UNIVERSITY OF TRENTO
Department of Computer Science and Information Engineering

International ICT Doctoral School

PhD Dissertation

Coactive Learning Algorithms for Constructive Preference Elicitation

Paolo DRAGONE

Advisors

Andrea PASSERINI
University of Trento

Michele VESCOVI
TIM - Telecom Italia

April 2019

Abstract

Preference-based decision problems often involve choosing one among a large set of options, making common tasks like buying a car or a domestic appliance very challenging for a customer to handle on her own. This is especially true when buying online, where the amount of available options is humongous, and expert advice is yet limited. *Recommender systems* have become essential computational tools for aiding users in this endeavor. Recommender systems represent one of the most successful applications of artificial intelligence. In the last decades, several recommendation approaches have been proposed for different types of applications, from assisted browsing of product catalogs to personalization of results in search engines. Depending on the application, the job of the recommender system may be to recommend a satisfying option for the given context, as in finding the next best song to play, as opposed to helping the user in finding an *optimal* instance, e.g. when looking for an apartment. The former is generally handled by data-driven approaches, such as collaborative filtering and contextual bandits, while in the latter case data is usually scarce, making it necessary to employ specialized algorithms for *preference elicitation*. Preference elicitation algorithms interactively build a *utility* model of the user preferences and then recommend the instances with the highest utility. Preference elicitation is especially effective when recommending infrequently purchased items, such as professional work tools, electronic devices and other products that can be explicitly stored e.g. in the database of an e-commerce website. Standard preference elicitation algorithms, however, struggle when the options are so numerous that cannot even be explicitly enumerated, and instead need to be represented implicitly as a collection of variables and constraints. Indeed, when a customer wants to configure a product by putting several components together, e.g. for a custom personal computer, the option space is combinatorial and grows exponentially with the number of components, making it impractical to store every single feasible combination explicitly. This is an example of *constructive* decision problem, in which an object has to be synthesized on the basis of the preferences of the customer and the constraints over the configuration domain. Constructive problems such as product configuration have traