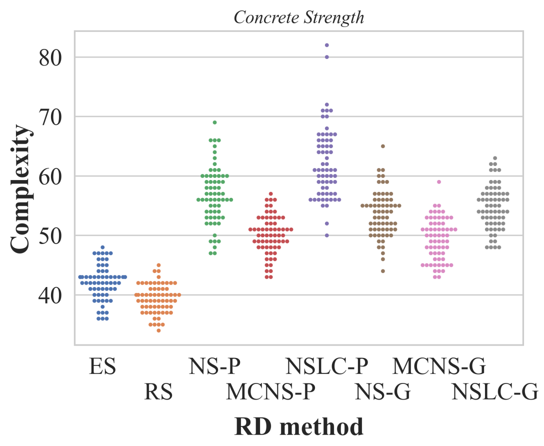
Figure 6.7: Distribution of RD runs' test *errors*. All datasets are standardized with unit variance. Note the different scales, reflecting varying difficulty of the learning tasks.



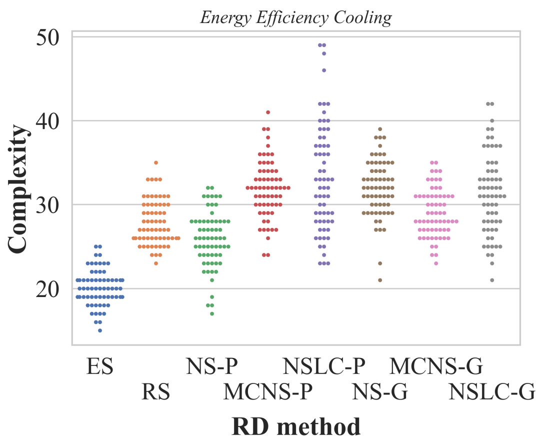
(a) Distribution of runs on CCPP



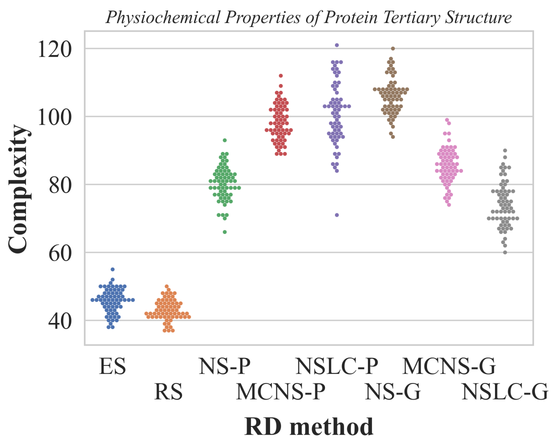
(b) Distribution of runs on ASN



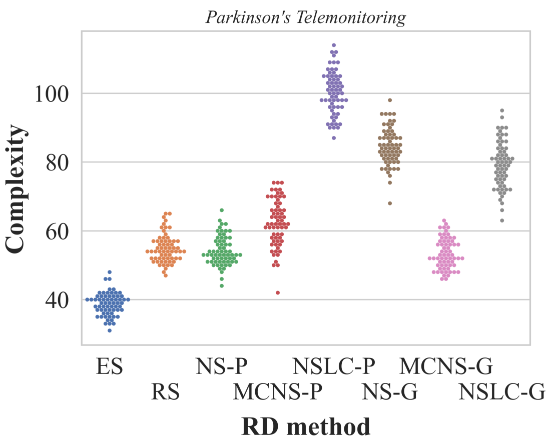
(c) Distribution of runs on CS



(d) Distribution of runs on EEC



(e) Distribution of runs on PPPTS



(f) Distribution of runs on PT

Figure 6.8: Distribution of RD runs' *model complexities*. Note the different scales, reflecting varying difficulty of the learning tasks.

Table 6.4: Mean and standard deviation (over 64 runs, rounded to one decimal) of the *model complexities* achieved over the experiments. Best entry in each column is marked in bold.

	CCPP	ASN	CS	EEC	PPPTS	PT
ES	**5.5 ± 0.8**	43.3 ± 2.8	42.1 ± 2.9	**20.1 ± 2.2**	45.7 ± 3.6	**38.8 ± 3.3**
RS	7.4 ± 1.1	37.3 ± 3.4	**39.3 ± 2.4**	27.9 ± 2.7	**43.1 ± 3.0**	54.9 ± 4.1
NS-P	10.2 ± 1.3	42.2 ± 3.7	56.8 ± 4.8	25.8 ± 3.4	80.4 ± 5.1	54.2 ± 4.3
MCNS-P	10.0 ± 1.3	**27.8 ± 3.1**	50.1 ± 3.3	31.9 ± 3.5	98.2 ± 5.3	62.3 ± 6.8
NSLC-P	9.7 ± 1.4	45.3 ± 4.3	62.2 ± 6.1	33.2 ± 6.5	100.6 ± 9.4	100.3 ± 6.2
NS-G	10.6 ± 1.6	66.1 ± 4.5	53.6 ± 3.9	32.0 ± 3.4	106.1 ± 5.4	84.5 ± 5.5
MCNS-G	10.3 ± 1.3	38.4 ± 3.4	49.4 ± 3.5	29.0 ± 2.9	85.5 ± 5.3	53.4 ± 4.4
NSLC-G	9.7 ± 1.2	53.3 ± 4.1	54.7 ± 3.7	31.3 ± 4.8	74.1 ± 6.6	79.3 ± 6.8

on errors in the novelty search runs, but ES and RS are clearly falling behind. However, one should note that due to the scale at which these errors occur it is unclear if they are of practical significance even if they are statistically different. In terms of model complexity (Table 6.4 and Figure 6.8a), RS is the second best approach, while ES substantially outperforms the others on average at about half the size of the results of NS-G and 57% of the best novelty search variation. Given the questionable validity of worse results on MSE, this could make ES the best overall approach on this dataset. The plot for errors on the ASN dataset (Figure 6.7b) as well as the means in Table 6.3 suggest that ES outperforms all other optimizers and that the novelty search variants are about equal but outperform RS. However, with respect to model complexity (Table 6.4 and Figure 6.8b), we observe that MCNS-P finds models of a much lower complexity, while the mean number of rules in the solutions found by the other optimizers is about equal, with the exception of NS-G and NSLC-G which are double the size of MCNS-P. Where ES's low error and average complexity might indicate that it found it easier to discover rules resulting in another area on the Pareto-front created by error and complexity, it is not really clear what might lead to the performance difference among the other approaches other than a worse capability during the search or a worse interaction with the GA. On CS, we again notice relatively similar performances on errors but MCNS-P has a tighter distribution and lower average (cf. Figure 6.7c and Table 6.3). MCNS-G did show one far off outlier but should otherwise be considered a good candidate for the best approach. Both models were about the same size on average and close to the mean of all models. While RS is best on model size (Figure 6.8c), the second best is ES, which is quite average on error but clearly better than RS. We can assume that ES is finding a better balance of the objectives than the other algorithms, although MCNS-X seemingly do well in this regard, also showing the lowest errors but third and fourth highest complexities. All of these differences are statistically significant, albeit probably not practically significant for errors and maybe not even for model size. For practically relevant model size differences we do not have clear data yet but made the conservative assumption that a difference of three rules should be relevant (cf. [Hei+23c]). The absolute differences of errors are again quite small on EEC (cf. Figure 6.7d), but MCNS-G shows the best performance and if ES was a bit more consistent on some runs it would also be a top candidate. For ES, we can assume that it again made a trade-off between complexity and error as it was almost 50% smaller than MCNS-G on average (cf. Table 6.4 and Figure 6.8d). Overall, the results on PPPTS show relatively high errors for all approaches as this is a relatively difficult dataset (cf. Figure 6.7e). ES, RS, and both NSLC variants are about even with NS and MCNS variants, beating them slightly in errors. There is a small tendency indicating MCNS-X to be a bit better on this dataset. Interestingly, MCNS-G shows the most extensive outliers of all datasets (and more than MCNS-P) but even then all but 4 of the 64 runs are below the median error of ES. Importantly, the models of ES and RS were about half the size of MCNS-G, which was smaller than MCNS-P (cf. Table 6.4 and Figure 6.8e). On the PT dataset, ES clearly beats all generational variants, NSLC-P, and RS on errors (cf. Figure 6.7f). RS falls similarly far behind as it did on ASN. NS-P and MCNS-P are slightly better than ES, with NS-P being more tightly distributed than MCNS-P. ES is still much more compact (Table 6.4 and Figure 6.8f) than the rest at ∼72% of NS-P's models and ∼39% of NSLC-P's, which might again indicate that it found a path towards small yet accurate models that was a bit more on the smaller side but otherwise not too much worse. Whether or not this loss in predictive power for more explainability is an acceptable trade-off for a real-world application of SupRB where it was to predict PPPTS or PT would be for domain experts to decide.

Overall, despite the hints that ES might strike a better balance between objectives than the other RD approaches are able to find, the visual analysis combined with the rounded statistics of mean and standard deviation is not capable of providing us with a conclusive answer regarding which of the RD methods should be preferred on a range of tasks like the ones considered. We thus investigate the gathered data more closely using Bayesian data analysis[19].

As in Section 6.3.2, we start by applying the model proposed by Calvo et al. [CCL18; Cal+19][20] to our data which, for each of the RD methods, provides us with the *posterior distribution over the probability of that RD method performing best* in regards to a specific metric. We apply this model to both the MSE observations, as well as the model complexity observations, and provide box plots (Figures 6.9 and 6.10) which show the most relevant distribution statistics.

For the *MSEs*, we see in Figure 6.9 that MCNS-P and MCNS-G are about equally likely to be on rank one, albeit with a relatively-speaking rather low probability. NS-P is a close third while RS is the least likely on rank one. ES is about equally likely to be the best as NSLC-P and NSLC-G. However, we can not really make any decision based on these results as a typical minimal threshold for automated decision making is 80% (or usually even more) for a single algorithm/approach [Ben+17]. Based on the results, we could conclude that any method other than RS (which seems to be outranked by all) could be a candidate with a small preference for NS-X and MCNS-X.

[19]We deliberately avoid the use of null-hypothesis significance tests due to their flaws and many possible pitfalls—cf. e.g. [Ben+17].

[20]We use the implementation provided by https://github.com/dpaetzel/cmpbayes using 10,000 steps in the MCMC process.

When considering *model complexity* (Figure 6.10), there is an about 75% probability that one of ES and RS performs the best, with ES being the most likely candidate at 45%. While this does also not meet the threshold for automated decision making, we can clearly see that ES beats the novelty search variations. If we redistribute the probability mass of RS, ES is more likely than all others combined to be the rank one algorithm for complexity. Note that the results differ from those in [Hei+23c], where ES was a bit better placed on errors but only on a distant rank three for complexity, where NS-P and MCNS-P took the top ranks. This shift is primarily due to the addition of the new (and more difficult to solve) PPPTS and PT datasets.

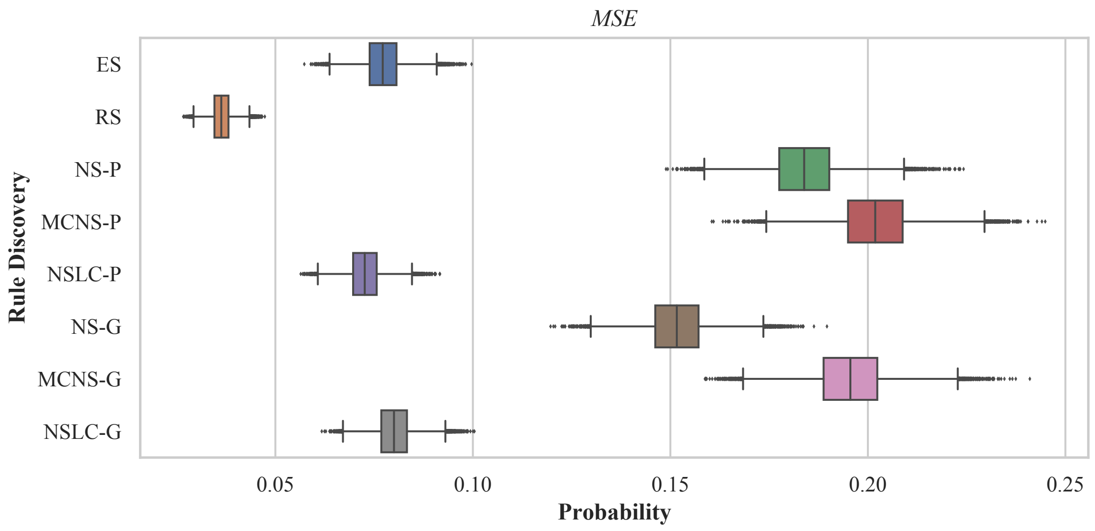


Figure 6.9: Box plot (with standard 1.5 IQR whiskers and outliers) of the posterior distribution obtained from the model by Calvo et al. [CCL18; Cal+19] applied to the MSE data. An RD method having a probability value of q % says that the probability of that RD method performing the best with respect to MSE is q %.

Based on the results of the Calvo model and the close look at the individual distributions, we can not make an automated decision but we can narrow the field a bit. As already stated, RS is probably not the best candidate. Given their low ranking on both complexity and MSE, we can also set aside NSLC-P and NSLC-G. ES being the top candidate on complexity should somewhat offset its lower ranking on MSE. Hence, we will consider it further. NS-P is higher ranked than NS-G on both errors and complexities and Table 6.4 shows practically relevant differences on model sizes, therefore, we remove NS-G from further consideration as well. Thus, our remaining candidates are ES, NS-P, MCNS-P, and MCNS-G.

As our global statistical analysis was insufficient to decide among these four algorithms, we next estimate the likely effect size of choosing among the different algorithms. While a well-trained eye might be able to do this from the visual presentation alone, we employ Corani and Benavoli's Bayesian correlated t-test [CB15][21] to make it more explicit and statistically sound.

[21]We use the implementation provided by https://github.com/dpaetzel/cmpbayes.

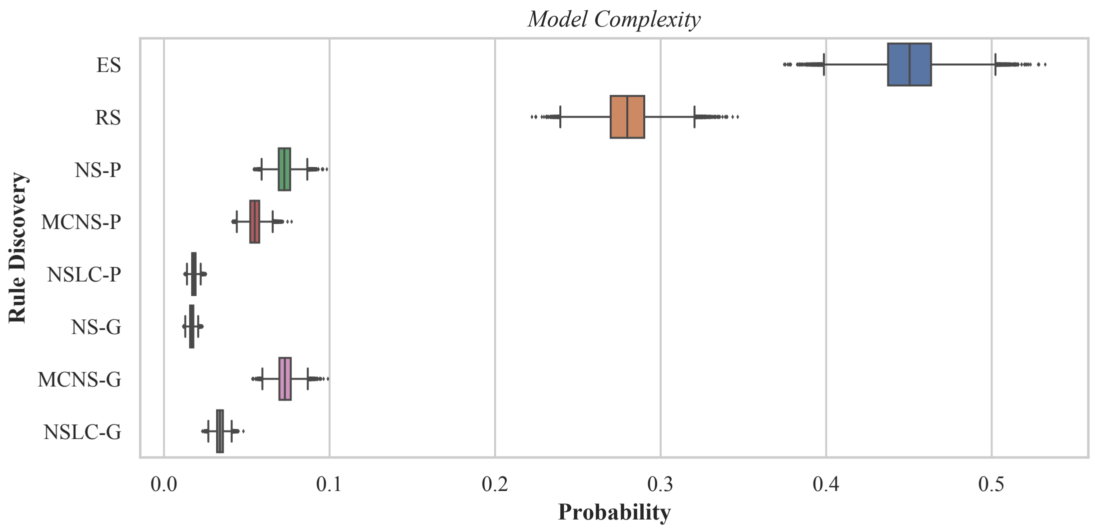


Figure 6.10: Box plot like Figure 6.9 but for the model complexity metric.

For this, we compare ES with the other three candidates, as ES was the remaining baseline and the originally used RD optimizer for which we seek an improvement. The resulting posteriors (given in Figures 6.11, 6.12, 6.13, and 6.14; including 99% high posterior density intervals (HPDIs)) are the *distribution of the differences between the considered metric for ES and for X* (practically, this equates to probability mass above zero indicating ES having a higher (worse) value than X).

For the MSE, we find that the expected effect differences are relatively small, and often we will have runs of ES that perform slightly better than the compared algorithm, while others are worse than it. In Figure 6.11, we visualize the results of the t-test between ES and NS-P for each dataset. For at least 99% of runs NS-P is better on CCPP while being worse on ASN. For each of the other datasets, there is at least some (more than 1%) probability mass on one side of the zero even if the mode of the distribution is on the other. For CS and EEC, we see relatively equal performance and PT is quite close to equal as well. When investigating the effect sizes more closely, we see that despite the vastly varying differences in absolute MSEs on each of the datasets achieved by the runs, the differences are almost on the same scale. So even for more complex datasets the algorithms do not drift that much further apart.[22] Comparing ES and MCNS-P (Figure 6.12), the optimizer to most likely be rank one according to the Calvo model (cf. Figure 6.9), paints a very similar picture. There are some minor differences, e.g. the mode being slightly shifted towards a MCNS-P performance on CS and PPPTS, which does explain the result of the Calvo model, but overall, this does not appear practically significant (and likely would not be statistically significant based on null-hypothesis testing either). Testing ES against MCNS-G (Figure 6.13) again results in a very similar visualization but ES is slightly worse on EEC and better on PT this time.

[22]CCPP is an obvious exception to this but this dataset is very easy to solve as it is almost linear.

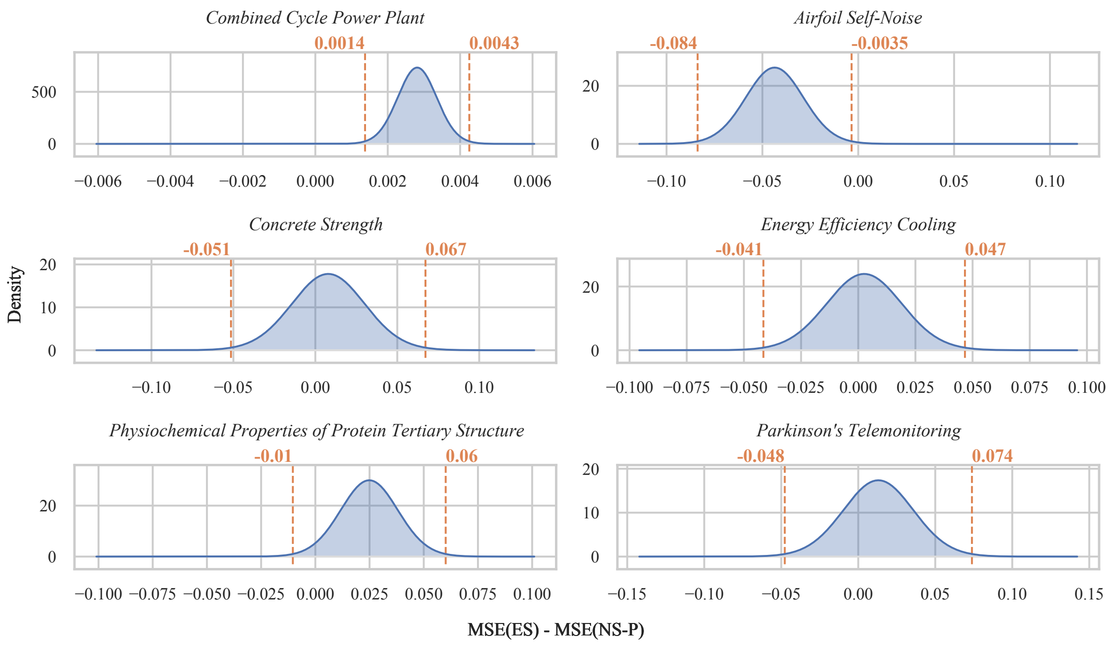


Figure 6.11: Density plot of the posterior distribution obtained from Corani and Benavoli's Bayesian correlated t-test [CB15] applied to the difference in MSE between ES and NS-P. Orange dashed lines and numbers indicate the 99 % HPDI (i.e. 99 % of probability mass lies within these bounds). HPDI bounds are rounded to two significant figures. Effectively, this indicates how likely a specific difference between one run of each RD method will be (practically, probability mass above zero equates to ES having a higher (worse) value than NS-P).

In Figure 6.14, we show the results of the t-test between ES and MCNS-P on model complexity. We include a region of practical equivalence (ROPE; in green). This effectively means that for any probability mass within this region, we consider equal performance rather than giving it to one of the algorithms. Therefore, the difference would be practically insignificant as well. We set the upper and lower bound of the ROPE to the mean of the standard deviations of the model size of each of the eight algorithms, which we assume is a number of rules small enough that having to analyse a small percentage of additional rules is causing negligible additional effort but still sensible for the task at hand.[23] Given the large differences in absolute model sizes between datasets, cf. Table 6.4, a relative value appears more sensible. However, we have no priors about how large a good model should be and how much noise is introduced by the different data splits (into train and test data). Therefore, we can not easily determine what differences are negligible. Thus, we assume that the different models will appropriately fit to the training data and form a roughly unimodal distribution in model size (which is supported by the shapes seen in Figure 6.8). We can then use their standard deviations as a measure for

[23]As the model complexity/size is a count of rules and therefore always an integer, we round the ROPE to the nearest integer as well.

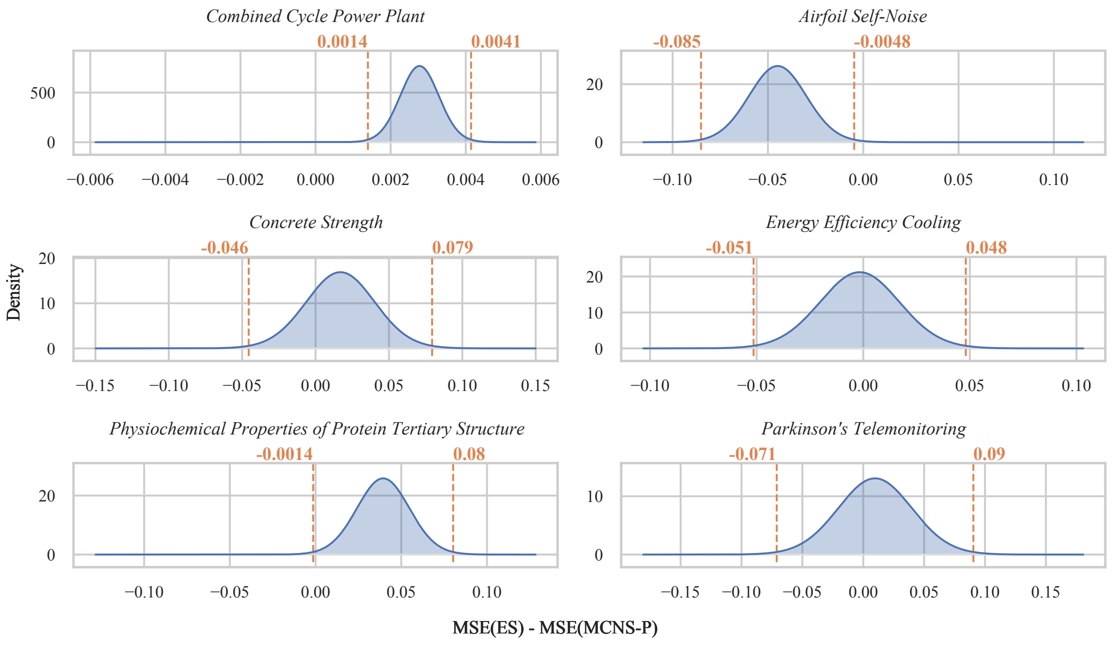


Figure 6.12: Density plot of the posterior distribution like Figure 6.11 for the difference in MSE between ES and MCNS-P.

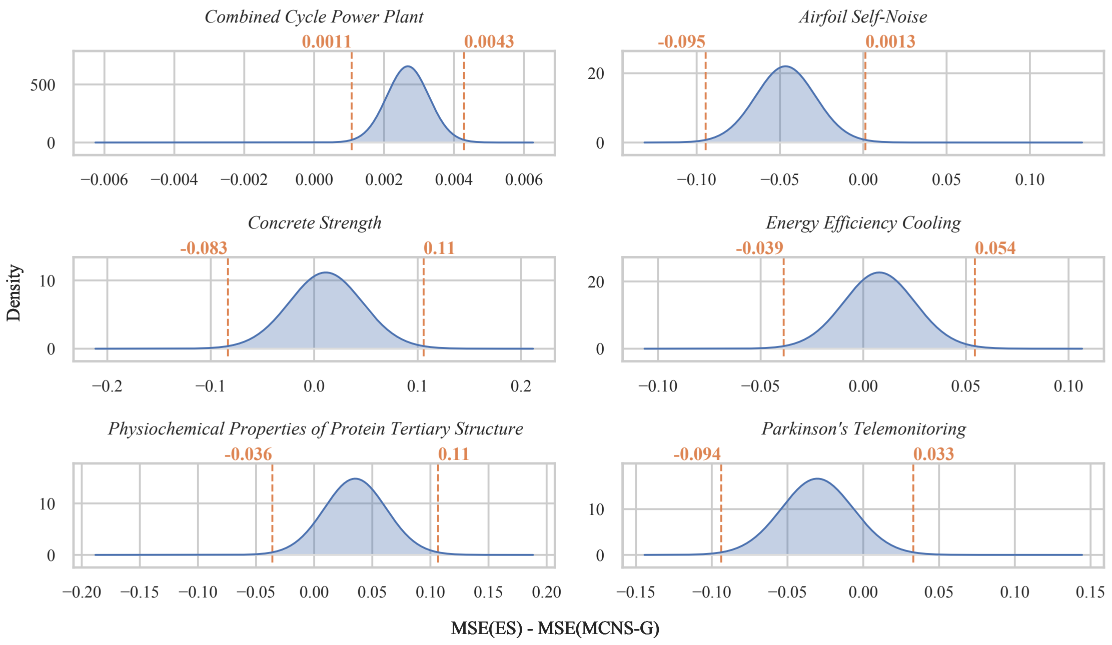


Figure 6.13: Density plot of the posterior distribution like Figure 6.11 for the difference in MSE between ES and MCNS-G.

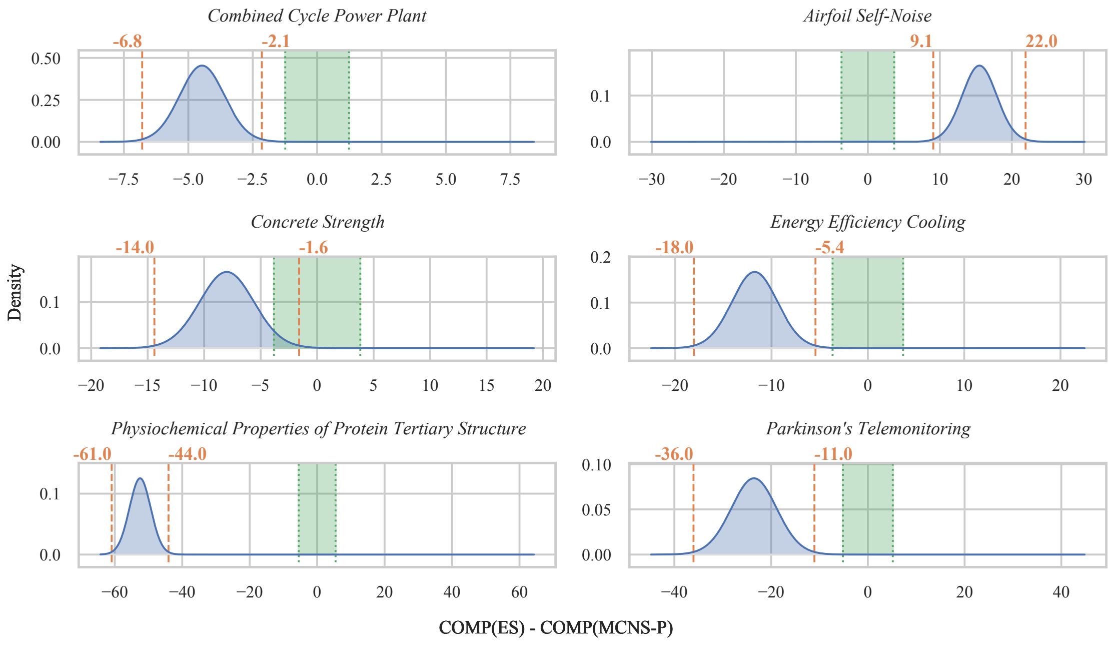


Figure 6.14: Density plot of the posterior distribution like Figure 6.11 for the difference in model complexity between ES and MCNS-P. In green, we show a region of practical equivalence (ROPE).

the range of model sizes we should expect. Fluctuations that are within the mean standard deviation are probably caused by factors that are beyond a model's control and do not signify relevant performance differences. The analysis result is rather definitive (with more than 99% likelihood) that in all cases but a few runs on CS there is a clearly smaller model produced by one of the two compared algorithms. In case of ASN, MCNS-P produces significantly smaller models, while on all other datasets ES shows the smaller models. Even on CS, where a few runs will produce practically equivalent results, over 99% of probability mass is on the side of ES. Comparisons of ES with NS-P and MCNS-G show very similar patterns, although both have considerably large amounts of runs that will produce practically equivalent results on ASN. The relevant plots can be found at https://github.com/heidmic/diss-graphs/tree/main/RD.

We conclude our analysis with a summary of the key takeaways.

- On datasets like the ones we considered, MCNS-P and MCNS-G have the highest probability of being the best options MSE-wise (with NS-P not far off). However, the collected data is not at all conclusive with respect to that. There remains an around 60% chance of MCNS-P and MCNS-G *not* having the best MSEs on such datasets.

- When considering model complexity alone, there is a high probability of ES being the best—however, its MSEs are much less likely among the best than the top algorithms on this metric.

- Randomly sampling rules can be a usable (albeit less powerful) option to generate the rules to build solutions from. While it does fall short on errors, it is almost competitive on model size. Given that some of the computation budget that is spent for tuning ES could be shifted towards generating more randomly-placed rules, it might not be as subpar as it appears now.
- A closer comparison of ES with NS-P, MCNS-P and MCNS-G yielded:
  - ES outperforms NS-P and MCNS-P MSE-wise on ASN but was outperformed on PPPTS.
  - The difference of ES and NS-P/MCNS-P on CCPP is likely not practically significant and the other datasets showed similar performance of the options.
  - ES also outperformed MCNS-G on ASN and probably performs slightly better on PT as well, while MCNS-G shows a less pronounced (than NS-P and MCNS-P) improved error on PPPTS.
  - The difference on CCPP is again not practically significant, while on CS and EEC, ES and MCNS-G show almost similar performance.
  - Interestingly, ES did show higher model sizes than the others on ASN but clearly outperformed them on the other datasets even when considering a sizeable ROPE.

6.5 Solution Composition—Other Metaheuristics

Once a good selection of rules has been discovered and placed into the pool, an appropriate model can be build. As this model serves as a *solution* to a learning task, we termed this process *solution composition (SC)* (cf. Section 4.1 and [Hei+22c]). As comprehensible yet accurate models are the prime focus of SupRB, SC should build solutions accordingly. The following introduces possible SC optimizers other than the Genetic Algorithm (GA) of the original SupRB version, which were subsequently benchmarked similarly to RD in Section 6.4. The alternative optimizers for SC discussed here are all based on various different mainstream metaheuristics (they can all be considered relatively standard and well-established but of course have different notorieties outside of the optimization community), whereas Section 6.6 will introduce some adaptations of the GA itself. This section builds on joint work partially published in [Wur+22], which is an expansion of Jonathan Wurth's Bachelor's thesis which was supervised jointly by Helena Stegherr and myself. However, the experimental setup was expanded and the statistical analysis replaced since the publication.

LCSs—similarly to other rule-based approaches—can suffer from creating more rules than are needed to model the problem sufficiently. This is especially the case on complex learning tasks, where the number of rules is already high and understanding the (hypothetical) optimal rule set is non-trivial. If, additionally, many similar (redundant) rules are part of the rule set, e.g., because several independently operating techniques to generate rules were combined or expert knowledge is incorporated into the training process, making even simple statements about accuracy and usefulness of individual rules is increasingly difficult. Interpreting such rule sets in their entirety gets confusing very quickly, especially if non-trivial mixing models are involved. Reducing the size of rule sets and removing unnecessary or even sub-par rules is, therefore, a sensible choice and directly correlates with an increase in interpretability and often accuracy.

In the context of online single-solution (Michigan-style) LCSs, such approaches are known under the term *rule compaction* and mostly performed as post-processing [TMU13; LBX21b]. The compaction itself is often performed deterministically based on estimated values or by applying heuristic procedures. Some batch multi-solution (Pittsburgh-style) LCSs such as GAssist [Bac04] perform an intermediate rule pruning step, albeit being similarly heuristic. In this section, a more general perspective will be taken on this: selecting a minimal subset of rules while maintaining other objectives like accuracy is a typical model selection [DTY18], or model pruning problem, and such problems can be solved efficiently using metaheuristics. Diao and Shen [DS15] evaluate several nature-inspired metaheuristics such as a GA and ACO on feature selection and find that all are capable of finding good quality solutions. An example of pruning neural network and decision tree ensembles using a GA can be found in [ZWT02] and [ZT03], respectively.

An obvious metaheuristic to apply here is a GA, especially because LCSs are systems originally designed to take advantage of the GA's ability to evolve and enhance its individuals, or rules [Hol92]. Accordingly, SupRB's original version did use a GA to select good subsets of rules. However, other metaheuristics like Particle Swarm Optimization (PSO) or Ant Colony Optimization (ACO) are similarly able to handle hard optimization problems, which includes selection tasks. ACO has also been applied as the primary optimizer in the batch single-solution Ant-Miner [PLF02] (cf. Section 2.4 for other systems and additional information on the Ant-Miner).

The *No Free Lunch Theorem* [WM97] states that no optimization algorithm can be the best on all possible problem classes, which raises an interesting question in this context: which metaheuristic performs best on the task of rule set compaction, or, in the context of SupRB, on *composing solutions*? Therefore, this section specifically compares five different metaheuristics for SC as part of SupRB, which in general tries to generate a rule set (solution) as compact and accurate as possible. The metaheuristics considered here include a GA, ACO, PSO, a Grey Wolf Optimizer (GWO) variant, and the Artificial Bee Colony algorithm (ABC), which all share the characteristic of being population-based and inspired by phenomena in nature.[24] Incorporating a non-evolutionary metaheuristics into an LCS might lead some researchers to argue that the system would not be an LCS anymore in the most narrow, traditional sense, however, the bounds of what can be considered one have shifted to a more general definition over the last years [UB17; Kov12]. We followed this more broad definition as well in [Hei+23a] and Chapter 2.

[24]Of course, many more specialized metaheuristics exist. In this set of experiments, we mostly wanted to investigate whether the switch to another base system would be appropriate, rather than comparing very SupRB specific optimizations within the potential optimizers.

6.5.1 Model Selection in SupRB

This section will recap some aspects of SupRB and set the necessary context for SC: The fundamental concept of SupRB is to split the LCS's usual process of finding a maximally concise and accurate rule set into two subtasks: RD and SC.

RD generates rules which fit a part of the input space using some local model and inserts them into a global population of rules: the (rule) *pool*. To keep interpretability high, the rules simply consist of rectangular bounds and a linear local model (Sections 6.1 to 6.2 discuss possible extensions on these two elements). By default, SupRB employs an ES to discover locally optimal rules, preferring to generate rules in parts of the input space where the in-sample error of the current global solution is the highest. The experiments in this section also utilize an ES for RD. Details on this rule generation process can be found in Chapter 4, Chapter 5, and Section 6.4. The rules in the pool are assumed to have the following properties, regardless of their origin:

- Rules in the pool meet some minimum standard, i.e. they are at least accurate enough so that knowledge about the part of the input space they match can be extracted.
- Some similar rules are part of the pool.
- Rules may (greatly) overlap.
- They are not modified or removed from the pool.

SC is the component that selects a minimal and maximally accurate subset of rules from the pool, mixes them according to some mixing model, and thus creates a valid model of the whole space. For this section we use the error and experience–based model presented in Chapter 4 and further discussed in Section 6.3. This model is also referred to as a (global) *solution* which, in the context and terminology of population-based metaheuristics, equates to an *individual*. SC performs a special kind of model selection, where not explicitly enumerated models are considered, but all possible subsets of rules 2^P from the pool P. Encoding these subsets can easily be achieved by using a binary string, or genome, G, where a 1 at index i encodes that the rule at index i is part of the subset, and 0 encodes that it is not. The following string thus represents the subset of choosing the first, third and last rule in the pool:

$$G = 101000\dots001 \tag{6.5}$$

As rules are only appended to the pool and existing rules are not modified, such a solution vector stays valid by padding it with zeros, although other possibilities exist, e.g. always including new rules (padding with ones) or using a random bit string.

A central problem in SC is that the exact number of rules required for an adequate solution is unknown (or how good that solution should be), so there is no way to know when the global optimum for a given pool is found. The overall goal of SC is therefore finding a good subset of rules, while simultaneously keeping the computational cost reasonable. This also motivated the alternating nature of RD and SC, which makes feedback on the quality of individual rules and the global solution immediate and simplifies the design process. As SC is not only performed at the end but also during the training process, knowledge about the global solution can then again be incorporated into the RD process, for example in the form of choosing regions of the input space that are not yet predicted sufficiently.

6.5.2 Selecting Subsets of the Pool

Searching the entire binary search space 2^N with $N = |P|$ is not feasible, even for relatively small N, and as only a subset that is “good enough” is sought after, the application of metaheuristics on this task is a sensible choice. Problems like Feature Selection [LM07] or Ensemble Pruning [Zho12] are characterized by choosing a minimal subset while simultaneously optimizing other objectives and the same conditions apply here. The task is therefore by definition multi-objective, and a choice must be made if multi-objective metaheuristics should be applied or if a fitness function is constructed that weighs the objectives in some way and that is then optimized by standard (single-objective) metaheuristics. This choice is not obvious and highly depends on the final goal of the optimization. The weighted fitness approach was chosen here, simply because multi-objective optimization would certainly find a whole population of (approximately) Pareto-optimal solutions, but somebody (a human) must then again choose from these solutions according to some criteria. Encoding these criteria into the fitness beforehand makes the selection on the one hand automatically reproducible, and on the other hand enables the metaheuristic to search the space of solutions that fit these criteria in much more detail, as many multi-objective optimizers do not consider a specific ratio of objectives. For the datasets investigated in this section, this combination of objectives into a single fitness was observed to be rational, but applying multi-objective optimization does also hold advantages, as will be shortly discussed in Chapter 7.

The following paragraphs will shortly introduce the five metaheuristics for SC which are compared in this section. Of these metaheuristics, only the GA operates on a binary space in its original version, so the other metaheuristics are binary adaptations of their original operating principles. Each metaheuristic contains some interchangeable components, or operators, sometimes several of which will be presented and chosen as part of hyperparameter tuning on a dataset basis. All of these metaheuristics are population-based, so they operate on a population I of individuals, or global solutions, and therefore define the population size as a parameter.

Genetic Algorithm

The Genetic Algorithm (GA) [Hol92; Mir19; Kra17] defines three components, namely *selection*, *crossover*, and *mutation*. The selection method is chosen from either *roulette wheel*, *tournament* with size k, *linear rank*, or *random* selection. Two genomes are combined using either n-point or uniform crossover, with a crossover rate of 90 %. The children are subsequently mutated by flipping each bit in the genome with a probability given by a constant mutation rate. Also, several elitists from the previous iteration are copied into the new population.

Ant Colony Optimization

Ant Colony Optimization (ACO) [SH98; DD99; DBS06] lets artificial ants traverse a solution graph and construct a solution by locally choosing which edge to take next. Approaches for binary problems typically define two mutually exclusive nodes l for every bit, where traversing one node means a 1 and the other one a 0. The solution construction components chosen here include a *binary* method [Wan+16], which decides on the selection of rules by their order of insertion into the pool. Specifically, pheromones τ are deposited for every rule, with the relative fitness ΔF of a rule used to calculate the heuristic value η:

$$\Delta F(r) = \frac{1}{2} \cdot \frac{F(r) - \min_{s \in P} F(s)}{\max_{s \in P} F(s) - \min_{s \in P} F(s)}, \tag{6.6}$$

$$\eta_r(l) = \begin{cases} 1 - \Delta F(r), & l = 0 \\ 1 + \Delta F(r), & l = 1 \end{cases}. \tag{6.7}$$

The value is therefore endorsing the selection of rules with fitness higher than the mean. Ants start at the first rule and select either the 1 or 0 node l for every rule r, solely based on $\tau_r(l)$ and $\eta_r(l)$, until the last one is reached. This virtually ignores the interactions between rules, which is why the *complete* solution construction [KN13; XC20] operates on a complete graph, with edges existing between every rule (or rather their subnodes l). For this method, ants traverse the rules in random order and either select or deselect a rule r based on $\tau_{r,r}, \eta_{r,r}$, and the $\tau_{s,r}, \eta_{s,r}$ of every rule s that was selected by the ant up to this point. The heuristic value $\eta_{i,j}$ is a combination of the relative fitness of rule j and the relative overlap ΔV of the bounds of rule i and j:

$$\Delta V(i,j) = \frac{V(b_i \cap b_j)}{\min\{V(b_i), V(b_j)\}}, \tag{6.8}$$

$$\gamma_{i,j}(l) = \begin{cases} 1 - \Delta V(i,j), & l = 0 \\ 1 + \Delta V(i,j), & l = 1 \end{cases}, \tag{6.9}$$

$$\eta_{i,j}(l) = \sqrt{\kappa_{i,j}(l) \cdot \gamma_{i,j}(l)}, \tag{6.10}$$

where $\kappa_{i,j}(l)$ is defined using the heuristic value of the binary solution construction, i.e. using Equation (6.6), and $V(b_r)$ is the volume of the bounds of rule r. Pheromones endorsing the selection are deposited for (all pairs of) selected rules by the n best ants in this iteration and pheromones inhibiting selection are deposited for rules that were not selected. Additionally, ACO traditionally defines α, β weighting τ and η, and the evaporation rate ρ of τ.

Grey Wolf Optimizer

Grey Wolf Optimizer (GWO) [MML14] updates its population by stochastically mixing them with the three best individuals. It is binarized here, using the two approaches introduced in [EZH16]. Both rely on the modified sigmoid, or rather logistic, function

$$\acute{\sigma}(x) = \frac{1}{1 + \exp(-10(x - 0.5))}, \tag{6.11}$$

and a comparison of a continuous value $x_i \in [0, 1]$ for rule i with a uniformly distributed random value rand $\in [0, 1]$:

$$G_i = \begin{cases} 1 & \text{if } \acute{\sigma}(x_i) \geq \text{rand} \\ 0 & \text{otherwise} \end{cases}. \tag{6.12}$$

The *sigmoid* component performs this binarization at the very end, while *crossover* updates the individuals mostly in binary and uses a uniform crossover mechanism to combine them. The number of the best individuals considered for updates is traditionally set to three for this metaheuristic (α, β, δ), but an extension to an arbitrary count is used here.

Particle Swarm Optimization

Particle Swarm Optimization (PSO) [KE95; PKB07] emulates the real-world movement of particles in the air for adapting and moving solutions in the search space. Three ways to update these particles are considered here. The *sigmoid* update [KE97] uses Equation (6.11) and Equation (6.12) similarly to GWO's sigmoid component to create genomes from continuous particle positions and defines two parameters, b and c. The same approach is chosen for *sigmoid quantum*, with the difference of using a Quantum PSO [SFX04] as underlying metaheuristic. Note that the position and velocity of particles, in contrast to GWO, are left continuous. *Binary quantum* [Wu+18; Wu+19] is a completely binary version of QPSO, which performs a kind of tournament selection to select particles, or *attractors*, and performs partial crossover using a special learning probability p_{learn}. All approaches incorporate some α into the particle update, which is decreased from $a_{\max}$ to $a_{\min}$ over the optimization process.

Artificial Bee Colony Algorithm

The Artificial Bee Colony algorithm (ABC) [KB07; Kar+14] constructs new solutions by combining individuals with others selected by random or roulette wheel selection. New solutions only replace their parents if the newly generated individual performs better. In addition, ABC occasionally resets stagnated individuals after a certain number of iterations given by the trials limit and creates a new random solution. The *sigmoid* approach uses the traditional continuous combination with Equations (6.11) to (6.12), while *bitwise* [JDK14] relies on bitwise AND, OR, and XOR operations to combine two genomes. Besides this recombination of solutions, individuals also explore their neighbourhood by adding a probabilistic random vector to the crossover result. The *dimension flip* update [San+19] performs a partial crossover of two solutions, with the percentage of crossed dimensions given by the flip rate r_{flip}.

6.5.3 Evaluation

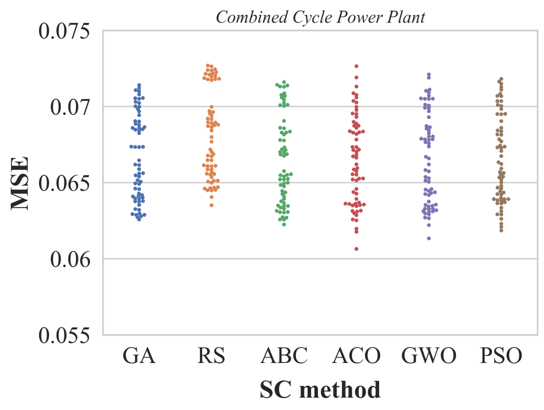
This section first describes the experimental setup we use to evaluate the performance of the different SC optimizers. Then, we present our results and the rigorous statistical analyses we performed to determine which SC optimizer we should recommend for further usage.

Experimental Setup

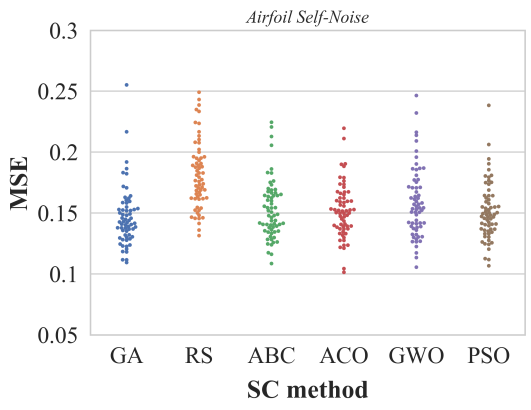
The five metaheuristics presented in Section 6.5.2 are integrated into SupRB itself[25], which is sufficiently modularized to allow easy replacement of RD and SC optimizers and the other major components discussed in this chapter. The general experimental setup follows Chapter 5 and the other sections in this chapter:[26] Input features are transformed into the range $[-1, 1]$, while the target is standardized. The evaluation uses 8-split cross-validation, each with $25\,\%$ of samples reserved as a validation set. Each metaheuristic is evaluated eight times using different seeds for each 8-split cross-validation, resulting in a total of 64 runs. In addition to experimentation with the five optimizers, a Random Search (RS) is performed as an additional baseline (besides the original GA), finding rule subsets using an identical experimental setup. RS randomly creates new individuals in each step, ultimately returning the best individual found.

SupRB performs 32 RD-SC cycles in total, generating four rules in each cycle for a total of 128 rules. For RD, the ES is used, cf. Section 4.1 and Section 6.4.2. All of SupRB's (including the ES and the respective SC optimizer) hyperparameters are tuned for every dataset independently. The SC metaheuristics are configured to perform 32 iterations with a population size of 32 and RS similarly evaluates $32 \cdot 32 = 1024$ individuals per cycle. All hyperparameter tuning is done using a Tree-structured Parzen Estimator implemented in the Optuna framework[27] [Aki+19], optimizing the *solution fitness* on 4-fold cross-validation with a fixed budget of up to 360 core hours.

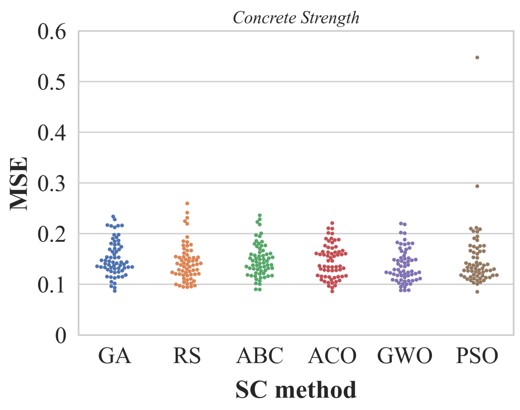
Similarly to RD, we evaluate on six real-world datasets presented in Table 6.2 and described in the experiment design section of Section 6.4.3. The work by Wurth et al. [Wur+22], which is the foundation of this section, only evaluated the SC approaches on the first four of these datasets and had a slightly different but overall similar setup.



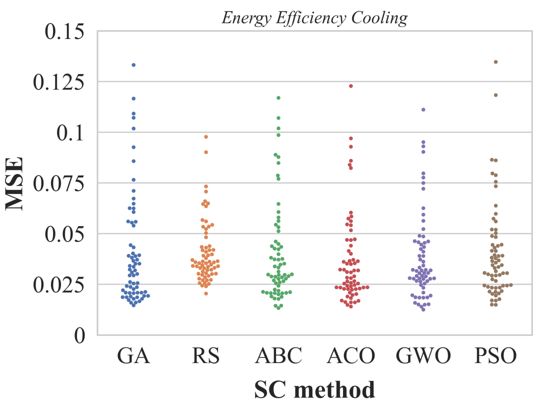
(a) Distribution of runs on CCPP



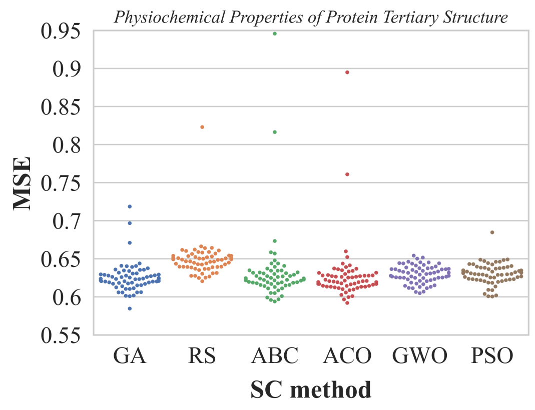
(b) Distribution of runs on ASN



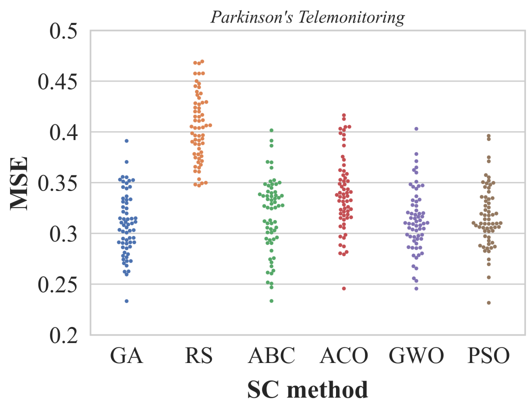
(c) Distribution of runs on CS



(d) Distribution of runs on EEC

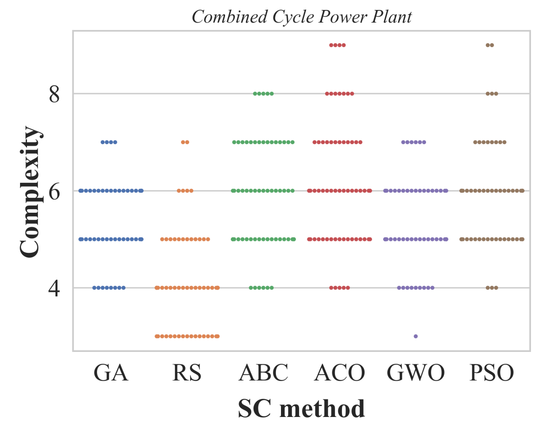


(e) Distribution of runs on PPPTS

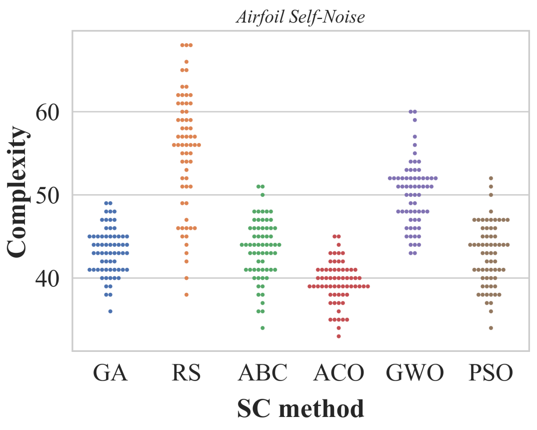


(f) Distribution of runs on PT

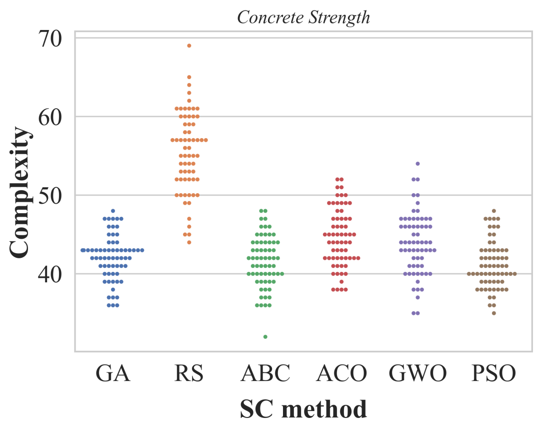
Figure 6.15: Distribution of SC metaheuristic runs' *errors*. Note the different scales, reflecting varying difficulty of the learning tasks.



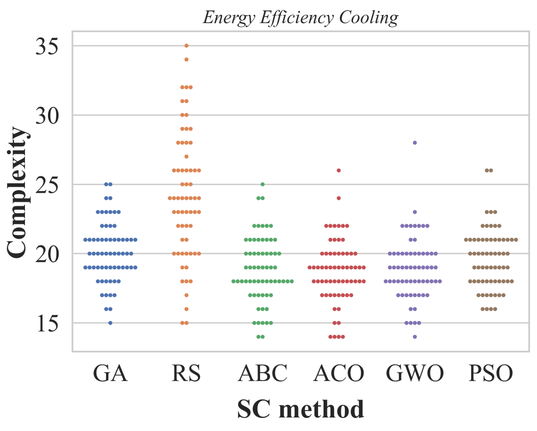
(a) Distribution of runs on CCPP



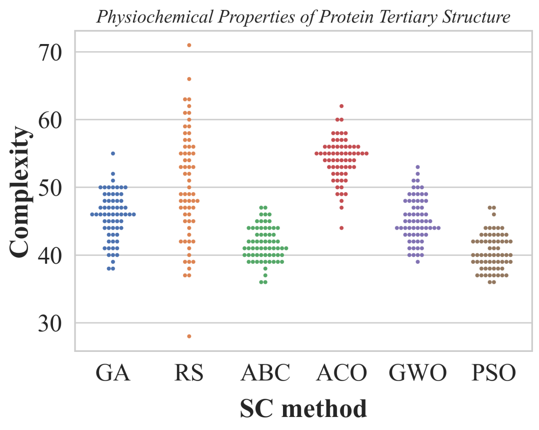
(b) Distribution of runs on ASN



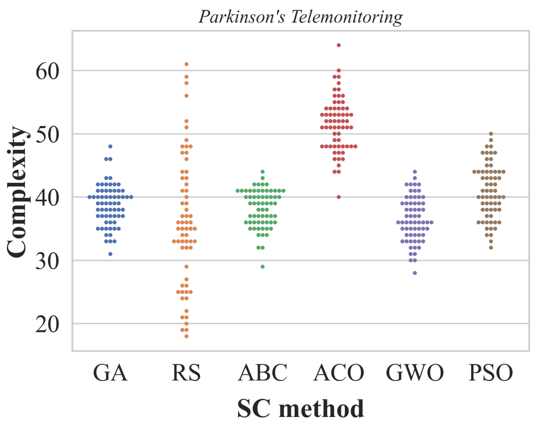
(c) Distribution of runs on CS



(d) Distribution of runs on EEC



(e) Distribution of runs on PPPTS



(f) Distribution of runs on PT

Figure 6.16: Distribution of SC metaheuristic runs' *model complexities*. Note the different scales, reflecting varying difficulty of the learning tasks.

Results and Discussion

Similar to the previous sections, in the following, we abbreviate *SupRB using X as its SC method* simply by *X*, e.g., *SupRB using ABC as its SC method* is just signified as *ABC*. The other optimizers follow accordingly.

The experiments show that all five metaheuristics introduced in Section 6.5.2 are able to compose good solutions and are very similar in their performance. For most of the datasets, they also clearly perform better than the baseline (RS). Figure 6.15 shows the distribution of the errors achieved on the testing data across the 64 runs per algorithm per dataset. Note that we effectively re-use the results achieved by *ES* during our experiments on RD (Section 6.4) as it used a GA as its SC component and is thus identical to this set of experiments where we investigate a GA for SC that uses an ES for RD. We also set the same parameter tuning ranges and the overall experimental setup is the same resulting in identical experimental results due to algorithm and data seeding. We can notice that some runs produced significant outliers to the worse, but the general distributions are quite similar between approaches both in shape and in location. From the distributions on errors alone, it is almost impossible to determine the best algorithms and make any statement other than that the metaheuristics are indeed better than RS.

Figure 6.16 shows the distribution of the model complexities of the approaches on each dataset. RS tends towards wider distributions. ACO is worse than the others on PPPTS, on PT, and, slightly, on CS, but better on ASN. GWO is worse on ASN but generally shows similar means than the median sized runs on the respective dataset.

As it is hard to make a definitive statement based on the distributions, we again employ Bayesian statistical testing. Similar to Section 6.3.2 and Section 6.4.3, we start by applying the model proposed by Calvo et al. [CCL18; Cal+19][28] to our data which, for each of the SC methods, provides us with the *posterior distribution over the probability of that SC method performing best* in regards to a specific metric. We apply this model to both the MSE observations as well as the model complexity observations and provide box plots (Figures 6.17 and 6.18) which show the most relevant distribution statistics.

In Figure 6.17, we see that the Calvo model confirms our previous analysis on the raw distributions of errors. RS falls behind the other algorithms and is unlikely to perform best. However, the other algorithms are only separated by a few percentage points. If at all, we can see a small tendency towards GA which is about 2% points more likely to perform the best on datasets similar to the ones investigated than ABC, which is the second most likely algorithm to achieve

[25]The newest version of SupRB's implementation is always found at https://github.com/heidmic/suprb. The version used for all experiments in this dissertation is long-term archived at https://doi.org/10.5281/zenodo.14181292.

[26]The code to reproduce the experiments can be found in SupRB's experiment repository: https://github.com/heidmic/suprb-experimentation/

[27]https://optuna.readthedocs.io/

[28]We use the implementation provided by https://github.com/dpaetzel/cmpbayes using 10,000 steps in the MCMC process.

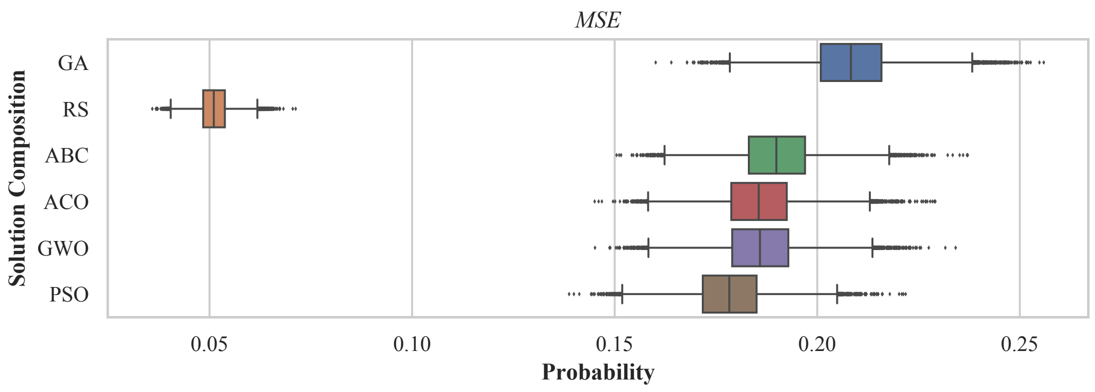


Figure 6.17: Box plot (with standard 1.5 IQR whiskers and outliers) of the posterior distribution obtained from the model by Calvo et al. [CCL18; Cal+19] applied to the MSE data. An SC method having a probability value of q % says that the probability of that SC method performing the best with respect to MSE is q %.

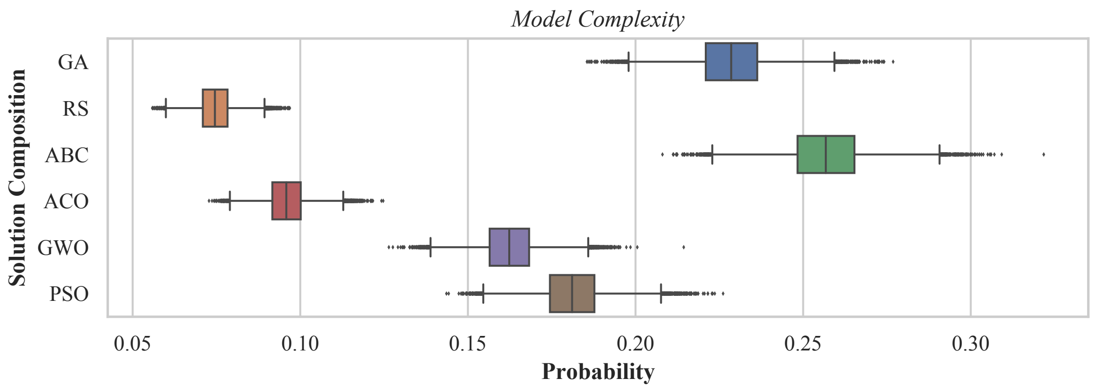


Figure 6.18: Box plot like Figure 6.17 but for the model complexity metric.

rank one performance on MSEs. The weakest of the metaheuristics, PSO, is only 3.5% points less likely to be on rank one than the GA.

In terms of complexities (cf. Figure 6.18), we see a slightly clearer picture but far from typical thresholds (above 80%) for automated decision making [Ben+17]. RS is still unlikely to perform best. ACO is also not likely a top performing approach. ABC is most likely on rank one while the GA is second most likely. However, I want to stress that this is by no means a clear generalist statement.

As ABC and the GA are the two most likely approaches to have the best performance on both objectives, we will now compare their expected performance difference per dataset using Corani and Benavoli's Bayesian correlated t-test [CB15].[29] Figure 6.19 shows the resulting

[29]We use the implementation provided by https://github.com/dpaetzel/cmpbayes.

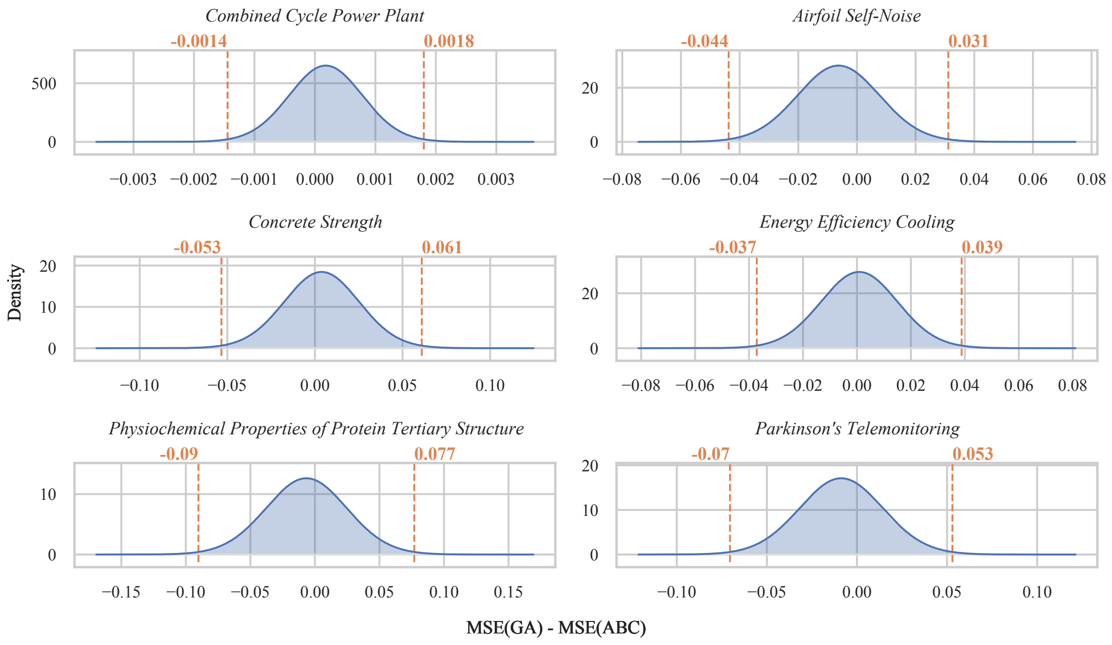


Figure 6.19: Density plot of the posterior distribution obtained from Corani and Benavoli's Bayesian correlated t-test [CB15] applied to the difference in MSE between GA and ABC. Orange dashed lines and numbers indicate the 99 % HPDI (i.e. 99 % of probability mass lies within these bounds). HPDI bounds rounded to two significant figures. Effectively, this indicates how likely a specific difference between one run of each SC method will be (practically, probability mass above zero equates to GA having a higher (worse) value than ABC).

distributions of the test for the MSEs. As we can easily see, they are quite balanced overall. The difference on CCPP is not practically relevant for at least 99% of cases. While we can see a very slight edge for ABC on CS and an even smaller one on EEC, GA is expected to have slightly lower errors more often on ASN, PPPTS, and PT, but even here 99% of the probability mass is still expecting a difference in error below 0.1 standard deviations of the respective datasets. Thus, we can comfortably state that ABC and GA are not only almost equally as likely to rank best globally on sets of similar datasets as the six we tested on, but also that we can expect them to perform almost equally on every individual dataset. Figure 6.20 shows the distributions of differences on model complexity according to the test. In addition to the distributions themselves, we set up a region of practical equivalence (ROPE). For probability mass that falls within this region, we consider the performance equal, regardless of which side of the zero the mass is on. Consequently, only the mass left or right of that region should be considered towards different performance of algorithms. We set the region, as we did in Section 6.4.3, to a dynamic bound based on the mean standard deviation of the algorithms (cf. the previous section for a detailed explanation of our reasoning). We can see that only for CCPP, EEC, and PPPTS a significant amount of probability mass is outside of the ROPE. Of

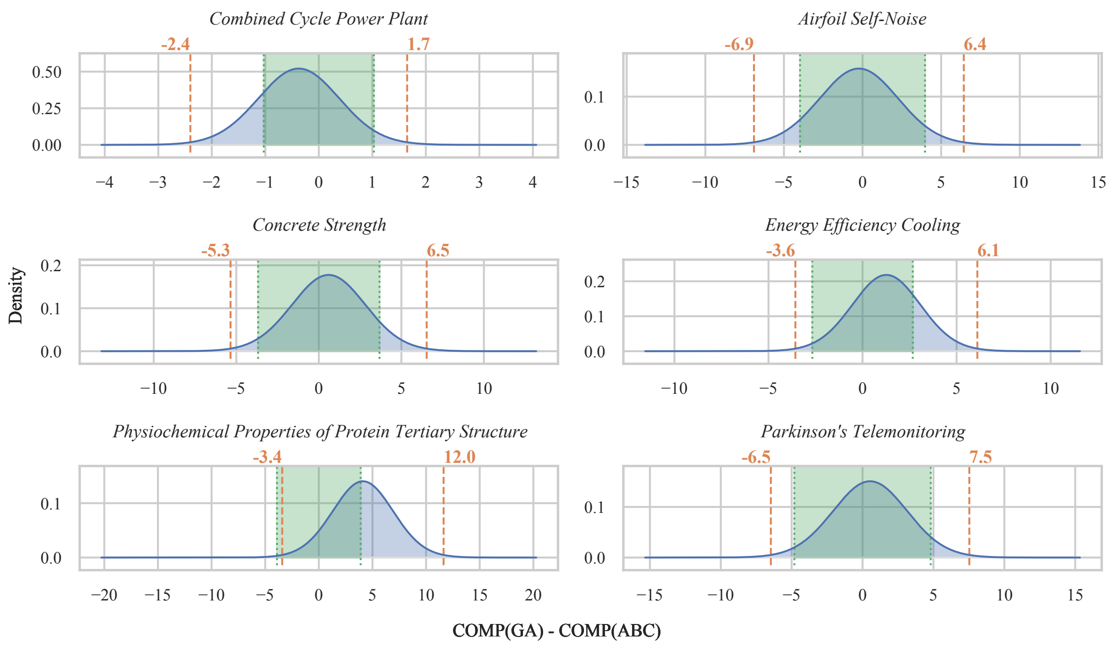


Figure 6.20: Density plot of the posterior distribution like Figure 6.19 for the difference in complexity between GA and ABC. In green, we show a region of practical equivalence (ROPE).

these three, PPPTS is the only one that has more than half of the mass outside of the ROPE (and towards ABC showing better performance). Thus, the only case where one algorithm practically significantly outperforms the other is on PPPTS, where ABC is expected to perform better than or equal to the GA in more than 99% of runs. We expect ABC to have about 5 rules less, which is about 1/9th of the GA's average model size.

ABC's good performance could be tied to its strong exploration capabilities with its dedicated search mechanism, which is an important feature in the changing fitness landscape that is created by adding more rules to the pool and therefore increasing the search space with each new SC phase. Similarly, the GA's exploration capability seems to allow it quite well to replace selected rules within its solution with new ones. However, we should keep in mind that the differences to the other metaheuristics are still rather small.[30]

Overall, we can say that ABC and GA are the most promising candidates for further investigation and practical use. As GAs are far more widespread and recognized, I would recommend their usage over the less well-known ABC if a choice had to be made. This also comes with the advantage that a lot more improvements for GAs have been studied and proposed in literature than for ABC. One possible set of such improvements will be investigated in the next section, where we will highlight and benchmark possible approaches to reduce the number of hyper-

[30]We also performed the t-tests for all other combinations of algorithms. The resulting plots can be found at https://github.com/heidmic/diss-graphs/tree/main/SC.

parameters to set for the GA in SupRB by introducing self-adaptive components to it that were inspired by literature on self-adaptive GAs.

6.6 Solution Composition—Self-adaptive Approaches

In the previous section (Section 6.5), we investigated whether we should use another optimizer rather than the Genetic Algorithm (GA) for solution composition (SC) in SupRB. This section deals with possible improvements of the GA itself. While we would of course want to improve the raw performance in terms of model size and prediction errors, another critical aspect of getting SupRB applied by practitioners in real-world use cases is the question of how hard it is to use. Self-adaptive Genetic Algorithms (SAGAs) could be a useful tool to decrease the number of hyperparameters that need to be tuned or at least to make the tunable hyperparameters less impactful. While this might come with some loss of training speed, it is probably a worthwhile trade-off to simultaneously make training easier for the data scientists and ML engineers. If successful, this could help with a wider adoption of SupRB for real-world tasks but also flatten the learning curve interested researchers might encounter if they were to study SupRB further.

This section presents a set of experiments comparing four different SAGA approaches with the GA baseline. It is based on the Bachelor's thesis of Maximilian Krischan which was supervised by me and was expanded with a much more substantial statistical analysis into a publication at the 2024 IEEE Congress of Evolutionary Computation [Hei+24].

The use of self-adaptive operators within the metaheuristic optimizer of LCSs has been an object of research for decades: Hurst and Bull [HB01] did build a fully self-adaptive ZCS [Wil94] (the LCS on which the more well-known XCS [Wil95] is based). They found that the self-adaptive parameters resulted in better performance over static ones even in stationary environments but especially in dynamic settings. Hurst and Bull [HB02] also tested a self-adaptive mutation and learning rate in XCS and showed that it improves XCS's poor performance on long action chain environments. Furthermore, Bull and Hurst [BH03] used self-adaptive parameters for an LCS using small neural networks as the individual rules and showed that this is beneficial on different variants of maze running. Unold [Uno10] investigated whether self-adaptive mutation benefits the rule discovery process of XCS and showed that it does for multiplexer tasks.

6.6.1 Self-adaptive GAs

In this subsection, we discuss the four SAGAs we adapted to be fit for the solution composition task in SupRB. To make this section easier to read, we decided to number them, even if in their original paper, they were given a name. This also reflects the fact that we had to do some changes to make them suitable. We selected these algorithms specifically as they are relatively easy to understand, and therefore to explain to inquiring stakeholders, and did not construct essentially new metaheuristics but focussed on smaller adjustments to the underlying GA.

SAGA1 This GA [Vas+01] dynamically adapts crossover and mutation rates to keep the genetic diversity in the population high. Diversity is determined by the coefficient of average solution fitness to the maximal fitness within this population. Crossover and mutation rates are adapted inversely. The more diverse a population is, the higher crossover and the lower mutation will be.

SAGA2 Sun and Lu [SL19] adjust crossover and mutation rates according to the diversity of the current population and the similarity between the current and the preceding generation. First, SAGA2 adapts the bounds available for both rates on a global level and then individual rates are set dynamically between these bounds. Diversity is measured based on minimal, maximal, and mean fitness. Similarity operates directly on the sets of fitness values. Based on a combination of diversity and similarity, bounds for mutation and crossover rates are increased or decreased, similar to SAGA1. Individual crossover rates are calculated for each operation based on the overall bounds and the relative fitness of two selected parents (using the mean of both) in comparison to the mean, min and max fitnesses within the current population. A similar process is used for mutation.

SAGA3 Kivijärvi et al. [KFN03] assign each individual new parameters for their respective mutation rates, crossover operators, and a noise factor. We make a small adjustment for SAGA3, in that we do not need the noise factor and instead introduce a crossover rate behaving similarly to the mutation rate. Parameters are generally propagated via crossover and mutated along with the other parameters of an individual. After selecting the parents, their bookkeeping parameters are recombined by randomly selecting one of the parents' parameters. Then, the new child's parameters are mutated according to a fixed probability of 0.05 as suggested by Kivijärvi et al. [KFN03]. Afterwards, crossover (of both parents) and mutation are applied to the chromosome as per the current parameters of the new child.

SAGA4 In this approach, inspired by [ST11], we adapt only the population size rather than other parameters. They also adapted mutation and crossover rates, but their experiments were not as promising as the population adaptivity. Instead of performing a generational replacement in the GA, each individual receives an "age" attribute, which gets reduced by one in each generation until the individual is eliminated by dropping to zero. The age is updated based on fitness, i.e. individuals with a fitness better than the median receive an extra generation to live, and overall size, i.e. if the population becomes very large (10 times the original size) all individuals below a dynamic fitness threshold loose two generations. All individuals start with an age of three. This probably reminds researchers with an XCS(F) background about the steady-state GA within XCS(F) that adds individuals to the population up to a maximum size and deletes them according to some deletion-scheme. In the case of SAGA4, we however do not have a hard upper limit but rather a relatively soft one and delete at every iteration, whereas in XCS(F) the deletion is only invoked when needed. Regardless, the similarity might resonate and be used as a bridge to explain SAGA4 to LCSs experts.

6.6.2 Evaluation

In this section, we first present the experimental setup including the tested datasets, then display the results and, finally, perform an extensive statistical analysis.

Experiment Setup

The code of our implementation of the self-adaptive GAs as well as our experiments have been merged into the respective official SupRB repositories.[31] In general, our setup follows the general outline as described in Section 5.2.1 and the previous sections of this chapter.[32]

The (non-adaptive) hyperparameters of our (SA)GAs[33] and SupRB's remaining configuration options were tuned for each learning task using a Tree-structured Parzen Estimator in the Optuna framework [Aki+19] that optimizes average *solution error* on 4-fold cross-validation with a fixed budget up to 360 core hours or 1000 trials. We opted to tune for error rather than fitness here, following the insights of the previous section where we found that the GA is able to provide sufficiently compact solutions (performing slightly better when compared to the other optimizers; cf. Section 6.5.3) and wanted to investigate whether focussing on errors during tuning could be more sensible for the GA. Our final evaluation uses eight different splits of training and test data, where the test data was always one fourth of the data set. We evaluate each GA with eight different random seeds for each train-test-split, resulting in a total of 64 runs.

We perform 32 iterations of SupRB, where each rule discovery phase produces four rules with an $(1, 20)$-ES. This ES is allowed to run until it did not find a new better rule for 146 generations (which is the default of SupRB). We limited the tuning of the ES slightly to be able to spend a larger share of the tuning budget on the (SA)GAs. Results can therefore be slightly worse than in previous experiments due to a less effective rule discovery. We automatically tuned the σ of the ES's mutation and the fitness weight between the objectives of large and accurate individual rules. The remaining options for the ES are statically configured as described above. Each SAGA has slightly different parameters which need to be configured but, in general, we tune the number of generations, the selection operator and (for all but SAGA3) the crossover

[31]The current version of SupRB is always found at https://github.com/heidmic/suprb while the version used for all experiments in this dissertation is long-term archived at https://doi.org/10.5281/zenodo.14181292.

[32]The code to reproduce the experiments can be found in SupRB's experiment repository: https://github.com/heidmic/suprb-experimentation/

[33]We use (SA)GAs to refer to the group of SAGAs and the GA.

operator to use. Crucially, we—in contrast to the previously presented experiments—also allowed the tuner to increase the number of generations in the expectation that self-adaptivity can sometimes take longer to arrive at a point of balance.

Table 6.5: Overview of the five datasets we benchmark our proposed SAGAs on.

Name (Abbreviation)	n_{dim}	n_{sample}
Combined Cycle Power Plant (CCPP) [KT12; Tüf14]	4	9568
Airfoil Self-Noise (ASN) [BPM89]	5	1503
Concrete Strength (CS) [Yeh98]	8	1030
Physicochemical Properties of Protein Tertiary Structure (PPPTS)	9	45739
Parkinsons Telemonitoring (PT) [Tsa+09]	18	5875

We test on five datasets part of the UCI Machine Learning Repository [DG17]. An overview of dimensionalities and sample sizes is given in Table 6.5. Section 6.4.3 provides a more extensive description of these datasets. I decided to omit the previously-used EEC for these experiments, as we found that the results did not add anything substantial as a benchmark that was not already contained by the also relatively linear CCPP dataset, thus keeping the presentation more streamlined.[34]

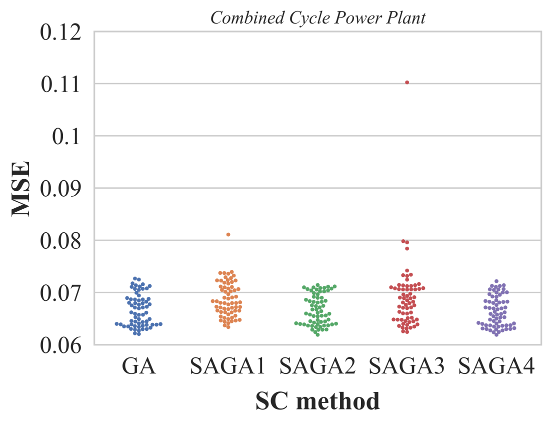
Results

As in the previous sections, in the following, we abbreviate *SupRB using X as its SC method* simply by *X*, e.g. *SupRB using SAGA1 as its SC method* is just signified as *SAGA1*. The other optimizers follow accordingly.

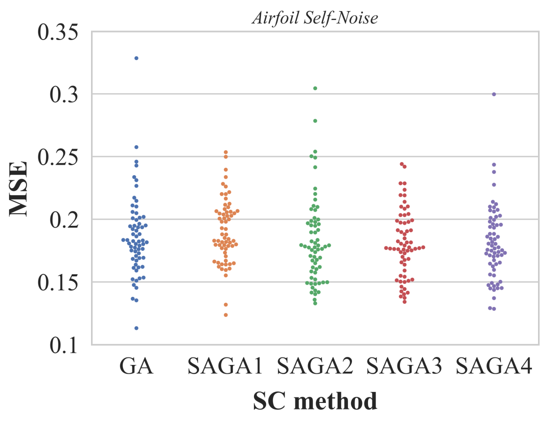
From the experimental results, we find that the original GA performs relatively similarly to the four SAGAs. On one hand, this is of course a positive result, as it tells us that self-adaptation is not needed (or at least not highly beneficial) to navigate the optimization landscape relatively well and the GA alone does not get stuck in an early (and bad) local optimum it found.[35] On the other hand, the benefits of using self-adaptivity become less clear at the first glance. But of course, we still trade in the need to configure a lot of hyperparameters for no (or at least very little) performance loss by making some of the previously static parameters self-adaptive, which can be a valuable benefit. However, training got slower with the SAGAs (a summary of a detailed runtime analysis is found at the end of this subsection).

When analysing the distributions of *mean squared errors* the 64 runs produced per dataset, we find that, albeit marginally, the GA, SAGA2, and SAGA3 outperform SAGA1 and SAGA4. Figure 6.21 displays these distributions in detail, where each dot represents one run's elitist's test scores. All 64 runs per algorithm and dataset were performed with the parameters determined by their individual tuning processes. Optically, it is not easy to make definitive distinctions on

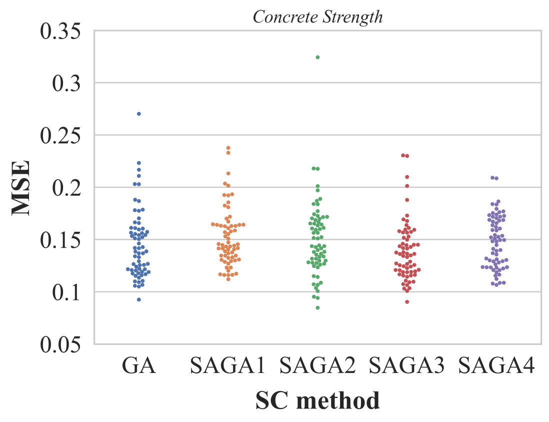
[34]This was also done for [Hei+24].
[35]Of course, this primarily corroborates the results of Section 6.5.



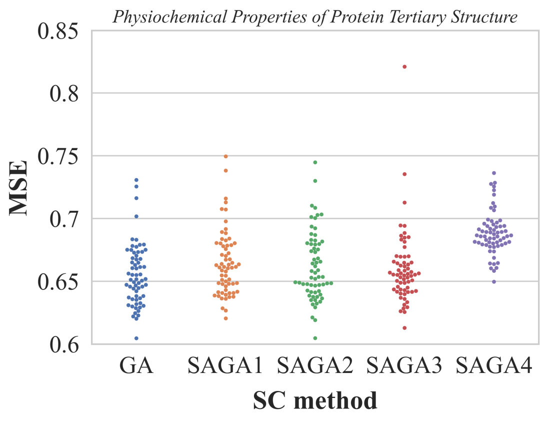
(a) Distribution of errors on CCPP



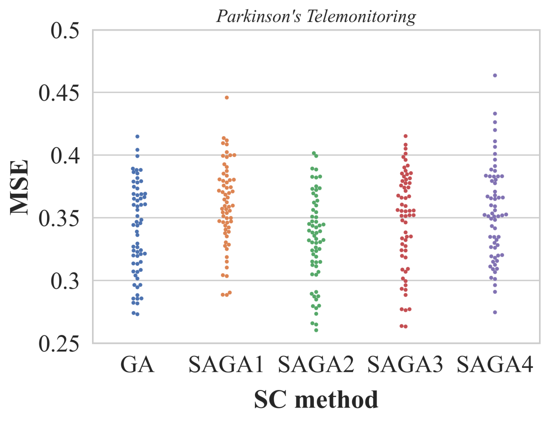
(b) Distribution of errors on ASN



(c) Distribution of errors on CS



(d) Distribution of errors on PPPTS



(e) Distribution of errors on PT

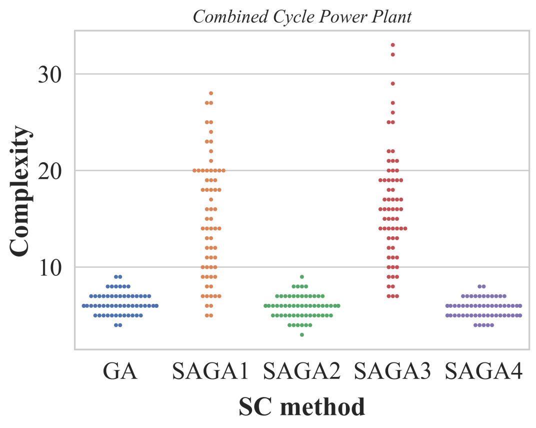
Figure 6.21: Distribution of (SA)GA runs' *mean squared errors.* Note the different scales, reflecting varying difficulty of the learning tasks. Also note that the EEC dataset, which was used in the previous sections, has been removed as explained in Section 6.6.2.

performance due to the shapes of the distributions. Other common forms of displaying results such as violin or box plots obscure this fact more but we find that this is an important results in itself. The performance of the algorithms is not only based on the data split but also the random seed and—even with self-adaptivity—the evolutionary search is unable to converge on the same value each time. From exemplary analyses of the pools of rules after training, we assume that this has more to do with the fact that some beneficial rules are missing (or others are ill-placed). However, the current solution composition approaches are not free from responsibility for the performance measured as they did not always converge optimally, especially during the earlier training cycles, and, crucially, RD uses information about the current elitist's performance to guide its search. An interesting observation is the number of sometimes even relatively strong outliers in performance, where individual runs underperformed vastly compared to their peers, but also the generally rather wide spreads.

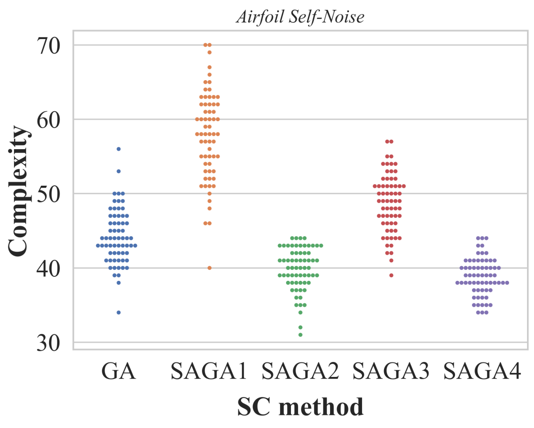
The (SA)GAs had a second objective that was to optimize model *complexity*. As discussed before in various chapters of this thesis, e.g., Section 3.1 and Section 4.3.2, when it comes to explainability, this is a crucial parameter that determines the usability of a specific model. Arguably, error is more relevant for many applications, but—on the other hand—in real-world processes we are often also confronted with situations where tolerances for prediction errors exist and it is reasonable to assume stakeholders would prefer models they can actually analyse if all models in question perform within tolerance. In Figure 6.22, we show complexities per algorithm and dataset in a similar manner to the errors in Figure 6.21. Recall that, in contrast to prediction errors, all complexities have to be integers, which makes these plots more neat looking but has no other direct implications. We can clearly see that SAGA4 is the best overall algorithm for complexity on this selection of datasets. Second best seems to be SAGA2, followed by the GA. SAGA1 and SAGA3 are clearly worse. Especially on PPPTS, SAGA4 seemingly shifted strongly toward substantially more compact solutions instead of more accurate ones. On the other sets, SAGA4 showed vaguely similar errors while still maintaining at least a small lead in terms of model complexity.

We did also measure runtimes for individual training and evaluation runs in isolated environments.[36] We found that, as expected, the GA is the fastest of the five algorithms under investigation, albeit marginally (reducing the runtime by less than 10% when compared to the others in most cases). However, it has more parameters to tune, so as soon as tuning comes into play, the algorithms become much more equal. While specifics were highly dataset dependent, SAGA2 is roughly equal to the others on all but PPPTS, where it took almost 55% longer than the GA. However, on CCPP, where GA and SAGA2 were almost identical in runtime, SAGA3 ran 30% longer than both. On the other datasets, the algorithms were rather close all the time with very low standard deviations (typically between 0.5% and 1.5% of the total runtime). Low variance leads us to assume that runtime is not really dependent on the data split, the random seed of the optimizer, or even on the performance of the algorithm, but purely a question of which values hyperparameters were tuned to, which in turn correlates with task difficulty.

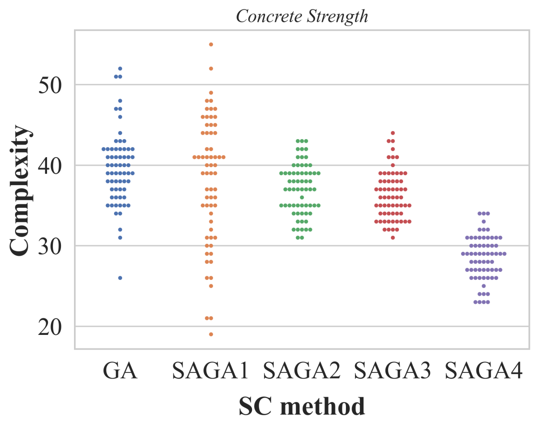
[36]The runtime analysis was done on a Ryzen 5 5600X desktop computer with 32GB RAM and Windows 11 that was not running any other tasks during the experiments.



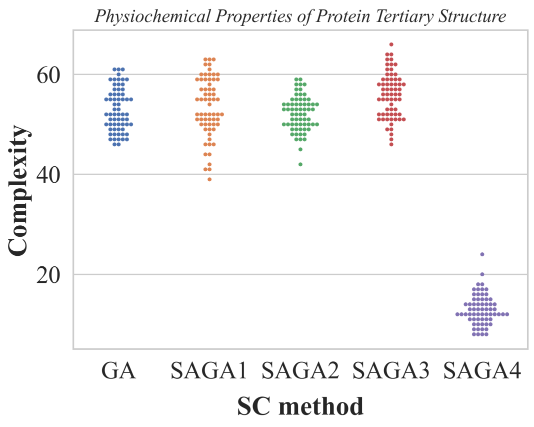
(a) Distribution of complexities on CCPP



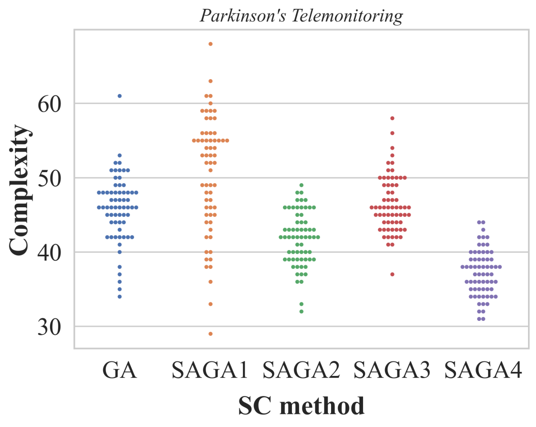
(b) Distribution of complexities on ASN



(c) Distribution of complexities on CS



(d) Distribution of complexities on PPPTS



(e) Distribution of complexities on PT

Figure 6.22: Distribution of (SA)GA runs' *model complexities*. Note the different scales, reflecting varying difficulty of the learning tasks. Also note that the EEC dataset, which was used in the previous sections, has been removed as explained in Section 6.6.2.

Statistical Analysis

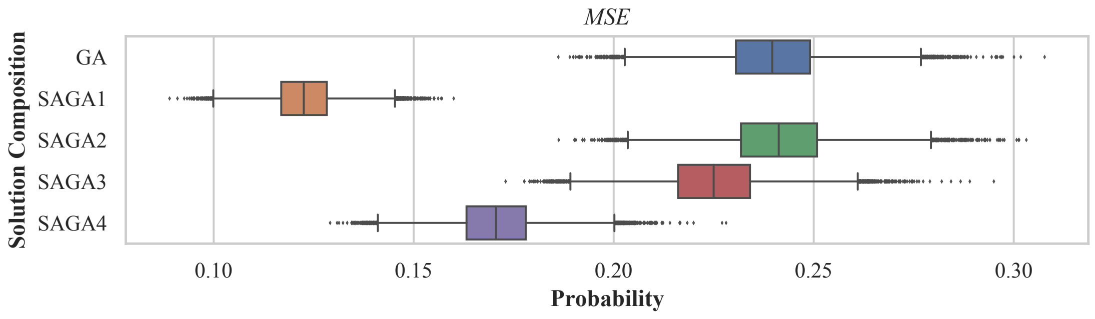


Figure 6.23: Box plot (with standard 1.5 IQR whiskers and outliers) of the posterior distribution obtained from the model by Calvo et al. [CCL18; Cal+19] applied to the MSE data. A SAGA method having a probability value of q% says that the probability of that SAGA method performing best with respect to MSE is q%.

As the results of our algorithms are not allowing a conclusive decision based on visualization alone, we perform a rigorous statistical analysis to aid us in the decision which algorithm we should select or propose to others to try first when using SupRB on their data. This analysis is similar to the previous three sections in this chapter (cf. Sections 6.3 to 6.5). We begin with the application of the model proposed by Calvo et al. [CCL18; Cal+19][37] to the gathered data. This model gives us the *posterior distribution over the probability of that solution composition method performing best*. Figure 6.23 displays the results of this analysis for prediction errors on the test set. Note that typical minimal thresholds for automated decision making are 80% (or usually even more) probability assigned to a single algorithm [Ben+17]. Based on that we cannot clearly make a decision regarding which algorithm to choose. However, if we were to pick a top 3, it would be GA, SAGA2, and SAGA3, reaffirming our results from the visual inspection of the MSE data in Figure 6.21. Nevertheless, SAGA1 and SAGA4 do not fall too far behind the others.

When performing the same analysis for complexity (cf. Figure 6.24), we find a much clearer picture than with the MSEs, confirming the results of our visual inspection. When choosing one of these algorithms on any dataset similar to those tested, we can expect SAGA4 to produce the smallest model for about 62% of runs. SAGA2 ranks on second place, being the best in about 17% of runs, and third is the GA with marginally more than 10% of cases.

Overall, based on the Calvo models, we conclude: when small models are the goal, we should probably run SupRB with SAGA4 in most cases, however, recall that we are still below the threshold typically used for automated decisions and SAGA4 seems to underperform a bit in terms of predictive performance (especially on PPPTS, cf. Figure 6.21d). If we wanted to make a general recommendation based on this experiment's results, SAGA2 seems like a plausible

[37]We use the implementation provided by https://github.com/dpaetzel/cmpbayes using 10,000 steps in the MCMC process.

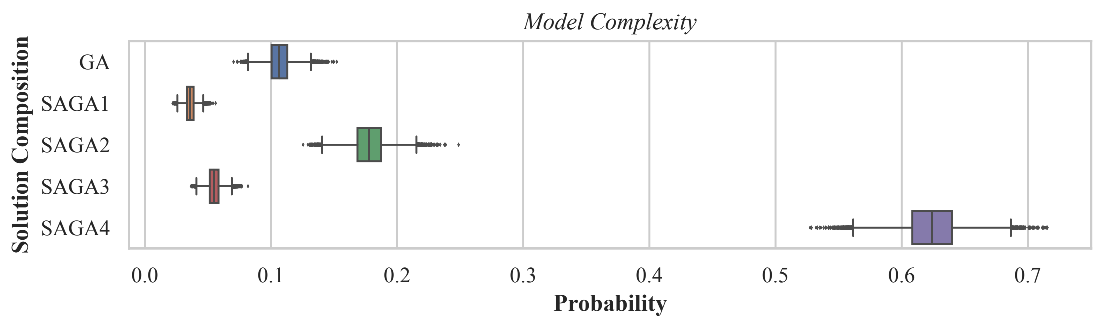


Figure 6.24: Box plot like Figure 6.23 but for the model complexity metric.

top candidate. Even the original GA is not far off and, depending on tuning budgets, might be a satisfactory middle-ground approach.

To better estimate the effect sizes of a decision on one algorithm, we now apply Corani and Benavoli's Bayesian correlated t-test [CB15][38] between the algorithm combinations in question. This test compares two solution composition approaches directly and takes into account the probability distribution of the difference in performance, i.e. it tells us how much better we can expect an algorithm to perform on a certain dataset. Note that while we did perform this test for all relevant combinations, we will only show comparisons of GA and SAGA2/SAGA4.[39] Figure 6.25 shows the results for this test between the GA and SAGA2 on errors as these were the best SC optimizers in terms of errors. We find that the performance is really strikingly similar. Even if CCPP and PT return slightly smaller or wider distributions (recall that the targets of the datasets were standardized, so we show errors scaled with the dataset's standard deviation rather than actual raw values), we can say that the modes are very close to equal performance for all datasets, and on ASN, CS, and PPPTS we even have very similar variance. While this is not surprising given the near equal probability for best results in the Calvo model, it is by no means guaranteed. That result of the Calvo model could also be achieved with t-tests that are showing substantially different effect sizes across datasets, e.g. GA superior on CCPP, SAGA2 on ASN, GA on CS, and so on, rather than overall equal performance. This difference is most likely *not practically significant* and it is therefore not even relevant whether it would be statistically significant according to the typically-utilized, but still arbitrary, thresholds (often called “significance levels”).

Figure 6.26 shows the results for this test between the GA and SAGA4 on errors, therefore comparing the baseline with the most likely best SC optimizer for model complexity according to our tests in Figure 6.24. CCPP, ASN and PT are about the same as with GA and SAGA2 in Figure 6.25. The mode for CS is also near 0 but the distribution is a bit tighter. The most noticeable difference is on PPPTS which is in the GA's favour. About 95% of runs of the GA will be better on this than the runs of SAGA4. While this is a statistically significant difference, the practical significance is a bit unclear as the effect size is quite small. In Section 6.1.1 we

[38]We use the implementation provided by https://github.com/dpaetzel/cmpbayes.
[39]The remaining t-tests can be found at https://github.com/heidmic/diss-graphs/tree/main/SAGA.

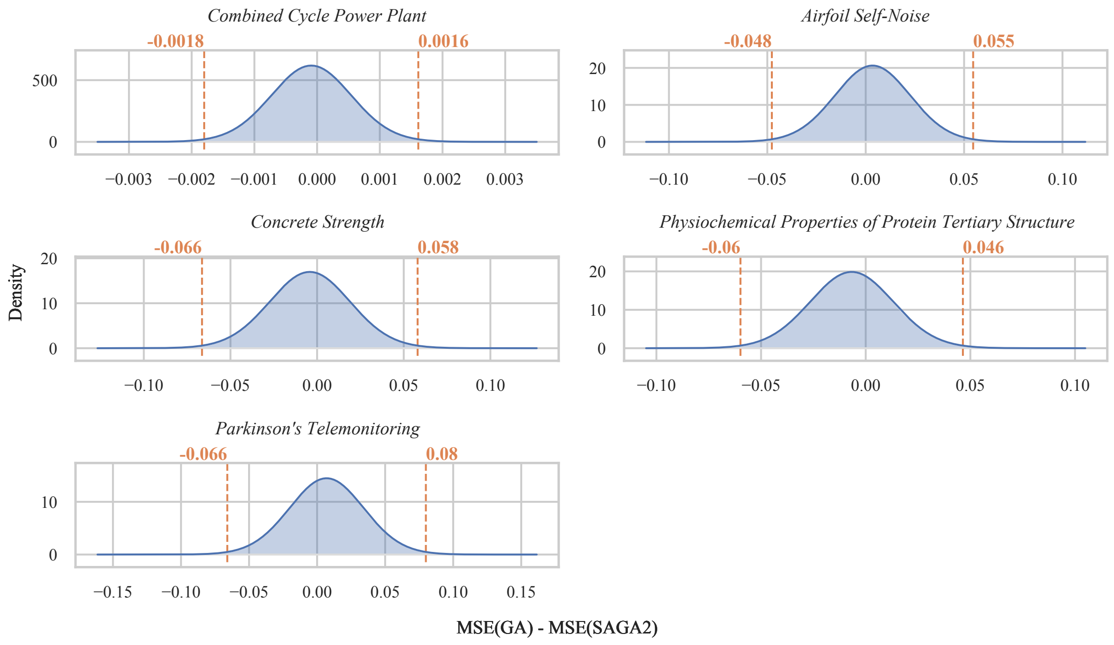


Figure 6.25: Density plot of the posterior distribution obtained from Corani and Benavoli's Bayesian correlated t-test [CB15] applied to the difference in MSE between GA and SAGA2. Orange dashed lines and numbers indicate the 99% HPDI (i.e. 99% of probability mass lies within these bounds). HPDI bounds rounded to two significant figures. Effectively, this indicates how likely a specific difference between one run of each SC method will be (practically, probability mass above zero equates to GA having a higher (worse) value than SAGA2).

used 0.005 as the region of practical equivalence (ROPE)[40] which the majority of probability mass would fall into. Only about half a percentage point would give the better performance to SAGA4 if we use this ROPE here as well. Note that with this ROPE all of the probability mass on CCPP falls within this window.

Finally, we show an example t-test for the difference in model complexity in Figure 6.27. We chose SAGA4 as the most compact algorithm and the GA, which was also the combination we saw a valuable difference on one dataset before (Figure 6.26). We add a ROPE identical to what we did for the t-tests on model complexity in Section 6.4.3. It is based on the mean of the standard deviations of all algorithms' model complexities on this dataset (see the relevant segment of Section 6.4.3 for our reasoning of choosing this value). We find that (as with errors) the performance on CCPP is practically equal according to our ROPE. However, we also find that, for CS and PPPTS, SAGA4 is composing significantly smaller models in 99% of cases. Considering the effect size on CS, we find that we should expect over 10 rules less, which equals about 25% of the GA's model sizes. On PPPTS, the size of the effect is even more considerable

[40]The ROPE is the region of differences in which we consider the algorithms practically equal in performance.

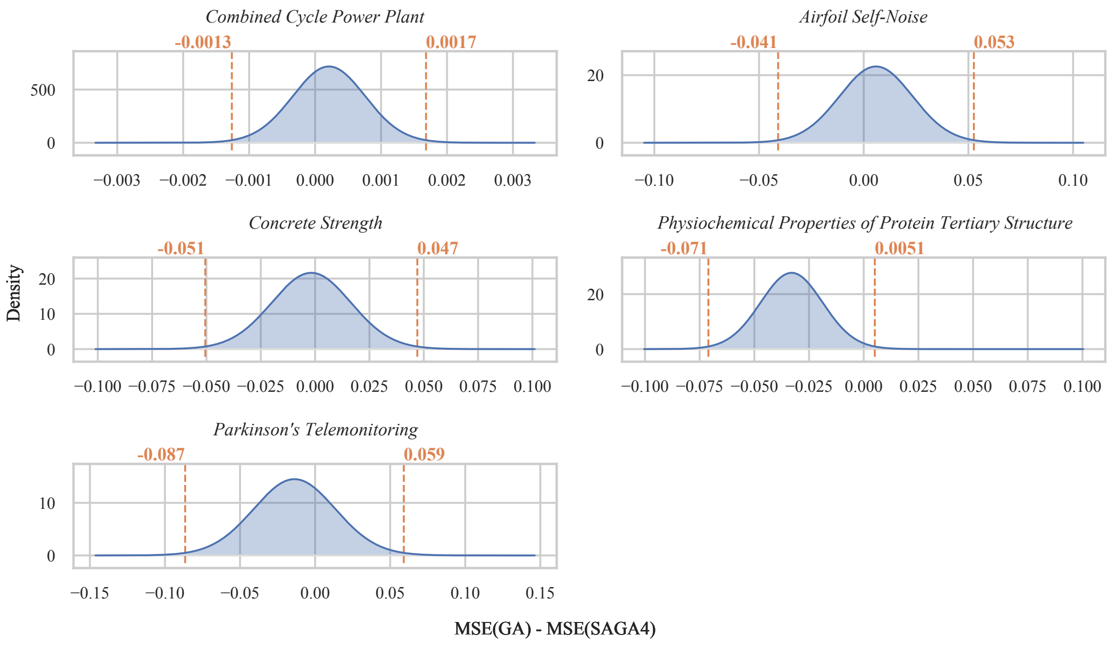


Figure 6.26: Density plot of the posterior distributions like Figure 6.25 but for the difference in MSE between GA and SAGA4.

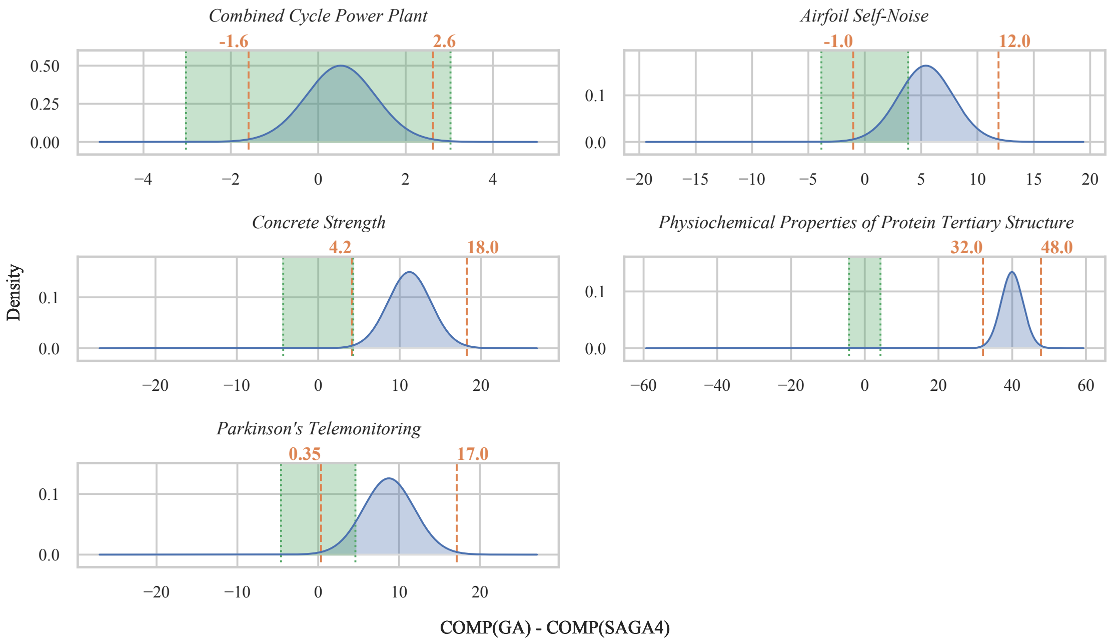


Figure 6.27: Density plot of the posterior distributions like Figure 6.25 but for the difference in complexity between GA and SAGA4. In green, we show a region of practical equivalence (ROPE). Probability mass right of the ROPE equates that we expect that the GA has a practically (and statistically) significantly higher model size.

(the GA had an average of 53 rules whereas SAGA4's average was 13). Whether or not this negates the drop in accuracy we noticed earlier is of course dependent on the actual application, but it is not hard to see why the optimizers would consider their returned solutions high-fitness individuals despite the clear difference. If we assume that there exists a Pareto front of both objectives, the solutions would lie on quite different ends of this front. Overall, I find it quite interesting that the change from a generational replacement in the GA to a (sort of) steady-state GA with deletions has this much of an effect and especially one in this direction of the fitness landscape, as the other SAGAs show smaller changes and the fitness function is the same for all optimizers. This is especially interesting as XCS(F) and its derivatives use a steady-state GA with a deletion mechanism. Therefore, at least for other LCS researchers, this option is probably an intuitive choice.

As a general recommendation, I would choose either the GA or SAGA4, depending on if a small increase of prediction performance is more important or if model complexity is more relevant. The GA's process is still easier to explain and understand than SAGA2, which has a lot of interplaying parts related to the fitness and the overall makeup of the population. While it does require less tuning, as was the goal of this study, I assume that in most cases we have the budget available. When tuning for an equal number of trials rather than a fixed time, one should note that the runtime of SAGA2 can be practically (and statistically) significantly higher though. If there is no tuning budget at all, SAGA2 is the better choice as crossover and mutation rates can be quite impactful on performance and are automatically adapted here.

7 Outlook

While we answered some of the more immediate questions surrounding SupRB within the studies that form the basis of this thesis, there are still a lot of possible avenues left for future research. I see both paths towards applied research, which can ultimately lead to pure application, and paths that focus more on improvements of the system itself.

In this thesis, I found that SupRB's initial design was made on good assumptions that cannot easily be outperformed by the numerous options proposed throughout Chapter 6. In my view, this is an important step to clear before testing the application with actual real-world users as, usually, the (time) budgets for evaluations regarding the practical explainability, and thus applicability, of models are rather limited within industry. Therefore, we should only discuss those (types of) models that we are confident about being good candidates with our stakeholders. SupRB using ES and GA for RD and SC, respectively, and using ordered bound hyperrectangular conditions with linear local models within rules (cf. Section 4.1) seems to be a good candidate.[1]

Thus, the immediate next step will be to investigate the practical explainability with the stakeholders from Section 3.5. This would also help confirm whether or not the stakeholders expressed their explainability-related requirements on model design correctly and effectively and could serve as an important step to improving the process further, possibly adjusting the questionnaire (cf. Section 3.4). Preliminary testing using the available data from that scenario shows that SupRB is able to learn meaningful relationships from that data with acceptable model sizes. After finding good hyperparameters and training some candidate models, we will design a study to determine whether or not these models are explainable for individual stakeholders. Ideally, we will be able to quantify (or at least rank) models and will hopefully find trends based on certain stakeholder traits, e.g. the groups defined in Section 3.5. Besides the baseline SupRB model, we could include some of the more promising options from the previous chapter but should at least include decision trees and an appropriate black-box model. From preliminary talks we had with some of the stakeholders on how this explainability study should best be designed, we know that stakeholders—other than the data scientists—might

- be unaware of AI/ML in general,
- lack the necessary knowledge about the types of models and how they operate,
- not understand (or agree on) how we measure performance,

[1]This does not mean to exclude some of the other options that were performing quite well in Chapter 6, and some of those (and especially combinations of these) could turn out to be better candidates than the baseline SupRB depending on the use case.

- fail to mentally visualize the models and derive what the implications of a specific model structure and parameters are,
- have difficulties distinguishing specific models from each other, e.g. because individual models might not always agree with each other, for example, as some rules could be present in one model but not the other, and
- get overwhelmed with the large number of new concepts and information.

This raises a lot of questions for designing the study, which have to be answered in the future. Hopefully, we will be able to build a template similarly to [HNH21] that we and others can then also evaluate on use cases from other companies/industries. Of course, also the study from [Hei+23b] should be repeated on other domains to reaffirm the correctness and effectiveness of the template of [HNH21].

For any study regarding the practical explainability of (RB)ML models for the stakeholders from our case study in Section 3.5, we would first need to find a way to effectively present the models. While visualizations and formats of presentation exist for LCSs and DTs (and others), these can quickly become quite complex, especially when large numbers of rules are involved or the datasets have many dimensions.[2] This could—among many others—raise questions such as (cf. also Section 4.3.2):

1. Should we present rules individually or grouped? If so, how do we group them?
2. How do we prepare the data? Should we use scaled features, keep original units, or even allow some feature engineering (cf. Table 5.4)?
3. What is a good UI/UX design for such a study? Should we try to keep it similar to what they are used to from their day-to-day interface? Do we even need to integrate the study into that interface and possibly measure their satisfaction in-line rather than in a more artificial environment?
4. Do we need to make these and other decisions per group of stakeholders or can we design a more general study? Do they even have the same questions as the other groups?

For question 4, we already have some answers: As we learned from our interviews in Section 3.5, operators will often have less interest in the model as a whole and will most likely be more interested in case-by-case decisions. Possibly, we could satisfy them with techniques like explanation-by-example and an integration into their interface. In one interview, an interviewee raised the suggestion that a short natural language explanation might suffice and be preferred to any complex plotting or other visualization technique. We performed some very preliminary testing using various LLMs but found their answers not specific enough and too repetitive. However, advances in this field (which right now are very fast) and extensive

[2]There is no clear cut bound for when the dimensionality is high enough to cause issues with understanding but probably most cases with more than ten dimensions without multicollinearity will be hard to follow. If we were to focus on plotting the rules, even two inputs can become challenging.

prompt-engineering does keep that a promising option in my view.[3] By contrast, process engineers will most likely want to gain knowledge from the data via the model and therefore want to analyse the rules and their relations and interactions with each other. While we could support their inquiries by providing "what-if" scenarios, we would certainly need more detail and a wider focus.

With such a study, we can hopefully answer some of the questions regarding the explainability of SupRB. Ideally, we can also test some of the assumptions we made regarding what options/configurations/design choices might be "less explainable", e.g. whether the restriction of mixing models to at most three rules is deemed useful or important as it does come with some loss in performance. Based on our findings, we will likely be able to improve SupRB further, both in regards of providing better performing models, but also by making the model's explainability better adapted to one specific use case.[4] One assumption we made based on our study and a look into the more general literature in this field is that the training process does not have to be explainable. Hopefully, we will also find answers to this question and maybe find avenues to test the training process' explainability and improve it in the future.

Tied into the question of determining the practical explainability is also the question of balancing model size versus performance. The experiments in Chapter 6 did regularly show that some configurations of SupRB drove it towards smaller models while others opted for smaller errors instead. We have some informed assumptions about preferred model sizes but should check whether these hold true when it comes to discussing actual models. Additionally, I suggest that the effects of choosing the α[5] and β[6] parameters of our fitness functions should be investigated more in-depth. However, I do not think that the stakeholders should be involved directly in this. Instead, I suggest that a larger set of experiments is conducted beforehand and then the stakeholders are confronted with some representatives of large and small models and can judge them according to individual preference.

If the assumption that stakeholders are rather ambivalent about matching is confirmed, future research could also explore the use of more complex matching functions (cf. Section 6.1.2). An interesting example could be code-fragments as proposed by Iqbal, Browne, and Zhang [IBZ14a], which might allow the transfer of knowledge to similar tasks or different machines performing the same general task but with a different configuration, which was for example used by Siddique, Browne, and Grimshaw [SBG20] to reuse knowledge at different abstraction levels. Another option could be the use of heterogeneous matching functions where for each feature the simplest option that still allows appropriate partitions is utilized. This could also

[3]From correspondence with some other international LCS researchers, I know that at least one group is currently working on similar options and we specifically included the use of LLMs for LCS rule generation and explanation into the call for the IWERL workshop at GECCO in 2025, which I will be co-organizing.

[4]There is of course the hope that similar domains or use cases will have similar requirements, but in general much of the literature surrounding explainable AI [e.g. DK18] seems to agree that there is no general-purpose approach.

[5]The two respective objectives are weighed against each other by α that is used for rule and solution fitness; cf. Equation (4.2).

[6]The sensitivity of the Pseudo-Accuracy objective to the in-sample MSE within each rule is determined by β; cf. Equation (4.3).

allow binary or mixed-integer inputs which would be a common way to describe machine configurations.

While Ridge regression seems to be a good local model (cf. Section 6.2), future work could test the use of ElasticNet and find good guidelines for the parameters of the regularization. Also, we did not yet test more complex local models in a thorough study. While the assumption that non-linear models are not sufficiently explainable seems very plausible (and was supported by stakeholders of Section 3.5), the described-above explainability study could challenge this assumption. Especially if we were to implement a system that is based on short natural language explanations and explanation-by-example, more complex models might still fit the requirements. However, personally, I doubt that SupRB should be the machine learning system of choice in this case. I rather think that neural networks will be appropriate in this case but this should be investigated with actual users in a real-world setting.

While we already investigated some options for mixing in Section 6.3, a lot of further options exist, such as non-constant weights that are dependent on the input region, e.g. lower the weight where the local model did have high training errors[7], or are fitted during solution composition (similarly to MoE models; cf. Section 2.2.2). Drugowitsch [Dru07] also discussed some other options we did not yet consider in SupRB.

The fitnesses of SupRB's rules and solutions have two objectives each. During training, we used a type of scalarization to weigh these objectives against each other. While this is the most sensible approach during most of the training (cf. Section 4.1 for our reasoning), it could be interesting to approximate the Pareto front instead in the final SC phase. This could provide an option for the actual users of the system to choose if they want to deploy a model with lower errors or one with lower complexity without any assumption about balancing. However, one should keep in mind that the assumption would still be made for most of the training and guide that training heavily. Additionally, after all rules have been added to the pool, I think it might be valuable to drop all biases about what rules might compose a good model (which we made by always adding the previous elitist to the population). Previously, that last SC phase is also only run as long as the previous ones without ensuring convergence. While in general, the results of SupRB are convincing, optimizing until convergence could boost the results slightly.

Combining the self-adapting approaches of SAGA2 and SAGA4 (cf. Section 6.6.1) should also be tested, as well as self-adaptive algorithms that use non-fitness-based diversity measures. This could further boost the training performance and make hyperparameter setting/tuning easier. I also encourage other researchers to develop more options for making SupRB's parameters self-adaptive or less impactful. In general, we have some understanding on what parameters have to be tuned per dataset and how sensitive SupRB's performance is to them, but a thorough hyperparameter study should be done in the future. Crucially, such a study should be compiled and published as a comprehensive overview to be available to a wide array of researchers and practitioners. Better knowledge about tuning ranges and tunable parameters could also speed up the overall training process as we could shift the focus towards the

[7]Note that rules should match larger quantities of examples for this to have noticeable effects.

most relevant parameters or could even set some in advance based on general guidelines. During the experiments of Chapter 6, we found that some tuning of the ES is critical for effective application. The experiments on SC showed that if we were to only just choose defaults for RD, the results are significantly worse (expect errors to be doubled) and that only tuning a subset of what we regularly tune already has slight effects (cf. Section 6.6.2).[8] We also found in separate studies that tuning every parameter available in the GA can have some undesired effects. Most of the time this made only a small difference but on some datasets it lead to very different model sizes and—in turn—higher errors and could even lead to collapsing populations which did not happen with more limited tuning and more assumptions. This effect could most easily be reproduced when the solution fitness calculating function was switched to other options we implemented before setting the fitness function described in Section 4.1 as the default. In my view, all of this strongly warrants increased hyperparameter studies[9] and some improvements such as self-adaptivity to make us less dependent on choosing (or searching for) good values.

Besides outfitting the RD method(s) with some forms of self-adaptivity[10] to make the RD process more robust, we could also experiment with different values for α in the fitness between different RD phases but in the same training process, or the termination criterion of the search. We also might want to make the number of cycles of SupRB dynamic rather than statically configured, thus, effectively choosing a different termination criterion for SupRB. An interesting, and previously unexplored, avenue to make SupRB's overall training process more efficient is to increase the information the optimizers utilize from the previous phase, i.e. RD uses more information about the elitist solution(s) returned by the last SC phase or includes the crowding of rules to choose the new starting points, and SC starts to utilize information of RD, e.g. tests the integration of the new rules first rather than randomly mutating into these (or not).

While our experiments on RD (cf. Section 6.4) showed that ES is competitive with the NS approaches, the experiments could be repeated by discovering more rules, based on other assumptions, and using different functions for the novelty score. The "behaviour" of rules being tied to the set of matched examples is reasonable but the Hamming distance might not be ideal. MAP-Elites [MC15] and other quality-diversity techniques might also be worth investigating as they have been successfully used for DTs [FCI23].

At the end of Section 4.1, I already hinted that SupRB's extension to also being available as a classifier rather than only as a regressor could be relatively easy to implement. At the time of writing this thesis, a student of mine is investigating this option. However, I still want to encourage other researchers to make that step. A dedicated researcher will have a much richer and diverse experience with ML and therefore be able to gain more insights while hopefully coming up with different and possibly more effective ways of handling this new domain. To allow the usage of SupRB for creating classifier models, the first step would be the use of local

[8]In the SAGA experiments, we tuned the α of rule fitness and the mutation's σ parameter but not the mutation's early stopping factor we tuned in the other experiments.

[9]Although this would constitute an even larger investment in terms of compute, I posit that these studies should be done on a large number of datasets.

[10]Possible options for self-adaptivity in the ES would be the mutation's σ or λ which controls the population size.

models for classification. Theoretically, constant models that output a single class, could be the easiest to explain option. However, this might raise questions on how mixing could be done, but we could of course choose the best performing rule or do majority voting with the best rule serving as a possible tie-breaker. Linear models, e.g. logistic regression, should also be a choice that produces accurate but explainable models. In any case, the fitness functions of rules and solutions would need to use an appropriate objective. Likely, accuracy is not the best option for both as we can expect data to be at least unbalanced for each rule even if it was balanced globally.

Pätzel, Heider, and Hähner [PHH23] proposed a new way of benchmarking LCSs effectively based on how well they are able to approximate data sampled from another LCS model's predictions where both models made similar assumptions, e.g. hyperrectangular matching and linear local models. SupRB should in the future be evaluated in this way in a thorough benchmarking study. Additionally, this benchmark could be included in the wide array of studies that could extend and improve SupRB further.

Of course, I want to urge everyone that became interested in SupRB after reading this thesis to test the system on additional datasets and try its usage within other industries and on other use cases. While including more datasets into a study comes with significant additional computational costs, it would also allow any statements on SupRB's performance to be more definitive and will hopefully help to find the boundaries of when SupRB is the most appropriate model generation method or when other methods and model types should be preferred.

I want to repeat our call from [Hei+23a] that questions relating to the metaheuristics side of LCSs should not be neglected any longer. On one hand, this includes starting to utilize state-of-the-art metaheuristic knowledge and techniques for optimization. On the other, we advocate for deepening the understanding of the relationship between LCS model fitting and LCS model selection. An open question is, for example, how the nature of the model fitting problem in LCSs relates to the nature of the corresponding model selection problem and, consequently, what metaheuristic to use. This begins at picking representations and operators that perform well, and translates to the usage of matching functions and local models. An interesting option could be the integration of modern metaheuristic frameworks such as MAHF [Ste+23; Wur+23], which allow the easy construction of highly flexible optimizers. Especially interesting is the large number of already available operators which can be combined in various new ways. Studies with these optimizers could go far beyond the archetypical approaches we used for SupRB's optimization needs within this thesis. While it is probably too computationally expensive for now, the use of automated algorithm design techniques [e.g. Wur+24] could also provide new and substantially more efficient (and possibly more effective) optimizers for SupRB.

Finally, I want to call for a reimplementation of the most promising aspects of SupRB in a faster language than Python. While Python allows a relatively easy to maintain code structure and can easily be highly modularized, it is also very inefficient in terms of computational overheads. Especially if we want to convince practitioners from industry to adopt SupRB and use it for their needs, we need to provide a library that is at least somewhat fast. This would not only be relevant for industry but will also allow the implemented versions of SupRB to be much more thoroughly benchmarked. Theoretically, many compiled languages are possible options, but C++ and Rust should be the most promising candidates in my view. If the reimplementation focuses on a very bare-bones system, e.g. SupRB with a fixed mixing model, OBR representation of conditions, Ridge local models, ES for RD, and GA for SC with limited operators, the time investment of writing the code should not even be unmanageable. However, a direct transfer of all concepts currently available in Python while maintaining the modular structure would be a considerable effort.

8 Conclusion

In this thesis, I introduced and benchmarked a new rule-based machine learning (RBML) approach that is based on critical research gaps in the field and can serve as the basis of new explainable artificial intelligence (XAI) applications.[1] This last chapter will first lay out the key elements of the preceding chapters and will focus on the major contributions of this work including the essential experimental findings. Then follows a critical discussion on the limitations of this work and potential weaknesses. Finally, I will summarize this work briefly for some closing remarks.

With the increasing capabilities of state-of-the-art methods from the fields of artificial intelligence (AI) and machine learning (ML) specifically, there is an increasing interest in further automation of a wide variety of tasks currently performed by humans. However, these tasks often involve decisions that can have critical effects. For example, doctors might consult intelligent (AI-based) agents on medical decisions, like treatment plans, which could have terminal effects on patients. On the other hand, production planners or other management roles might base major financial decisions on these models, like which parts to produce and where, when, and how to do it. On a smaller scale, individual machine operators might ask the agent for assistance in their day-to-day operations to avoid producing parts that cannot be used and have to be recycled or destroyed (cf. the scenario from Section 3.2 and Section 3.5.1). While the latter might have less global impact, it nonetheless matters to the individual, which often will be responsible for the successful execution of their job and cannot use the model's performance as a justification for poor operations. Overall, this leads to one common revelation: We need stakeholders to trust the models if we want them to be used and benefited from.

A promising path to this necessary level of trust is offered by the concepts of XAI. This field develops methods that make individual models (or algorithms) and their decisions explainable to stakeholders. One common approach is to use methods that are inherently explainable, either by using them to approximate a black-box model that was trained on the task or by directly training an explainable model. Within the explainable models, RBML models are seen as a good compromise between explainability and predictive power, although these models can also become too complex, e.g., by incorporating large quantities of rules, to reasonably use in XAI task.

Learning Classifier Systems (LCSs) are a family of RBML methods that produces models slightly more complex than the very prevalent Decision Trees (DTs) but easier to explain than some of the better performing DT-based methods like Random Forests (RFs) and XGBoost. Therefore,

[1]Most of the individual components of this work have already been partially published before, cf. Section 1.2, but have been polished, detailed, and extended for this thesis.

I believe they could fill a critical niche and essential role in achieving XAI and thus enable ML-based tools for new use cases and with new stakeholders, further increasing automation but also making operations more efficient and effective.

This thesis makes three major contributions as introduced in Section 1.1:

The first contribution, *C1*, is a new perspective on LCSs based on their target models and the optimization tasks involved in creating these, which substitutes the relatively prevalent view of LCSs' algorithmic nature. The essential points of this are found in Chapter 2 which was based on [Hei+23a]. Section 2.1.2 reintroduced the general model structure that is produced by all LCSs and therefore agnostic of training scheme or specific algorithmic elements. Building on this, Section 2.1.3 explains what the classical ML terms of *model selection* and model fitting mean for LCSs and divides these into two subtasks each based on the individual components of an LCS model. In many LCSs, both optimization tasks that make up model selection, and therefore determine the model structure, are performed quasi-simultaneously by multiple—and sometimes competing—components.[2] All LCSs use a metaheuristic to solve model selection and many researchers would argue that this should be an evolutionary algorithm, although Section 2.4 shows several approaches that can reasonably be called LCS but are commonly not. Model fitting is typically done using other heuristics or algorithms, e.g., linear least squares or recursive least squares to fit linear local models for regression tasks. In Section 2.2, we differentiated between LCSs and similar approaches, most notably DTs, Mixture of Experts (MoE), Genetic Programming (GP), and ensemble learning methods like bagging and boosting. The role of metaheuristics in LCSs was specified in Section 2.3, which discussed what representations the metaheuristics typically operate on, what operators they use, and how fitness is often defined in LCSs. The last element of *C1* is a new classification system for LCSs (cf. Section 2.3.1). Traditionally, they are differentiated into Pittsburgh-style, Michigan-style, and hybrid, however this is neither precise nor descriptive as it is based on where the first similar model was proposed rather than on what is done. Therefore, we propose to distinguish between the number of solutions during training and if online or batch learning is performed. With this, most Michigan-style systems fall into the *online single-solution* class while most Pittsburgh-style systems are classified as *batch multi-solution*.

The second contribution, *C2*, is a method to assess the requirements stakeholders have for the explainability of models for a specific use case. This was motivated, introduced, and tested in Chapter 3. Section 3.1 discussed the explainability of RBML on a general level, including the limits and potential issues that arise when using such models for XAI. A potential yet abstracted and generalized scenario for the usage of an XAI-based agent from the field of manufacturing was then introduced in Section 3.2, with Section 3.3 arguing why LCSs are uniquely suited to make up the core component of such an agent. To assess the requirements of stakeholders, Section 3.4 proposed a set of seven questions that should be answered by all relevant stakeholders (or at least a representative sample of each role). While this questionnaire partially focusses on LCSs specifically, three of the questions are quite general, applicable for all

[2]Usually, the model selection tasks do not have individual fitness signals available for their optimization process which is a critical issue this thesis solves within its third contribution, *C3*.

ML models, and most questions can easily be adjusted to other types of models. This template to determine stakeholder preferences is then executed in a real-world case study (cf. Section 3.5) where I interviewed different individuals in different stakeholder roles from a manufacturing plant for a specific application scenario where an operator is advised on configuring line parameters by an intelligent agent to ensure smooth and optimal production even in cases where operator experience is limited. After laying out the answers to each question, a summary was made in Section 3.5.5 which confirmed that the stakeholders would value the explainability of the agent and that statistical measures are insufficient to build trust. Rather, the agent should be using few transparent rules that can easily be evaluated by hand and that any ad-hoc decision should involve as few rules as possible (cf. Section 3.5.6 for more details on what the trained models should look like according to the stakeholders).[3] However, the training process itself is not specifically of interest as long as the model is explainable.

The third contribution of this thesis, *C3*, is the Supervised Rule-based learning system (SupRB). SupRB is a new LCS that trains its models differently than its predecessors by disentangling the two subtasks of model selection by separately solving the discovery of rules and the composition of rules into models.[4] This allows the fitnesses of rules to be independent from each other and the models as a whole and, thus, a much greater control over the optimization process as it is less obscured by interactions. With this, model sizes can be reduced significantly compared to state-of-the-art LCSs. The training process is also more straightforward as the components do not supersede each others effects and it follows clear goals. Section 4.1 comprehensively described SupRB in its basic form which uses a $(1, \lambda)$–Evolution Strategy (ES) for rule discovery (RD) and a Genetic Algorithm (GA) for solution composition (SC). In SupRB, the RD and SC components are invoked sequentially until a termination criterion—typically a set number of RD and SC phases—is reached. RD creates new rules using independent fitness signals based on the rule's in-sample error and volume. The stochastic search process starts more likely where the currently best known solution (created by SC) showed high prediction errors. Individual rules use interval-based matching and linear local models.[5] The generated rules are added to the pool, where they remain unchanged in all subsequent training steps. Then, SC tries to find a (new) subset of the rules in the pool that shows low prediction errors while being of a low model complexity (number of rules). SupRB's computational complexity was shortly discussed in Section 4.2. As SupRB's primary goal is to create models that are easier to explain (satisfying the requirements laid out by Section 3.5.6) than those of previous LCSs, I extensively discuss the strengths and remaining weaknesses and limitations of SupRB in that regard in Section 4.3. I divided my discussion into training (Section 4.3.1), which is most relevant for researchers[6], and the model itself (Section 4.3.2) which is critical for stakeholders

[3]While the results of this study should not be overestimated and other companies or domains might not confirm the specifics, it is quite reasonable to assume that any application that requires explainability will follow a similar trend regarding these requirements. I recommend to always execute the template in the process of implementing an XAI application. Even if no new insights are generated and the results of our study are confirmed, involving relevant stakeholders in the decision-making and design process ensures they feel heard and included, which can already significantly contribute to a positive reception of the system.

[4]In fact, all four optimization subtasks are solved independently and addressed specifically in SupRB.

[5]For now, only a regressor is implemented but I propose how to easily adapt the system to a classifier as well.

[6]At least according to our findings from Section 3.5.

of all backgrounds. I concluded that section with some open questions on how the interface between SupRB and a (non-technical) stakeholder could and should look with reference to the scenario from Section 3.5.

SupRB was then benchmarked on real-world datasets against *XCSF, DT, and RF* (cf. Chapter 5) which were selected because they are established and prevalent RBML approaches. Section 5.1 revisited the explainability of RBML approaches for the purpose of making clear how the models of these four methods can be compared against each other in terms of explainability. The results of the benchmark were presented in Section 5.2. We expected SupRB's models to have better error scores than DT, worse errors than RF, and similar errors to XCSF. We also expected SupRB to have much small model sizes than XCSF and especially RF while being similar or slightly worse in terms of model size than DT. We found this confirmed by our benchmark. We analysed based on descriptive statistics and plotting of distributions and performed Bayesian correlated t-tests to confirm our conclusions. Especially of note was the difference in model complexities between XCSF and SupRB, which differed by orders of magnitude even though some compaction was performed on the XCSF model. We also found that the RBML methods did not exhibit strictly similar trends across datasets, i.e. some perform better on a subset and worse on the others in relation to their peers, and even within datasets the performance is dependent on which data split is made and the random seed that initializes the method. In Table 5.4, I presented an example rule which was chosen from one of SupRB's models and discussed in the surrounding text how it could be analysed and interpreted.

While the performance we found was along the lines we expected, we wanted to test some more options for different components of SupRB to determine if our original choice was sound or if we could find substantial improvements to the system's capabilities of creating small and well-performing models. Chapter 6 details these grouped by components investigated, starting with the rules (conditions/matching functions and local models) and moving to the mixing of multiple matching rules, rule discovery, and, ultimately, the composition of multiple rules into a solution to the learning task.

Regarding the *matching functions* (or *conditions*) of rules (cf. Section 6.1), we investigated different representations (encodings for the optimizer) of the hyperrectangular (interval-based) conditions, finding that the most commonly used options in LCSs, ordered bound (OBR) and center spread (CSR), also work best in SupRB, although our test suggested there might be a small advantage when using OBR. Then, I added a theoretical discussion on other options for matching functions: Hyperellipsoids are the most obvious next option as they are still somewhat human readable and somewhat intuitive and promise a better performance, especially on the surface of the matched hypervolume. Also interesting would be the use of "code fragments" as they might allow a great reusability of models, e.g., between similar but not identical lines in the same manufacturing plant, and can be evaluated by a human. However, these are much harder to read than hyperrectangles.

In SupRB, linear models are used for regression. Constant models are not sufficiently good at approximating real-world data (which would lead to more rules being needed) and more complex options are more difficult to explain to stakeholders which I discussed in Section 6.2

and the relevant sections on explainability. When looking more detailed at local models, I found that we need regularization during this model fitting step to yield effective local models and found Ridge regression to lead to better sets of rules than Lasso regression.

To make a model prediction, the individual predictions of all matching rules have to be combined. There are some different approaches to that but commonly some form of weighted average is made where the weight is based on the performance of the rules. Other options such as simple majority voting for classification, winner-takes-all based on the highest fitness or complex weighting networks (as inherently done in Mixture of Experts models) exist but are often less intuitive or efficient. Assigning the rules their weights constitutes the second subtask of model fitting and all major LCS do this heuristically. In SupRB, we use a mixing model (cf. Equation (4.1)) where we compute a weighted average based on the matching rules' average in-sample error and experience (the number of datapoints seen during training). Both of these metrics are constant for rules in the pool and independent of which specific rules are partaking in the mixing process. Section 6.3 features experiments on possible improvements of this mixing model: Section 3.5 suggested that stakeholders could be overwhelmed when faced with a large number of matching rules and would therefore prefer that at-most three rules partake in the model's overall prediction. We then implemented a mixing model that uses such a limit for one through five rules. We tested three approaches on how to determine which rules should be mixed from all matching rules: selecting the rules according to their fitness, selecting them completely randomly every time, and selecting them randomly based on a fitness-proportionate distribution. Additionally, there is the reasonable assumption that, while more experienced rules should be better at making predictions on their matched subspace and should be assigned a higher weight in the mixing model, there is a reasonable upper cap until which experience accounted for. Rules with experiences orders of magnitude higher than the problems dimensionality are likely not better than other rules that also have large yet numerically lower experiences, e.g., on a ten dimensional problem a linear model trained on 1,000 data points should not be significantly better than one trained on 100 data points sampled from the same distribution. Therefore, they should have the same mixing weight. For this, we tested a cap on experience based on a hard number or on a multiplier of the dimensionality. In both cases, we determined the exact value using our hyperparameter tuning process. We found that limiting the number of rules has severe effects on the performance. A limit of three was clearly outperformed (when testing the model returned by SupRB on holdout data) by the baseline across datasets as confirmed by our statistical testing. However, this might still be a worthwhile trade-off for practical applications in case a real-world test proves that this limit does indeed improve the explainability and therefore likelihood of application, trust, and acceptance. Interestingly, we found that an increase in the limit does not show significant improvements over a limit of three. I suspect that this is mostly because cases with more matching rules (which also make effectively dissimilar predictions) are relatively rare due to the generally low rule counts in SupRB models. But, as this mixing model was worse than the base line, this limit seems to at least hinder the training process. Likely, some rules inside the model are no longer contributing to the fitness signal which makes the optimization process less effective. Furthermore, we found that imposing a maximum on how much the experience factors into the weight has a positive effect. While overall, this effect was relatively small it can be more pronounced on some datasets. In general, larger data sets (with the same dimensionality) benefit more from this, which can make this a worthwhile improvement when applying SupRB in use cases similar to the predictive quality model from Section 3.5 where we typically have hundreds of thousands of data points available.

After investigating options for the components forming SupRB's models directly, I then presented different experiments on the training process of SupRB by proposing and testing various options for the individual optimizers at play.

For the RD component, we benchmarked the originally used $(1, \lambda)$-ES against a form of random search and different (μ, λ)-ES (with elitism) approaches that not only use the fitness as the driving signal but also account for rule novelty, i.e. how far their matched subspaces differ from those of existing rules. We tested three variations of novelty search we adapted from literature and used two different ways of selecting the rules against which the novelty is computed. As the original results [Hei+23c] were not as conclusive as we had liked, I added two additional datasets to this study, finding that, while the eight RD options perform not too far off from each other on a global scale, performance is dataset-dependent. The original ES clearly produces the most compact models (low number of rules) yet is still sufficiently competitive on errors. Although, some of the novelty search options are capable of creating models that are significantly better at making predictions at the cost of model size.

The second optimization task in SupRB, SC, is the selection of a good subset of all discovered rules to be returned as the model after training or to base the RD search's starting points on during training. For this, we originally proposed a traditional GA which we find to work reasonably well. We then investigated additional options in two separate studies:

Initially, we tested a variety of mainstream metaheuristics to replace the GA (cf. Section 6.5). As the original results [Wur+22] were not too conclusive, I also extended the experiment with the additional two real-world datasets from my extended RD investigations. Of the four additional metaheuristics[7] tested, we found Artificial Bee Colony (ABC) to be the most probable candidate to replace the GA. It composed slightly smaller models than the GA while being very similar in terms of prediction errors. I suspect that this is due to its stronger exploration capabilities that allow it to find better intermediate solutions during the alternating phases of SupRB's training process, which in turn leads to a better guidance of the RD process. However, one should note that the other options were also very similar on errors and only slightly—and on some datasets—worse on model sizes than GA and ABC. Overall, I would still recommend to use the GA. It is the most well-known technique, has large numbers of possible extensions and improvements that were proposed in the relevant literature over the last decades, and evolutionary techniques are the traditional choice for LCSs.

In the latest publication on SupRB [Hei+24], we investigated the use of different Self-adaptive GAs (SAGAs) in SupRB. Currently, there are quite numerous hyperparameters to tune due to the two optimizers and, of course, the remaining system, e.g., the number of phases, the

[7]We also included another version of random search as a baseline and found it to perform notably worse than the others.

number of rules per RD phase, etc. The usage of self-adaptive methods promises not only the possibility of better overall fitnesses of solutions but also the replacement of hyperparameters which have to be tuned correctly to allow good solutions to the learning task. This can save on overall computation budgets, but can also make the system easier to use for non-experts which is a critical feature to get it used within industry in day-to-day operations. In Section 6.6, we adapted four different SAGAs from literature to fit to our optimization task. In the experiments, we found that three of the options perform reasonably well. SAGA2 and SAGA3 are on-par with the GA in terms of errors, while SAGA4 falls behind on one dataset. However, SAGA4 produced the most compact models on all datasets and showed significantly smaller models on the one dataset it fell behind on on errors. Likely, its mechanisms lead it to follow paths within the fitness landscape that lead to another segment of the Pareto front.

Chapter 7 presented a large number of the many possible next steps for future research. I highlighted directions regarding explainability, practical application, additional learning paradigms (i.e. classification), algorithmic extensions, additional options similar to the studies within Chapter 6, possible benchmarking opportunities, and integration into another metaheuristic framework.

In this summary of results, I highlighted many of the strengths and achievements of this thesis: Basically, the work that culminated into this, highlights and then closes relevant research gaps towards XAI applications in the real-world while still making fundamental contributions. SupRB is a robust and effective RBML system that produces generally explainable models according to specific requirements.

However, this work did not answer all questions or "solve" XAI. Due to their sheer number, I was unable to address—let alone benchmark against—all other LCSs out there. There might also be more approaches that could fill the same gap in the XAI field if they were modernized and developed further.

While I believe the questionnaire to be a useful template for assessing requirements for XAI, it is geared towards LCSs and other approaches might work as well but need substantial adjustments to the questions. Moreover, it might be possible to create a more general template to determine what type of model is needed, although I would argue that this is mostly a question of what can approximate the data sufficiently. If a DT with reasonably low depth is able to make adequate predictions, this is the simpler model and should be preferred. While it is possible to adapt the template to determine the design of DT models for a use case, some things like bad predictions before depths of 20 or more could be used to already exclude DTs from consideration before interviewing stakeholders. Additionally, we only presented a single use case and its requirements in the articles on which Chapter 3 was based. Other use cases, especially from other domains or outside of manufacturing, might yield different answers even though I believe general trends regarding model design will be commonplace as long as XAI is required at all. Theoretically, the questionnaire should work in other domains as well and SupRB is of course a universal function approximator and applicable everywhere but I believe it is important to verify the method.

SupRB is not universally explainable. I would argue that it is easier to explain and understand than previous LCSs (especially when following the solution trajectories) but the search process is still stochastic and involves many steps. SupRB's models are much smaller than those of many other LCSs but they are also more complex than a DT with a similar number of rules and of course much more complex than a linear model. In general, the overlapping nature of LCS rules is a clear trade-off between more accurate predictions and model explainability. High-dimensional rules can be difficult to analyse even if based on hyperrectangular conditions and linear models as humans struggle with imagining anything beyond three-dimensional space. As Table 5.4 showed, even more general rules will only match a small share of the overall feature space. At the moment, there are no well-developed and rigorously tested techniques on how to present LCS models or even only individual rules to stakeholders—especially non-experts and non-technical stakeholders. Despite the likely advantages of SupRB's models, there is a non-zero chance that the most relevant stakeholders could not be convinced and an application fails.[8]

This thesis did also not include a thorough comparison of the explainability of the four RBML methods (and further variations of SupRB) to the specific stakeholders from Section 3.5. Therefore, I want to reiterate my call from Chapter 7 to perform such a study and publish it for the benefit of other researchers. Additionally, I hope we will see the development of new methods on how methods like SupRB or other LCSs can improve operations in practical applications, including how they can be effectively explained, ideally tested in live systems followed by a detailed presentation in a prominent publication.

The benchmarks of SupRB were only made on a low number of real-world datasets. While they were selected for sound reasoning (based on their size, dimensionality, and linearity) and allow a relevant evaluation of the system, the statements made could be even stronger if evaluated on more datasets. Especially in cases of narrow performance differences, it would be interesting to see if these hold on new data, even though I do not expect significant changes in the general trends seen in all experiments. For this reason, I included two additional datasets in the investigations on rule conditions, RD, and SC, where the tests were not too conclusive on the datasets from Chapter 5 alone. However, more tests would always be preferable even though they take significant computation budgets to perform and time to analyse.

We never tested hyperellipsoids or other more complex options for rule conditions (or local models). While we know that they are more complex to explain, we have no firm understanding on how much performance we are missing out on and if (and by how much) stakeholders would even consider other options to be a loss in terms of explainability. For example, if they only cared about specific instances (e.g., the current situation) they might not even care to know why these rules match, rendering the matching function irrelevant. On the other hand, it is possible that the gains from these other functions are minimal or even non-existent as they harm the training process due to the optimization becoming more difficult as well.

[8]There is of course the possibility of a stakeholder generally rejecting all forms of "AI" or recommender system but that would not be SupRB's sole responsibility.

We did not manage to find optimizers that improved the performance of the system by a lot. Now, the original choices on optimizers and system design were made based on sound assumptions and background knowledge and were made with care, however, it is somewhat disappointing that none of the changes could lift the performance considerably.

There might be combinations of the different options presented in Chapter 6 that yield better performances and a thorough hyperparameter study could also find some improvements to SupRB. Especially after the explainability of SupRB is validated in practice, a comprehensive guide on how to configure SupRB, together with a faster implementation, would make an even better case for practical use of the system in industry than the results I presented alone.

In conclusion, this thesis advanced the state-of-the-art in the field of XAI which is critical to move ML towards widespread application. It did so with a focus on LCSs by advancing the theoretical understanding, proposing and demonstrating a method to assess specific design requirements based on the explainability of models, and by culminating these findings in a new system that produces models that are much smaller yet similarly accurate than its competition. This system was successfully benchmarked on real-world data and advanced considerably over a number of publications tackling individual components. Therefore, it can form a well-rounded basis for further studies on XAI within interactive scenarios with real stakeholders and, ultimately, the advancement of workplace automation.

Bibliography

[Aki+19] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 2623–2631. ISBN: 978-1-4503-6201-6. DOI: 10/gf7mzz.

[Ala12] Bilal Alatas. "A novel chemistry based metaheuristic optimization method for mining of classification rules". In: *Expert Systems with Applications* 39.12 (Sept. 2012), pp. 11080–11088. DOI: 10.1016/j.eswa.2012.03.066.

[ALF99] Dieferson L. Alves de Araujo, Heitor S. Lopes, and Alex A. Freitas. "A parallel genetic algorithm for rule discovery in large databases". In: *IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.99CH37028).* IEEE, 1999. DOI: 10.1109/ICSMC.1999.823354.

[AM14a] Sarab AlMuhaideb and Mohamed El Bachir Menai. "A new hybrid metaheuristic for medical data classification". In: *International Journal of Metaheuristics* 3.1 (2014), p. 59. DOI: 10.1504/ijmheur.2014.058860.

[AM14b] Sarab AlMuhaideb and Mohamed El Bachir Menai. "HColonies: a new hybrid metaheuristic for medical data classification". In: *Applied Intelligence* 41.1 (Feb. 2014), pp. 282–298. DOI: 10.1007/s10489-014-0519-z.

[Ara+22] Claus Aranha et al. "Metaphor-based metaheuristics, a call for action: the elephant in the room". In: *Swarm Intelligence* 16.1 (Mar. 1, 2022), pp. 1–6. ISSN: 1935-3820. DOI: 10.1007/s11721-021-00202-9.

[AS12] Basheer M. Al-Maqaleh and Hamid Shahbazkia. "A Genetic Algorithm for Discovering Classification Rules in Data Mining". In: *International Journal of Computer Applications* 41.18 (Mar. 2012), pp. 40–44. DOI: 10.5120/5644-8072.

[AS17] Zulfiqar Ali and Waseem Shahzad. "Comparative Analysis and Survey of Ant Colony Optimization based Rule Miners". In: *International Journal of Advanced Computer Science and Applications* 8.1 (2017). DOI: 10.14569/ijacsa.2017.080108.

[Bac+22] Jaume Bacardit, Alexander E. I. Brownlee, Stefano Cagnoni, Giovanni Iacca, John McCall, and David Walker. "The intersection of evolutionary computation and explainable AI". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion.* ACM, July 2022. DOI: 10.1145/3520304.3533974.

[Bac04] Jaume Bacardit. “Pittsburgh Genetic-Based Machine Learning in the Data Mining Era: Representations, Generalization, and Run-Time”. PhD thesis. Ramon Llull University, Barcelona, 2004.

[Bar+12] Rodrigo C. Barros, Márcio P. Basgalupp, André C.P.L.F. de Carvalho, and Alex A. Freitas. “A hyper-heuristic Evolutionary Algorithm for Automatically Designing Decision Tree Algorithms”. In: *Proceedings of the Fourteenth International Conference on Genetic and Evolutionary Computation - GECCO ’12*. ACM Press, 2012. DOI: 10.1145/2330163.2330335.

[Bar+20] Alejandro Barredo Arrieta et al. “Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI”. In: *Information Fusion* 58 (2020), pp. 82–115. ISSN: 1566-2535. DOI: 10.1016/j.inffus.2019.12.012.

[Bar02] Alwyn Barry. “The stability of long action chains in XCS”. In: *Soft Comput.* 6.3-4 (2002), pp. 183–199. DOI: 10.1007/s005000100115.

[BBM20] Marco Barsacchi, Alessio Bechini, and Francesco Marcelloni. “An analysis of boosted ensembles of binary fuzzy decision trees”. In: *Expert Systems with Applications* 154 (2020). ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2020.113436.

[Ben+17] Alessio Benavoli, Giorgio Corani, Janez Demšar, and Marco Zaffalon. “Time for a Change: A Tutorial for Comparing Multiple Classifiers through Bayesian Analysis”. In: *J. Mach. Learn. Res.* 18.1 (2017), pp. 2653–2688. ISSN: 1532-4435.

[BFM97] Thomas Bäck, D. B. Fogel, and Z. Michalewicz, eds. *Handbook of Evolutionary Computation*. CRC Press, Jan. 1997. DOI: 10.1201/9780367802486.

[BG03] Ester Bernadó-Mansilla and Josep M. Garrell-Guiu. “Accuracy-Based Learning Classifier Systems: Models, Analysis and Applications to Classification Tasks”. In: *Evolutionary Computation* 11.3 (Sept. 2003), pp. 209–238. ISSN: 1063-6560. DOI: 10.1162/106365603322365289.

[BG07] Jaume Bacardit and Josep Maria Garrell. “Bloat Control and Generalization Pressure Using the Minimum Description Length Principle for a Pittsburgh Approach Learning Classifier System”. In: *Learning Classifier Systems*. Ed. by Tim Kovacs, Xavier Llorà, Keiki Takadama, Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 59–79. ISBN: 978-3-540-71231-2.

[BGB22] Jordan T. Bishop, Marcus Gallagher, and Will N. Browne. “Pittsburgh learning classifier systems for explainable reinforcement learning: comparing with XCS”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO ’22. Boston, Massachusetts: Association for Computing Machinery, 2022, pp. 323–331. ISBN: 9781450392372. DOI: 10.1145/3512290.3528767.

[BH03] Larry Bull and Jacob Hurst. “A neural learning classifier system with self-adaptive constructivism”. In: *The 2003 Congress on Evolutionary Computation, 2003. CEC ’03*. Vol. 2. 2003, 991–997 Vol. 2.

[Bis09] Christopher M. Bishop. *Pattern recognition and machine learning, 8th Edition*. Information science and statistics. Springer, 2009. ISBN: 9781493938438.

[BK06] Jaume Bacardit and Natalio Krasnogor. *BioHEL: Bioinformatics-oriented Hierarchical Evolutionary Learning*. Tech Report available at http://eprints.nottingham.ac.uk/id/eprint/482. 2006.

[BK09] Urszula Boryczka and Jan Kozak. "New Algorithms for Generation Decision Trees—Ant-Miner and Its Modifications". In: *Studies in Computational Intelligence*. Springer Berlin Heidelberg, 2009, pp. 229–262. DOI: 10.1007/978-3-642-01091-0_11.

[BK10] Urszula Boryczka and Jan Kozak. "Ant Colony Decision Trees – A New Method for Constructing Decision Trees Based on Ant Colony Optimization". In: *Computational Collective Intelligence. Technologies and Applications*. Springer Berlin Heidelberg, 2010, pp. 373–382. DOI: 10.1007/978-3-642-16693-8_39.

[BLW08] Martin V. Butz, Pier Luca Lanzi, and Stewart W. Wilson. "Function Approximation With XCS: Hyperellipsoidal Conditions, Recursive Least Squares, and Compaction". In: *IEEE Transactions on Evolutionary Computation* 12.3 (2008), pp. 355–376. DOI: 10.1109/TEVC.2007.903551.

[BO02] Larry Bull and Toby O'Hara. "Accuracy-Based Neuro and Neuro-Fuzzy Classifier Systems". In: *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*. GECCO'02. New York City, New York: Morgan Kaufmann Publishers Inc., 2002, pp. 905–911. ISBN: 1558608788.

[BO15] James Brookhouse and Fernando E. B. Otero. "Discovering Regression Rules with Ant Colony Optimization". In: *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM, July 2015. DOI: 10.1145/2739482.2768450.

[BP21] Vaishak Belle and Ioannis Papantonis. "Principles and Practice of Explainable Machine Learning". In: *Frontiers in Big Data* 4 (2021). ISSN: 2624-909X. DOI: 10.3389/fdata.2021.688969.

[BPM89] Thomas Brooks, Dennis Pope, and Michael Marcolini. *Airfoil Self-Noise and Prediction*. NASA Reference Publication 1218. 1989.

[Bre+84] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification And Regression Trees*. Routledge, Oct. 1984. DOI: 10.1201/9781315139470.

[Bre96] Leo Breiman. "Bagging Predictors". In: *Mach. Learn.* 24.2 (1996), pp. 123–140. DOI: 10.1007/BF00058655.

[BS02] Martin V. Butz and Wolfgang Stolzmann. "An Algorithmic Description of ACS2". In: *Advances in Learning Classifier Systems*. Ed. by Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 211–229. ISBN: 978-3-540-48104-1.

[But05] Martin V. Butz. “Kernel-Based, Ellipsoidal Conditions in the Real-Valued XCS Classifier System”. In: *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation.* GECCO ’05. Washington DC, USA: Association for Computing Machinery, 2005, pp. 1835–1842. ISBN: 1595930108. DOI: 10.1145/1068009.1068320.

[CA23] Felipe Campelo and Claus Aranha. “Lessons from the Evolutionary Computation Bestiary”. In: *Artificial Life* 29.4 (Nov. 2023), pp. 421–432. ISSN: 1064-5462. DOI: 10.1162/artl_a_00402.

[Cal+19] Borja Calvo, Ofer M. Shir, Josu Ceberio, Carola Doerr, Hao Wang, Thomas Bäck, and Jose A. Lozano. “Bayesian Performance Analysis for Black-Box Optimization Benchmarking”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion.* GECCO ’19. Prague, Czech Republic: Association for Computing Machinery, 2019, pp. 1789–1797. ISBN: 9781450367486. DOI: 10.1145/3319619.3326888.

[CB15] Giorgio Corani and Alessio Benavoli. “A Bayesian approach for comparing cross-validated algorithms on multiple data sets”. In: *Machine Learning* 100.2 (Sept. 2015), pp. 285–304. ISSN: 1573-0565. DOI: 10.1007/s10994-015-5486-z.

[CCL18] Borja Calvo, Josu Ceberio, and Jose A. Lozano. “Bayesian Inference for Algorithm Ranking Analysis”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion.* GECCO ’18. Kyoto, Japan: Association for Computing Machinery, 2018, pp. 324–325. ISBN: 9781450357647. DOI: 10.1145/3205651.3205658.

[CI23] Leonardo L. Custode and Giovanni Iacca. “Evolutionary Learning of Interpretable Decision Trees”. In: *IEEE Access* 11 (2023), pp. 6169–6184. DOI: 10.1109/ACCESS.2023.3236260.

[CJM12] Ivan Chorbev, Boban Joksimoski, and Dragan Mihajlov. “SA Tabu Miner: A hybrid heuristic algorithm for rule induction”. In: *Intelligent Decision Technologies* 6 (2012), pp. 265–271. ISSN: 18758843, 18724981. DOI: 10.3233/IDT-2012-0142.

[CPC19] Diogo V. Carvalho, Eduardo M. Pereira, and Jaime S. Cardoso. “Machine Learning Interpretability: A Survey on Methods and Metrics”. In: *Electronics* 8.8 (July 2019), p. 832. DOI: 10.3390/electronics8080832.

[DAL05] Hai H. Dam, Hussein A. Abbass, and Chris Lokan. “Be Real! XCS with Continuous-Valued Inputs”. In: *Proceedings of the 7th Annual Workshop on Genetic and Evolutionary Computation.* GECCO ’05. Washington, D.C.: Association for Computing Machinery, 2005, pp. 85–87. ISBN: 9781450378000. DOI: 10.1145/1102256.1102274.

[DBS06] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. “Ant Colony Optimization”. In: *IEEE Computational Intelligence Magazine* 1.4 (Nov. 2006), pp. 28–39. ISSN: 1556-6048. DOI: 10/dq339r.

[DD99] Marco Dorigo and Gianni Di Caro. “Ant Colony Optimization: A New Meta-Heuristic”. In: *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406).* Vol. 2. July 1999, 1470–1477 Vol. 2. DOI: 10/b5h9xz.

[DeJ88] Kenneth DeJong. “Learning with Genetic Algorithms: An Overview”. In: *Mach. Learn.* 3 (1988), pp. 121–138. DOI: 10.1007/BF00113894.

[Den18] Houtao Deng. “Interpreting tree ensembles with inTrees”. In: *International Journal of Data Science and Analytics* 7.4 (July 2018), pp. 277–287. DOI: 10.1007/s41060-018-0144-8.

[DG17] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository.* 2017.

[Die00] Thomas G. Dietterich. “Ensemble Methods in Machine Learning”. In: *Multiple Classifier Systems.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 1–15. ISBN: 978-3-540-45014-6.

[DJ19] Clarisse Dhaenens and Laetitia Jourdan. “Metaheuristics for data mining”. In: *4OR* 17.2 (Apr. 2019), pp. 115–139. DOI: 10.1007/s10288-019-00402-4.

[DK18] Finale Doshi-Velez and Been Kim. “Considerations for Evaluation and Generalization in Interpretable Machine Learning”. In: *The Springer Series on Challenges in Machine Learning*. Springer International Publishing, 2018, pp. 3–17. DOI: 10.1007/978-3-319-98131-4_1.

[Dru07] Jan Drugowitsch. “Learning classifier systems from first principles: a probabilistic reformulation of learning classifier systems from the perspective of machine learning”. PhD thesis. University of Bath (United Kingdom), 2007.

[Dru08] Jan Drugowitsch. *Design and Analysis of Learning Classifier Systems - A Probabilistic Approach.* Vol. 139. Studies in Computational Intelligence. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

[DS04] Marco Dorigo and Thomas Stützle. *Ant colony optimization.* Cambridge, Mass: MIT Press, 2004. ISBN: 9780262042192.

[DS15] Ren Diao and Qiang Shen. “Nature Inspired Feature Selection Meta-Heuristics”. In: *Artificial Intelligence Review* 44.3 (Oct. 2015), pp. 311–340. ISSN: 1573-7462. DOI: 10/gn732t.

[DTY18] Jie Ding, Vahid Tarokh, and Yuhong Yang. “Model Selection Techniques: An Overview”. In: *IEEE Signal Processing Magazine* 35.6 (Nov. 2018), pp. 16–34. ISSN: 1558-0792. DOI: 10/gf8ck7.

[EBK11] Narayanan Unny Edakunni, Gavin Brown, and Tim Kovacs. “Online, GA based mixture of experts: a probabilistic model of UCS”. In: *13th Annual Genetic and Evolutionary Computation Conference, GECCO 2011, Proceedings, Dublin, Ireland, July 12-16, 2011.* Ed. by Natalio Krasnogor and Pier Luca Lanzi. New York, NY, USA: ACM, 2011, pp. 1267–1274.

[Eda+09] Narayanan Unny Edakunni, Tim Kovacs, Gavin Brown, and James A. R. Marshall. “Modeling UCS As a Mixture of Experts”. In: *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation.* GECCO ’09. New York, NY, USA: ACM, 2009, pp. 1187–1194.

[EZH16] Eid Emary, Hossam M. Zawbaa, and Aboul Ella Hassanien. “Binary Grey Wolf Optimization Approaches for Feature Selection”. In: *Neurocomputing* 172 (Jan. 2016), pp. 371–381. ISSN: 0925-2312. DOI: 10/gmf2xg.

[FCI23] Andrea Ferigo, Leonardo Lucio Custode, and Giovanni Iacca. “Quality Diversity Evolutionary Learning of Decision Trees”. In: *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*. SAC ’23. Tallinn, Estonia: Association for Computing Machinery, 2023, pp. 425–432. ISBN: 9781450395175. DOI: 10.1145/3555776.3577591.

[FKB13] María A. Franco, Natalio Krasnogor, and Jaume Bacardit. “GAssist vs. BioHEL: critical assessment of two paradigms of genetics-based machine learning”. In: *Soft Comput.* 17.6 (2013), pp. 953–981. DOI: 10.1007/s00500-013-1016-8.

[FPP86] J. Doyne Farmer, Norman H. Packard, and Alan S. Perelson. “The immune system, adaptation, and machine learning”. In: *Physica D: Nonlinear Phenomena* 22.1-3 (Oct. 1986), pp. 187–204. DOI: 10.1016/0167-2789(86)90240-x.

[Fre02] Alex A. Freitas. *Data Mining and Knowledge Discovery with Evolutionary Algorithms.* Springer Berlin Heidelberg, 2002. DOI: 10.1007/978-3-662-04923-5.

[Fre03] Alex A. Freitas. “A Survey of Evolutionary Algorithms for Data Mining and Knowledge Discovery”. In: *Natural Computing Series.* Springer Berlin Heidelberg, 2003, pp. 819–845. DOI: 10.1007/978-3-642-18965-4_33.

[FS96] Yoav Freund and Robert E. Schapire. “Experiments with a New Boosting Algorithm”. In: *Machine Learning, Proceedings of the Thirteenth International Conference (ICML ’96), Bari, Italy, July 3-6, 1996.* Ed. by Lorenza Saitta. Morgan Kaufmann, 1996, pp. 148–156.

[Gar05] Simon M. Garrett. “How Do We Evaluate Artificial Immune Systems?” In: *Evolutionary Computation* 13.2 (June 2005), pp. 145–177. DOI: 10 . 1162 / 1063656054088512.

[GMC15] Jorge Gomes, Pedro Mariano, and Anders Lyhne Christensen. “Devising Effective Novelty Search Algorithms: A Comprehensive Empirical Study”. In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation.* GECCO ’15. Madrid, Spain: Association for Computing Machinery, 2015, pp. 943–950. ISBN: 9781450334723. DOI: 10.1145/2739480.2754736.

[Gol89] David Goldberg. *Genetic algorithms in search, optimization, and machine learning.* Reading, Mass: Addison-Wesley Publishing Company, 1989. ISBN: 0201157675.

[GUC12] Jorge Gomes, Paulo Urbano, and Anders Lyhne Christensen. “Progressive Minimal Criteria Novelty Search”. In: *Advances in Artificial Intelligence – IBERAMIA 2012.* Ed. by Juan Pavón, Néstor D. Duque-Méndez, and Rubén Fuentes-Fernández. Springer Berlin Heidelberg, 2012, pp. 281–290.

[Gup+17] Preeti Gupta, Tarun Kumar Sharma, Deepti Mehrotra, and Ajith Abraham. "Knowledge building through optimized classification rule set generation using genetic based elitist multi objective approach". In: *Neural Computing and Applications* 31.S2 (May 2017), pp. 845–855. DOI: 10.1007/s00521-017-3042-4.

[HB01] Jacob Hurst and Larry Bull. "A Self-Adaptive Classifier System". In: *Advances in Learning Classifier Systems*. Ed. by Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 70–79. ISBN: 978-3-540-44640-8.

[HB02] Jacob Hurst and Larry Bull. "A Self-Adaptive XCS". In: *Advances in Learning Classifier Systems*. Ed. by Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 57–73. ISBN: 978-3-540-48104-1.

[HBL09] Gerard David Howard, Larry Bull, and Pier-Luca Lanzi. "Towards continuous actions in continuous space and time using self-adaptive constructivism in neural XCSF". In: *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*. GECCO '09. Montreal, Québec, Canada: Association for Computing Machinery, 2009, pp. 1219–1226. ISBN: 9781605583259. DOI: 10.1145/1569901.1570065.

[Hei+22a] Michael Heider, Helena Stegherr, David Pätzel, Roman Sraj, Jonathan Wurth, Benedikt Volger, and Jörg Hähner. "Approaches for Rule Discovery in a Learning Classifier System". In: *Proceedings of the 14th International Joint Conference on Computational Intelligence - ECTA,* INSTICC. Setúbal, Portugal: SciTePress, 2022, pp. 39–49. DOI: 10.5220/0011542000003332.

[Hei+22b] Michael Heider, Helena Stegherr, Jonathan Wurth, Roman Sraj, and Jörg Hähner. "Investigating the Impact of Independent Rule Fitnesses in a Learning Classifier System". In: *Bioinspired Optimization Methods and Their Applications*. Ed. by Marjan Mernik, Tome Eftimov, and Matej Črepinšek. Cham: Springer International Publishing, 2022, pp. 142–156. ISBN: 978-3-031-21094-5. DOI: 10.1007/978-3-031-21094-5_11.

[Hei+22c] Michael Heider, Helena Stegherr, Jonathan Wurth, Roman Sraj, and Jörg Hähner. "Separating Rule Discovery and Global Solution Composition in a Learning Classifier System". In: *Genetic and Evolutionary Computation Conference Companion (GECCO '22 Companion)*. 2022. DOI: 10.1145/3520304.3529014.

[Hei+23a] Michael Heider, David Pätzel, Helena Stegherr, and Jörg Hähner. "A Metaheuristic Perspective on Learning Classifier Systems". In: *Metaheuristics for Machine Learning: New Advances and Tools*. Ed. by Mansour Eddaly, Bassem Jarboui, and Patrick Siarry. Singapore: Springer Nature Singapore, 2023, pp. 73–98. ISBN: 978-981-19-3888-7. DOI: 10.1007/978-981-19-3888-7_3.

[Hei+23b] Michael Heider, Helena Stegherr, Richard Nordsieck, and Jörg Hähner. “Assessing Model Requirements for Explainable AI: A Template and Exemplary Case Study”. In: *Artificial Life* 29.4 (2023), pp. 468–486. ISSN: 1064-5462. DOI: 10.1162/artl_a_00414.

[Hei+23c] Michael Heider, Helena Stegherr, David Pätzel, Roman Sraj, Jonathan Wurth, Benedikt Volger, and Jörg Hähner. “Discovering Rules for Rule-Based Machine Learning with the Help of Novelty Search”. In: *SN Computer Science* 4.6 (2023), p. 778. ISSN: 2661-8907. DOI: 10.1007/s42979-023-02198-x.

[Hei+23d] Michael Heider, Helena Stegherr, Roman Sraj, David Pätzel, Jonathan Wurth, and Jörg Hähner. “SupRB in the context of rule-based machine learning methods: A comparative study”. In: *Applied Soft Computing* 147 (2023), p. 110706. ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2023.110706.

[Hei+24] Michael Heider, Maximilian Krischan, Roman Sraj, and Jörg Hähner. “Exploring Self-Adaptive Genetic Algorithms to Combine Compact Sets of Rules”. In: *2024 IEEE Congress on Evolutionary Computation (CEC)*. 2024, pp. 1–8. DOI: 10.1109/CEC60901.2024.10612101.

[Hei24] Michael Heider. “Towards Self-Explaining Assistance Systems in Tomorrow’s Factories”. In: *Organic Computing—Doctoral Dissertation Colloquium 2023*. Ed. by Sven Tomforde and Christian Krupitzer. Kassel University Press, 2024. DOI: 10.17170/kobra-202402269661.

[Her+22] Lukas-Valentin Herm, Kai Heinrich, Jonas Wanner, and Christian Janiesch. “Stop ordering machine learning algorithms by their explainability! A user-centered investigation of performance and explainability”. In: *International Journal of Information Management* (June 2022), p. 102538. DOI: 10.1016/j.ijinfomgt.2022.102538.

[HF05] Nicholas Holden and Alex A. Freitas. “A hybrid particle swarm/ant colony algorithm for the classification of hierarchical biological data”. In: *Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005*. IEEE, 2005. DOI: 10.1109/SIS.2005.1501608.

[HF08] Nicholas Holden and Alex A. Freitas. “A Hybrid PSO/ACO Algorithm for Discovering Classification Rules in Data Mining”. In: *Journal of Artificial Evolution and Applications* 2008 (May 2008), pp. 1–11. DOI: https://doi.org/10.1155/2008/316145.

[HGA20] Julian Hatwell, Mohamed Medhat Gaber, and R. Muhammad Atif Azad. “CHIRPS: Explaining random forest classification”. In: *Artificial Intelligence Review* 53.8 (June 2020), pp. 5747–5788. DOI: 10.1007/s10462-020-09833-6.

[HNH21] Michael Heider, Richard Nordsieck, and Jörg Hähner. “Learning Classifier Systems for Self-Explaining Socio-Technical-Systems”. In: *Proceedings of LIFELIKE 2021 co-located with 2021 Conference on Artificial Life (ALIFE 2021)*. Ed. by Anthony Stein, Sven Tomforde, Jean Botev, and Peter Lewis. 2021.

[Hol76] John H. Holland. “Adaptation”. In: *Progress in theoretical biology* 4 (1976). Ed. by Robert Rosen and Fred M. Snell, pp. 263–293.

[Hol92] John H. Holland. “Genetic Algorithms”. In: *Scientific American* 267.1 (1992), pp. 66–73. ISSN: 0036-8733. DOI: 10/bmbqnb.

[HPH20] Michael Heider, David Pätzel, and Jörg Hähner. “Towards a Pittsburgh-style LCS for learning manufacturing machinery parametrizations”. In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion.* ACM, July 2020. DOI: 10.1145/3377929.3389963.

[HPW22] Michael Heider, David Pätzel, and Alexander R. M. Wagner. “An overview of LCS research from 2021 to 2022”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion.* 2022, pp. 2086–2094. DOI: 10.1145/3520304.3533985.

[HT08] Emma Hart and Jon Timmis. “Application areas of AIS: The past, the present and the future”. In: *Applied Soft Computing* 8.1 (Jan. 2008), pp. 191–201. DOI: 10.1016/j.asoc.2006.12.004.

[IBZ12] Muhammad Iqbal, Will N. Browne, and Mengjie Zhang. “XCSR with Computed Continuous Action”. In: *AI 2012: Advances in Artificial Intelligence.* Ed. by Michael Thielscher and Dongmo Zhang. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 350–361. ISBN: 978-3-642-35101-3.

[IBZ14a] Muhammad Iqbal, Will N. Browne, and Mengjie Zhang. “Reusing Building Blocks of Extracted Knowledge to Solve Complex, Large-Scale Boolean Problems”. In: *IEEE Transactions on Evolutionary Computation* 18.4 (2014), pp. 465–480. DOI: 10.1109/TEVC.2013.2281537.

[IBZ14b] Muhammad Iqbal, Will N. Browne, and Mengjie Zhang. “Reusing Building Blocks of Extracted Knowledge to Solve Complex, Large-Scale Boolean Problems”. In: *IEEE Transactions on Evolutionary Computation* 18.4 (2014), pp. 465–480. DOI: 10.1109/TEVC.2013.2281537.

[Jac+91] Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. “Adaptive Mixtures of Local Experts”. In: *Neural Computation* 3.1 (1991), pp. 79–87. DOI: 10.1162/neco.1991.3.1.79.

[Jan98] Cezary Z. Janikow. “Fuzzy decision trees: issues and methods”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 28.1 (1998), pp. 1–14. DOI: 10.1109/3477.658573.

[JDK14] Dongli Jia, Xintao Duan, and Muhammad Khurram Khan. “Binary Artificial Bee Colony Optimization Using Bitwise Operation”. In: *Computers & Industrial Engineering* 76 (Oct. 2014), pp. 360–365. ISSN: 0360-8352. DOI: 10/f6npmn.

[JJ94] Michael I. Jordan and Robert A. Jacobs. “Hierarchical Mixtures of Experts and the EM Algorithm”. In: *Neural Computation* 6.2 (1994), pp. 181–214. DOI: 10.1162/neco.1994.6.2.181.

[JLZ06] Licheng Jiao, Jing Liu, and Weicai Zhong. “An organizational coevolutionary algorithm for classification”. In: *IEEE Transactions on Evolutionary Computation* 10.1 (Feb. 2006), pp. 67–80. DOI: 10.1109/TEVC.2005.856068.

[Kar+14] Dervis Karaboga, Beyza Gorkemli, Celal Ozturk, and Nurhan Karaboga. "A Comprehensive Survey: Artificial Bee Colony (ABC) Algorithm and Applications". In: *Artificial Intelligence Review* 42.1 (June 2014), pp. 21–57. ISSN: 1573-7462. DOI: 10/f53fbk.

[Kau+23] Davinder Kaur, Suleyman Uslu, Kaley J. Rittichier, and Arjan Durresi. "Trustworthy Artificial Intelligence: A Review". In: *ACM Computing Surveys* 55.2 (Mar. 2023), pp. 1–38. DOI: 10.1145/3491209.

[KB07] Dervis Karaboga and Bahriye Basturk. "A Powerful and Efficient Algorithm for Numerical Function Optimization: Artificial Bee Colony (ABC) Algorithm". In: *Journal of Global Optimization* 39.3 (Nov. 2007), pp. 459–471. ISSN: 1573-2916. DOI: 10/c25dm6.

[KE95] James Kennedy and Russell Eberhart. "Particle Swarm Optimization". In: *Proceedings of ICNN'95 - International Conference on Neural Networks.* Vol. 4. Nov. 1995, 1942–1948 vol.4. DOI: 10/bdc3t3.

[KE97] James Kennedy and Russell C. Eberhart. "A Discrete Binary Version of the Particle Swarm Algorithm". In: *Computational Cybernetics and Simulation 1997 IEEE International Conference on Systems, Man, and Cybernetics.* Vol. 5. Oct. 1997, 4104–4108 vol.5. DOI: 10/b7gp8s.

[KFN03] Juha Kivijärvi, Pasi Fränti, and Olli Nevalainen. "Self-Adaptive Genetic Algorithm for Clustering". In: *Journal of Heuristics* 9.2 (2003), pp. 113–129. ISSN: 1572-9397. DOI: 10.1023/A:1022521428870.

[KN13] Shima Kashef and Hossein Nezamabadi-pour. "A New Feature Selection Algorithm Based on Binary Ant Colony Optimization". In: *The 5th Conference on Information and Knowledge Technology.* May 2013, pp. 50–54. DOI: 10/gnxcc4.

[Kov12] Tim Kovacs. "Genetics-Based Machine Learning". In: *Handbook of Natural Computing.* Ed. by Grzegorz Rozenberg, Thomas Bäck, and Joost N. Kok. Berlin, Heidelberg: Springer, 2012, pp. 937–986. ISBN: 978-3-540-92910-9. DOI: 10.1007/978-3-540-92910-9_30.

[Koz93] John R. Koza. *Genetic programming - on the programming of computers by means of natural selection.* Complex adaptive systems. MIT Press, 1993. ISBN: 978-0-262-11170-6.

[Kra17] Oliver Kramer. "Genetic Algorithms". In: *Genetic Algorithm Essentials.* Ed. by Oliver Kramer. Studies in Computational Intelligence. Cham: Springer International Publishing, 2017, pp. 11–19. ISBN: 978-3-319-52156-5. DOI: 10.1007/978-3-319-52156-5_2.

[Kru+22] Christian Krupitzer, Christian Gruhl, Bernhard Sick, and Sven Tomforde. "Proactive hybrid learning and optimisation in self-adaptive systems: The swarm-fleet infrastructure scenario". In: *Information and Software Technology* 145 (2022). ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2022.106826.

[KT12] Heysem Kaya and Pinar Tüfekci. “Local and Global Learning Methods for Predicting Power of a Combined Gas & Steam Turbine”. In: *International Conference on Emerging Trends in Computer and Electronics Engineering*. 2012.

[LAM03] Bo Liu, H. A. Abbas, and B. McKay. “Classification rule discovery with ant colony optimization”. In: *IEEE/WIC International Conference on Intelligent Agent Technology, 2003. IAT 2003.* IEEE Comput. Soc, 2003. DOI: 10.1109/iat.2003.1241052.

[Lan+05] Pier Luca Lanzi, Daniele Loiacono, Stewart W. Wilson, and David E. Goldberg. “Extending XCSF beyond linear approximation”. In: *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*. GECCO ’05. Washington DC, USA: Association for Computing Machinery, 2005, pp. 1827–1834. ISBN: 1595930108. DOI: 10.1145/1068009.1068319.

[Lan+06] Pier Luca Lanzi, Daniele Loiacono, Stewart W. Wilson, and David E. Goldberg. “Prediction Update Algorithms for XCSF: RLS, Kalman Filter, and Gain Adaptation”. In: *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*. GECCO ’06. Seattle, Washington, USA: Association for Computing Machinery, 2006, pp. 1505–1512. ISBN: 1595931864. DOI: 10.1145/1143997.1144243.

[Lan01] Pier Luca Lanzi. “Mining Interesting Knowledge from Data with the XCS Classifier System”. In: *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*. GECCO’01. San Francisco, California: Morgan Kaufmann Publishers Inc., 2001, pp. 958–965. ISBN: 1-55860-774-9.

[LBL16] Himabindu Lakkaraju, Stephen H. Bach, and Jure Leskovec. “Interpretable Decision Sets”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, Aug. 2016. DOI: 10.1145/2939672.2939874.

[LBX19] Yi Liu, Will N. Browne, and Bing Xue. “Absumption to Complement Subsumption in Learning Classifier Systems”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO ’19. Prague, Czech Republic: Association for Computing Machinery, 2019, pp. 410–418. ISBN: 9781450361118. DOI: 10.1145/3321707.3321719.

[LBX20] Yi Liu, Will N. Browne, and Bing Xue. “Absumption and Subsumption Based Learning Classifier Systems”. In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. GECCO ’20. Cancún, Mexico: Association for Computing Machinery, 2020, pp. 368–376. ISBN: 9781450371285. DOI: 10.1145/3377930.3389813.

[LBX21a] Yi Liu, Will N. Browne, and Bing Xue. “A Comparison of Learning Classifier Systems’ Rule Compaction Algorithms for Knowledge Visualization”. In: *ACM Transactions on Evolutionary Learning and Optimization* 1.3 (2021), 10:1–10:38. ISSN: 2688-299X. DOI: 10/gn8gjt.

[LBX21b] Yi Liu, Will N. Browne, and Bing Xue. “Visualizations for rule-based machine learning”. In: *Natural Computing* (2021). ISSN: 1572-9796. DOI: 10.1007/s11047-020-09840-0.

[Leh12] Joel Lehman. “Evolution Through the Search for Novelty”. PhD thesis. University of Central Florida, 2012.

[LL06] Pier Luca Lanzi and Daniele Loiacono. “XCSF with Neural Prediction”. In: *2006 IEEE International Conference on Evolutionary Computation*. 2006, pp. 2270–2276. DOI: 10.1109/CEC.2006.1688588.

[LM07] Huan Liu and Hiroshi Motoda. *Computational Methods of Feature Selection (Chapman & Hall/Crc Data Mining and Knowledge Discovery Series)*. Chapman & Hall/CRC, 2007. ISBN: 978-1-58488-878-9.

[LPK21] Pantelis Linardatos, Vasilis Papastefanopoulos, and Sotiris Kotsiantis. “Explainable AI: A Review of Machine Learning Interpretability Methods”. In: *Entropy* 23.1 (2021). ISSN: 1099-4300. DOI: 10.3390/e23010018.

[LS10] Joel Lehman and Kenneth O. Stanley. “Revising the Evolutionary Computation Abstraction: Minimal Criteria Novelty Search”. In: *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*. GECCO ’10. Portland, Oregon, USA: Association for Computing Machinery, 2010, pp. 103–110. ISBN: 9781450300728. DOI: 10.1145/1830483.1830503.

[Lug+19] Edwin Lughofer et al. “Autonomous Supervision and Optimization of Product Quality in a Multi-stage Manufacturing Process based on Self-adaptive Prediction Models”. In: *Journal of Process Control* 76 (Apr. 2019), pp. 27–45. DOI: 10.1016/j.jprocont.2019.02.005.

[Luk13] Sean Luke. *Essentials of metaheuristics: a set of undergraduate lecture notes*. Lulu Com, 2013. ISBN: 9781300549628.

[Mar+17] Andreas Margraf, Anthony Stein, Leonhard Engstler, Steffen Geinitz, and Jorg Hahner. “An Evolutionary Learning Approach to Self-configuring Image Pipelines in the Context of Carbon Fiber Fault Detection”. In: *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*. 2017, pp. 147–154. DOI: 10.1109/ICMLA.2017.0-165.

[Mar+23] Andreas Margraf, Henning Cui, Simon Heimbach, Jörg Hähner, Steffen Geinitz, and Stephan Rudolph. “Model-driven optimisation of monitoring system configurations for batch production”. In: *Proceedings of the 11th International Conference on Model-Based Software and Systems Engineering MODELSWARD 2023, February 19-21, 2023, in Lisbon, Portugal*. 2023, pp. 176–183. DOI: 10.5220/0011688900003402.

[MC15] Jean-Baptiste Mouret and Jeff Clune. “Illuminating search spaces by mapping elites”. In: *CoRR* (2015). Available on arXiv: https://arxiv.org/abs/1504.04909.

[Mir+20] Thiago Zafalon Miranda, Diorge Brognara Sardinha, Márcio Porto Basgalupp, Yaochu Jin, and Ricardo Cerri. *Generation of Consistent Sets of Multi-Label Classification Rules with a Multi-Objective Evolutionary Algorithm*. https://arXiv.org/abs/2003.12526. Mar. 2020.

[Mir19] Seyedali Mirjalili. “Genetic Algorithm”. In: *Evolutionary Algorithms and Neural Networks: Theory and Applications*. Ed. by Seyedali Mirjalili. Studies in Computational Intelligence. Cham: Springer International Publishing, 2019, pp. 43–55. ISBN: 978-3-319-93025-1. DOI: 10.1007/978-3-319-93025-1_4.

[ML21] Vincent Margot and George Luta. “A New Method to Compare the Interpretability of Rule-Based Algorithms”. In: *AI* 2.4 (Nov. 2021), pp. 621–635. DOI: 10.3390/ai2040037.

[MM12] Bart Minnaert and David Martens. “Towards a Particle Swarm Optimization-Based Regression Rule Miner”. In: *2012 IEEE 12th International Conference on Data Mining Workshops*. IEEE, Dec. 2012. DOI: 10.1109/icdmw.2012.44.

[MML14] Seyedali Mirjalili, Seyed Mohammad Mirjalili, and Andrew Lewis. “Grey Wolf Optimizer”. In: *Advances in Engineering Software* 69 (Mar. 2014), pp. 46–61. ISSN: 0965-9978. DOI: 10/gfxvkw.

[Moh+08] Hamid Mohamadi, Jafar Habibi, Mohammad Saniee Abadeh, and Hamid Saadi. “Data mining with a simulated annealing based fuzzy classification system”. In: *Pattern Recognition* 41.5 (May 2008), pp. 1824–1833. DOI: 10.1016/j.patcog.2007.11.002.

[MSC19] Thiago Zafalon Miranda, Diorge Brognara Sardinha, and Ricardo Cerri. *Preventing the Generation of Inconsistent Sets of Classification Rules*. https://arXiv.org/abs/1908.09652. Aug. 2019.

[MSU11] Christian Müller-Schloer, Hartmut Schmeck, and Theo Ungerer. *Organic Computing—A Paradigm Shift for Complex Systems*. Birkhäuser Basel, 2011. DOI: 10.1007/978-3-0348-0130-0.

[MT17] Christian Müller-Schloer and Sven Tomforde. *Organic Computing—Technical Systems for Survival in the Real World*. Cham: Springer International Publishing, 2017. ISBN: 978-3-319-68477-2. DOI: 10.1007/978-3-319-68477-2.

[Myl+22] Nikolaos Mylonas, Ioannis Mollas, Nick Bassiliades, and Grigorios Tsoumakas. “Local Multi-Label Explanations for Random Forest”. In: (July 2022).

[Nak+17] Masaya Nakata, Will N. Browne, Tomoki Hamagami, and Keiki Takadama. “Theoretical XCS Parameter Settings of Learning Accurate Classifiers”. In: *Proceedings of the Genetic and Evolutionary Computation Conference 2017. Berlin, Germany, July 15–19, 2017*. Ed. by Peter A. N. Bosman. GECCO ’17. New York, NY, USA: ACM, 2017, pp. 473–480.

[NB16] Syed S. Naqvi and Will N. Browne. “Adapting learning classifier systems to symbolic regression”. In: *2016 IEEE Congress on Evolutionary Computation (CEC)*. July 2016, pp. 2209–2216. DOI: 10.1109/CEC.2016.7744061.

[NBZ20] Trung B. Nguyen, Will N. Browne, and Mengjie Zhang. “Relatedness measures to aid the transfer of building blocks among multiple tasks”. In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference.* GECCO ’20. Cancún, Mexico: Association for Computing Machinery, 2020, pp. 377–385. ISBN: 9781450371285. DOI: 10.1145/3377930.3390169.

[ND20] Bijaya Kumar Nanda and Satchidananda Dehuri. “Ant Miner: A Hybrid Pittsburgh Style Classification Rule Mining Algorithm”. In: *International Journal of Artificial Intelligence and Machine Learning* 10.1 (Jan. 2020), pp. 45–59. DOI: 10.4018/ijaiml.2020010104.

[OFJ08] Fernando E. B. Otero, Alex A. Freitas, and Colin G. Johnson. “cAnt-Miner: An Ant Colony Classification Algorithm to Cope with Continuous Attributes”. In: *Ant Colony Optimization and Swarm Intelligence.* Springer Berlin Heidelberg, 2008, pp. 48–59. DOI: 10.1007/978-3-540-87527-7_5.

[OFJ13] Fernando E. B. Otero, Alex A. Freitas, and Colin G. Johnson. “A New Sequential Covering Strategy for Inducing Classification Rules With Ant Colony Algorithms”. In: *IEEE Transactions on Evolutionary Computation* 17.1 (Feb. 2013), pp. 64–76. DOI: 10.1109/tevc.2012.2185846.

[Orh+20] Romain Orhand, Anne Jeannin-Girardon, Pierre Parrend, and Pierre Collet. “PEPACS: Integrating Probability-Enhanced Predictions to ACS2”. In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion.* GECCO ’20. Cancún, Mexico: Association for Computing Machinery, 2020, pp. 1774–1781. ISBN: 9781450371278. DOI: 10.1145/3377929.3398121.

[Ped+11] Fabian Pedregosa et al. “Scikit-Learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830. ISSN: 1532-4435.

[Per+16] Eike Permin et al. “Self-optimizing Production Systems”. In: *Procedia CIRP* 41 (Dec. 2016), pp. 417–422. DOI: 10.1016/j.procir.2015.12.114.

[PHH23] David Pätzel, Michael Heider, and Jörg Hähner. “Towards Principled Synthetic Benchmarks for Explainable Rule Set Learning Algorithms”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO ’23 Companion).* ACM, 2023. DOI: 10.1145/3583133.3596416.

[PHW21a] David Pätzel, Michael Heider, and Alexander R. M. Wagner. “An Overview of LCS Research from 2020 to 2021”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion.* GECCO ’21. Lille, France: Association for Computing Machinery, 2021, pp. 1648–1656. ISBN: 9781450383516. DOI: 10.1145/3449726.3463173.

[PHW21b] David Pätzel, Michael Heider, and Alexander R. M. Wagner. “An overview of LCS research from 2020 to 2021”. In: *GECCO ’21: Genetic and Evolutionary Computation Conference, Companion Volume, Lille, France, July 10-14, 2021.* Ed. by Krzysztof Krawiec. ACM, 2021, pp. 1648–1656. DOI: 10.1145/3449726.3463173.

[PKB07] Riccardo Poli, James Kennedy, and Tim Blackwell. “Particle Swarm Optimization”. In: *Swarm Intelligence* 1.1 (June 2007), pp. 33–57. ISSN: 1935-3820. DOI: 10/dhnq29.

[PLF02] Rafael S. Parpinelli, Heitor S. Lopes, and Alex A. Freitas. “An Ant Colony Algorithm for Classification Rule Discovery”. In: *Data Mining*. IGI Global, 2002, pp. 191–208. DOI: 10.4018/978-1-930708-25-9.ch010.

[PNH24] David Pätzel, Richard Nordsieck, and Jörg Hähner. “A Closer Look at Length-niching Selection and Spatial Crossover in Variable-length Evolutionary Rule Set Learning”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO ’24 Companion. Melbourne, VIC, Australia: Association for Computing Machinery, 2024, pp. 1779–1787. ISBN: 9798400704956. DOI: 10.1145/3638530.3664178.

[PP21] Richard J. Preen and David Pätzel. *XCSF*. Version 1.1.6. Dec. 2021. DOI: 10.5281/zenodo.5806708. URL: https://github.com/rpreen/xcsf.

[PSN20a] David Pätzel, Anthony Stein, and Masaya Nakata. “An Overview of LCS Research from IWLCS 2019 to 2020”. In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*. GECCO ’20. Cancún, Mexico: Association for Computing Machinery, 2020, pp. 1782–1788. ISBN: 9781450371278. DOI: 10.1145/3377929.3398105.

[PSN20b] David Pätzel, Anthony Stein, and Masaya Nakata. “An overview of LCS research from IWLCS 2019 to 2020”. In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*. GECCO ’20. Cancún, Mexico: Association for Computing Machinery, 2020, pp. 1782–1788. ISBN: 9781450371278. DOI: 10.1145/3377929.3398105.

[PŠP12] Vili Podgorelec, Matej Šprogar, and Sandi Pohorec. “Evolutionary design of decision trees”. In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 3.2 (Dec. 2012), pp. 63–82. DOI: 10.1002/widm.1079.

[Qui93] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993. ISBN: 1558602380.

[RBK12] Grzegorz Rozenberg, Thomas Bäck, and Joost N. Kok, eds. *Handbook of Natural Computing*. Springer Berlin Heidelberg, 2012. DOI: 10.1007/978-3-540-92910-9.

[RSG16] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “”Why Should I Trust You?”: Explaining the Predictions of Any Classifier”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 1135–1144. ISBN: 9781450342322. DOI: 10.1145/2939672.2939778.

[San+19] Clodomir J. Santana, Mariana Macedo, Hugo Siqueira, Anu Gokhale, and Carmelo J. A. Bastos-Filho. “A Novel Binary Artificial Bee Colony Algorithm”. In: *Future Generation Computer Systems* 98 (Sept. 2019), pp. 180–196. ISSN: 0167-739X. DOI: 10/gnxb6q.

[SB03] Christopher Stone and Larry Bull. “For Real! XCS with Continuous-Valued Inputs”. In: *Evolutionary Computation* 11.3 (Sept. 2003), pp. 299–336.

[SB05] Christopher Stone and Larry Bull. “An Analysis of Continuous-Valued Representations for Learning Classifier Systems”. In: *Foundations of Learning Classifier Systems*. Ed. by Larry Bull and Tim Kovacs. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 127–175.

[SB10] Patrick O. Stalph and Martin V. Butz. “Current XCSF Capabilities and Challenges”. In: *Learning Classifier Systems*. Ed. by Jaume Bacardit, Will N. Browne, Jan Drugowitsch, Ester Bernadó-Mansilla, and Martin V. Butz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 57–69. ISBN: 978-3-642-17508-4.

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction. Second Edition*. Cambridge, MA: MIT Press, 2018. ISBN: 9780262039246.

[SB19] Philipp Schmidt and Felix Biessmann. *Quantifying Interpretability and Trust in Machine Learning Systems*. Available on arXiv: https://arxiv.org/abs/1901.08558. 2019.

[SBG20] Abubakar Siddique, Will N. Browne, and Gina M. Grimshaw. “Learning classifier systems: appreciating the lateralized approach”. In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*. GECCO ’20. Cancún, Mexico: Association for Computing Machinery, 2020, pp. 1807–1815. ISBN: 9781450371278. DOI: 10.1145/3377929.3398101.

[Sch+20] Gerrit Schoettler, Ashvin Nair, Jianlan Luo, Shikhar Bahl, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. “Deep Reinforcement Learning for Industrial Insertion Tasks with Visual Inputs and Natural Rewards”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020, pp. 5548–5555. DOI: 10.1109/IROS45743.2020.9341714.

[SFX04] Jun Sun, Bin Feng, and Wenbo Xu. “Particle Swarm Optimization with Particles Having Quantum Behavior”. In: *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753)*. Vol. 1. June 2004, 325–331 Vol.1. DOI: 10/b6dd5w.

[SH98] Thomas Stützle and Holger Hoos. “Improvements on the Ant-System: Introducing the MAX-MIN Ant System”. In: *Artificial Neural Nets and Genetic Algorithms*. Ed. by George D. Smith, Nigel C. Steele, and Rudolf F. Albrecht. Vienna: Springer, 1998, pp. 245–249. ISBN: 978-3-7091-6492-1. DOI: 10/dqzfsm.

[SHH22] Helena Stegherr, Michael Heider, and Jörg Hähner. “Classifying metaheuristics: towards a unified multi-level classification system”. In: *Natural Computing* 21 (2022), pp. 155–171. DOI: 10.1007/s11047-020-09824-0.

[SHH23] Helena Stegherr, Michael Heider, and Jörg Hähner. “Assisting convergence behaviour characterisation with unsupervised clustering”. In: *Proceedings of the 15th International Joint Conference on Computational Intelligence, November 13-15, 2023, in Rome, Italy*. Ed. by Niki van Stein, Francesco Marcelloni, H. K. Lam, Marie Cottrell, and Joaquim Filipe. 2023, pp. 108–118. DOI: 10.5220/0012202100003595.

[Sid+24] Abubakar Siddique, Michael Heider, Muhammad Iqbal, and Hiroki Shiraishi. “A Survey on Learning Classifier Systems from 2022 to 2024”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion.* GECCO ’24 Companion. Melbourne, VIC, Australia: Association for Computing Machinery, 2024, pp. 1797–1806. ISBN: 9798400704956. DOI: 10.1145/3638530.3664165.

[SJK04] P. S. Shelokar, V. K. Jayaraman, and B. D. Kulkarni. “An ant colony classifier system: application to some process engineering problems”. In: *Computers & Chemical Engineering* 28.9 (Aug. 2004), pp. 1577–1584. DOI: 10.1016/j.compchemeng.2003.12.004.

[SK11] Rizauddin Saian and Ku Ruhana Ku-Mahamud. “Hybrid Ant Colony Optimization and Simulated Annealing for Rule Induction”. In: *2011 UKSim 5th European Symposium on Computer Modeling and Simulation.* IEEE, Nov. 2011. DOI: 10.1109/ems.2011.17.

[SL19] Na Sun and Yong Lu. “A self-adaptive genetic algorithm with improved mutation mode based on measurement of population diversity”. In: *Neural Computing and Applications* 31.5 (2019), pp. 1435–1443. ISSN: 1433-3058. DOI: 10.1007/s00521-018-3438-9.

[SMH17] Anthony Stein, Roland Maier, and Jörg Hähner. “Toward curious learning classifier systems: combining XCS with active learning concepts”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion - GECCO ’17, Berlin, Germany — July 15 - 19, 2017.* 2017. ISBN: 9781450349390. DOI: 10.1145/3067695.3082488.

[SMH18] Anthony Stein, Simon Menssen, and Jörg Hähner. “What about Interpolation? A Radial Basis Function Approach to Classifier Prediction Modeling in XCSF”. In: *Proceedings of the Genetic and Evolutionary Computation Conference.* GECCO ’18. Kyoto, Japan: Association for Computing Machinery, 2018, pp. 537–544. ISBN: 9781450356183. DOI: 10.1145/3205455.3205599.

[SSC12] Bikash Kanti Sarkar, Shib Sankar Sana, and Kripasindhu Chaudhuri. “A genetic algorithm-based rule extraction system”. In: *Applied Soft Computing* 12.1 (Jan. 2012), pp. 238–254. DOI: 10.1016/j.asoc.2011.08.049.

[SSN03] Tiago Sousa, Arlindo Silva, and Ana Neves. “A Particle Swarm Data Miner”. In: *Progress in Artificial Intelligence.* Springer Berlin Heidelberg, 2003, pp. 43–53. DOI: 10.1007/978-3-540-24580-3_12.

[SSN04] Tiago Sousa, Arlindo Silva, and Ana Neves. “Particle Swarm based Data Mining Algorithms for classification tasks”. In: *Parallel Computing* 30.5-6 (May 2004), pp. 767–783. DOI: https://doi.org/10.1016/j.parco.2003.12.015.

[ST11] Magdalena Smetek and Bogdan Trawiński. “Investigation of Genetic Algorithms with Self-Adaptive Crossover, Mutation, and Selection”. In: *Hybrid Artificial Intelligent Systems.* Ed. by Emilio Corchado, Marek Kurzyński, and Michał Woźniak. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 116–123. ISBN: 978-3-642-21219-2. DOI: 10.1007/978-3-642-21219-2_16.

[ST22] Connor Schönberner and Sven Tomforde. “Deep Reinforcement Learning with a Classifier System—First Steps”. In: *Architecture of Computing Systems.* Ed. by Martin Schulz, Carsten Trinitis, Nikela Papadopoulou, and Thilo Pionteck. Cham: Springer International Publishing, 2022, pp. 256–270. ISBN: 978-3-031-21867-5.

[Ste+20] Anthony Stein, Roland Maier, Lukas Rosenbauer, and Jörg Hähner. “XCS Classifier System with Experience Replay”. In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference.* GECCO ’20. Cancún, Mexico: Association for Computing Machinery, 2020, pp. 404–413. ISBN: 9781450371285. DOI: 10.1145/3377930.3390249.

[Ste+21a] Helena Stegherr, Michael Heider, Leopold Luley, and Jörg Hähner. “Design of large-scale metaheuristic component studies”. In: *GECCO ’21: Genetic and Evolutionary Computation Conference, Companion Volume, Lille, France, July 10-14, 2021.* Ed. by Krzysztof Krawiec. ACM, 2021, pp. 1217–1226. DOI: 10.1145/3449726.3463168.

[Ste+21b] Anthony Stein, Sven Tomforde, Jean Botev, and Peter R Lewis. “Lifelike computing systems”. In: *Proceedings of the Lifelike Computing Systems Workshop (LIFELIKE 2021) at the 19th International Conference on Artificial Life (ALIFE 2021).* Available at: http://ceur-ws.org/Vol-3007. Prague, Czech Republic, 2021.

[Ste+23] Helena Stegherr, Leopold Luley, Jonathan Wurth, Michael Heider, and Jörg Hähner. *A framework for modular construction and evaluation of metaheuristics.* Tech. rep. 2023-01. Fakultät für Angewandte Informatik, 2023.

[Ste19] Anthony Stein. “Interpolation-Assisted Evolutionary Rule-Based Machine Learning - Strategies to Counter Knowledge Gaps in XCS-Based Self-Learning Adaptive Systems”. Doctoral Thesis. Universität Augsburg, 2019.

[STH16] Matthias Sommer, Sven Tomforde, and Jörg Hähner. “An Organic Computing Approach to Resilient Traffic Management”. In: *Autonomic Road Transport Support Systems.* Ed. by Thomas Leo McCluskey, Apostolos Kotsialos, Jörg P. Müller, Franziska Klügl, Omer F. Rana, and René Schumann. 2016. ISBN: 9783319258065. DOI: 10.1007/978-3-319-25808-9s_7.

[SYZ10] Haijun Su, Yupu Yang, and Liang Zhao. “Classification rule discovery with DE/QDE algorithm”. In: *Expert Systems with Applications* 37.2 (Mar. 2010), pp. 1216–1222. DOI: https://doi.org/10.1016/j.eswa.2009.06.029.

[Tan+03] K. C. Tan, Q. Yu, C. M. Heng, and T. H. Lee. “Evolutionary computing for knowledge discovery in medical diagnosis”. In: *Artificial Intelligence in Medicine* 27.2 (Feb. 2003), pp. 129–154. DOI: 10.1016/s0933-3657(03)00002-2.

[TBP07] Kreangsak Tamee, Larry Bull, and Ouen Pinngern. “Towards Clustering with XCS”. In: *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation.* GECCO ’07. London, England: Association for Computing Machinery, 2007, pp. 1854–1860. ISBN: 9781595936974. DOI: 10.1145/1276958.1277326.

[TF10] Ajay Kumar Tanwani and Muddassar Farooq. “Classification Potential vs. Classification Accuracy: A Comprehensive Study of Evolutionary Algorithms with Biomedical Datasets”. In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2010, pp. 127–144. DOI: 10.1007/978-3-642-17508-4_9.

[Tim+08] J. Timmis, P. Andrews, N. Owens, and E. Clark. “An interdisciplinary perspective on artificial immune systems”. In: *Evolutionary Intelligence* 1.1 (Jan. 2008), pp. 5–26. DOI: 10.1007/s12065-007-0004-2.

[TMU13] Jie Tan, Jason Moore, and Ryan Urbanowicz. “Rapid Rule Compaction Strategies for Global Knowledge Discovery in a Supervised Learning Classifier System”. In: *ECAL 2013: The Twelfth European Conference on Artificial Life*. MIT Press, 2013, pp. 110–117. DOI: 10.7551/978-0-262-31709-2-ch017.

[Tra+07] Hau Trung Tran, Cédric Sanza, Yves Duthen, and Thuc Dinh Nguyen. “XCSF with computed continuous action”. In: *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*. GECCO ’07. London, England: Association for Computing Machinery, 2007, pp. 1861–1869. ISBN: 9781595936974. DOI: 10.1145/1276958.1277327.

[Tsa+09] Athanasios Tsanas, Max Little, Patrick McSharry, and Lorraine Ramig. “Accurate telemonitoring of Parkinson’s disease progression by non-invasive speech tests”. In: *Nature Precedings* (2009).

[Tüf14] Pınar Tüfekci. “Prediction of Full Load Electrical Power Output of a Base Load Operated Combined Cycle Power Plant Using Machine Learning Methods”. In: *International Journal of Electrical Power & Energy Systems* 60 (2014), pp. 126–140. ISSN: 0142-0615. DOI: 10/gn9s2h.

[TX12] Athanasios Tsanas and Angeliki Xifara. “Accurate Quantitative Estimation of Energy Performance of Residential Buildings Using Statistical Machine Learning Tools”. In: *Energy and Buildings* 49 (2012), pp. 560–567. ISSN: 0378-7788. DOI: 10/gg5vzx.

[UB17] Ryan J. Urbanowicz and Will N. Browne. *Introduction to Learning Classifier Systems*. 1st. Springer Publishing Company, Incorporated, 2017. ISBN: 978-3-662-55006-9.

[UGM12] Ryan J. Urbanowicz, Ambrose Granizo-Mackenzie, and Jason H. Moore. “An Analysis Pipeline with Statistical and Visualization-guided Knowledge Discovery for Michigan-style Learning Classifier Systems”. In: *IEEE Computational Intelligence Magazine* 7.4 (2012), pp. 35–45. DOI: 10.1109/MCI.2012.2215124.

[UM09] Ryan J. Urbanowicz and Jason H. Moore. “Learning Classifier Systems: A Complete Introduction, Review, and Roadmap”. In: *Journal of Artificial Evolution and Applications* (2009).

[UM15] Ryan J. Urbanowicz and Jason H. Moore. “ExSTraCS 2.0: description and evaluation of a scalable learning classifier system”. In: *Evolutionary Intelligence* 8.2 (Sept. 2015), pp. 89–116. ISSN: 1864-5917. DOI: 10.1007/s12065-015-0128-8.

[Uno10] Olgierd Unold. “Self-adaptive Learning Classifier System”. In: *Journal of Circuits, Systems, and Computers* 19.01 (2010), pp. 275–296.

[van+97] A. H. C. van Kampen, Z. Ramadan, M. Mulholland, D. B. Hibbert, and L. M. C. Buydens. “Learning classification rules from an ion chromatography database using a genetic based classifier system”. In: *Analytica Chimica Acta* 344.1-2 (May 1997), pp. 1–15. DOI: 10.1016/s0003-2670(96)00629-0.

[Vas+01] J.A. Vasconcelos, J.A. Ramirez, R.H.C. Takahashi, and R.R. Saldanha. “Improvements in genetic algorithms”. In: *IEEE Transactions on Magnetics* 37.5 (2001), pp. 3414–3417. DOI: 10.1109/20.952626.

[Ver+22] Diederick Vermetten, Hao Wang, Manuel López-Ibañez, Carola Doerr, and Thomas Bäck. “Analyzing the Impact of Undersampling on the Benchmarking and Configuration of Evolutionary Algorithms”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO ’22. Boston, Massachusetts: Association for Computing Machinery, 2022, pp. 867–875. ISBN: 9781450392372. DOI: 10.1145/3512290.3528799.

[Vir+21] Marco Virgolin, Andrea De Lorenzo, Francesca Randone, Eric Medvet, and Mattias Wahde. “Model learning with personalized interpretability estimation (ML-PIE)”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO ’21. Lille, France: Association for Computing Machinery, 2021, pp. 1355–1364. ISBN: 9781450383516. DOI: 10.1145/3449726.3463166.

[VL21] Giulia Vilone and Luca Longo. “Notions of explainability and evaluation approaches for explainable artificial intelligence”. In: *Information Fusion* 76 (Dec. 2021), pp. 89–106. DOI: 10.1016/j.inffus.2021.05.009.

[Wan+16] Youchuan Wan, Mingwei Wang, Zhiwei Ye, and Xudong Lai. “A Feature Selection Method Based on Modified Binary Coded Ant Colony Optimization Algorithm”. In: *Applied Soft Computing* 49 (Dec. 2016), pp. 248–258. ISSN: 1568-4946. DOI: 10/f9knrr.

[Wei09] Thomas Weise. *Global optimization algorithms-theory and application*. Self-Published Thomas Weise, 2009.

[Wil00] Stewart W. Wilson. “Get Real! XCS with Continuous-Valued Inputs”. In: *Learning Classifier Systems*. Ed. by Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 209–219. ISBN: 978-3-540-45027-6.

[Wil01] Stewart W. Wilson. “Mining Oblique Data with XCS”. In: *Advances in Learning Classifier Systems*. Ed. by Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 158–174. ISBN: 978-3-540-44640-8.

[Wil02a] Stewart W. Wilson. “Classifiers that approximate functions”. In: *Natural Computing* 1.2/3 (2002), pp. 211–234. DOI: 10.1023/a:1016535925043.

[Wil02b] Stewart W. Wilson. "Compact Rulesets from XCSI". In: *Advances in Learning Classifier Systems*. Springer Berlin Heidelberg, 2002, pp. 197–208. DOI: 10.1007/3-540-48104-4_12.

[Wil94] Stewart W. Wilson. "ZCS: A Zeroth Level Classifier System". In: *Evolutionary Computation* 2.1 (1994), pp. 1–18. DOI: 10.1162/evco.1994.2.1.1.

[Wil95] Stewart W. Wilson. "Classifier Fitness Based on Accuracy". In: *Evolutionary Computation* 3.2 (1995), pp. 149–175.

[Wil98] Stewart W. Wilson. "Generalization in the XCS Classifier System". In: *Genetic Programming 1998: Proceedings of the Third Annual Conference*. 1998, pp. 665–674.

[WM97] David H. Wolpert and William G. Macready. "No Free Lunch Theorems for Optimization". In: *IEEE Transactions on Evolutionary Computation* 1.1 (Apr. 1997), pp. 67–82. ISSN: 1941-0026. DOI: 10/d2x5nj.

[Woo+24] Alexa Woodward, Harsh Bandhey, Jason H. Moore, and Ryan J. Urbanowicz. "Survival-LCS: A Rule-Based Machine Learning Approach to Survival Analysis". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '24. Melbourne, VIC, Australia: Association for Computing Machinery, 2024, pp. 431–439. ISBN: 9798400704949. DOI: 10.1145/3638529.3654154.

[Wu+18] Qing Wu, Yuanfeng Shen, Zheping Ma, Jin Fan, and Ruiquan Ge. "iBQPSO: An Improved BQPSO Algorithm for Feature Selection". In: *2018 International Joint Conference on Neural Networks (IJCNN)*. July 2018, pp. 1–8. DOI: 10/gnxcd9.

[Wu+19] Qing Wu, Zheping Ma, Jin Fan, Gang Xu, and Yuanfeng Shen. "A Feature Selection Method Based on Hybrid Improved Binary Quantum Particle Swarm Optimization". In: *IEEE Access* 7 (2019), pp. 80588–80601. ISSN: 2169-3536. DOI: 10/gnxcfb.

[Wur+22] Jonathan Wurth, Michael Heider, Helena Stegherr, Roman Sraj, and Jörg Hähner. "Comparing different Metaheuristics for Model Selection in a Supervised Learning Classifier System". In: *Genetic and Evolutionary Computation Conference Companion (GECCO '22 Companion)*. 2022. DOI: 10.1145/3520304.3529015.

[Wur+23] Jonathan Wurth, Helena Stegherr, Michael Heider, Leopold Luley, and Jörg Hähner. "Fast, Flexible, and Fearless: A Rust Framework for the Modular Construction of Metaheuristics". In: *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*. GECCO '23 Companion. Lisbon, Portugal: Association for Computing Machinery, 2023, pp. 1900–1909. ISBN: 9798400701207. DOI: 10.1145/3583133.3596335.

[Wur+24] Jonathan Wurth, Helena Stegherr, Michael Heider, and Jörg Hähner. "GRAHF: A Hyper-Heuristic Framework for Evolving Heterogeneous Island Model Topologies". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '24. Melbourne, VIC, Australia: Association for Computing Machinery, 2024, pp. 1054–1063. ISBN: 9798400704949. DOI: 10.1145/3638529.3654136.

[XC20] Yunshuang Xiao and Shuyu Chen. "RBFACO: A New Feature Selection Algorithm". In: *2020 IEEE International Conference on Bioinformatics and Biomedicine (BIBM).* Dec. 2020, pp. 2884–2890. DOI: 10/gnfbnv.

[Yeh98] I-Cheng Yeh. "Modeling of Strength of High-Performance Concrete Using Artificial Neural Networks". In: *Cement and Concrete Research* 28.12 (1998), pp. 1797–1808. ISSN: 0008-8846. DOI: 10/dxm5c2.

[YWG12] Seniha Esen Yuksel, Joseph N. Wilson, and Paul D. Gader. "Twenty Years of Mixture of Experts". In: *IEEE Transactions on Neural Networks and Learning Systems* 23.8 (2012), pp. 1177–1193. DOI: 10.1109/TNNLS.2012.2200299.

[Zha+17] Y. Zhang, C. Qian, J. Lv, and Y. Liu. "Agent and Cyber-Physical System Based Self-Organizing and Self-Adaptive Intelligent Shopfloor". In: *IEEE Transactions on Industrial Informatics* 13.2 (2017), pp. 737–747.

[Zho+21] Jianlong Zhou, Amir H. Gandomi, Fang Chen, and Andreas Holzinger. "Evaluating the Quality of Machine Learning Explanations: A Survey on Methods and Metrics". In: *Electronics* 10.5 (Mar. 2021), p. 593. DOI: 10.3390/electronics10050593.

[Zho+24] Ryan Zhou et al. "Evolutionary Computation and Explainable AI: A Roadmap to Understandable Intelligent Systems". In: *IEEE Transactions on Evolutionary Computation* (2024). DOI: 10.1109/TEVC.2024.3476443.

[Zho12] Zhi-Hua Zhou. *Ensemble Methods: Foundations and Algorithms.* 1st. Chapman & Hall/CRC, 2012. ISBN: 978-1-4398-3003-1. DOI: 10.1201/b12207.

[ZT03] Zhi-Hua Zhou and Wei Tang. "Selective Ensemble of Decision Trees". In: *Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing.* Ed. by Guoyin Wang, Qing Liu, Yiyu Yao, and Andrzej Skowron. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2003, pp. 476–483. ISBN: 978-3-540-39205-7. DOI: 10/b7mzv4.

[ZWT02] Zhi-Hua Zhou, Jianxin Wu, and Wei Tang. "Ensembling Neural Networks: Many Could Be Better than All". In: *Artificial Intelligence* 137.1 (May 2002), pp. 239–263. ISSN: 0004-3702. DOI: 10/fnqnjq.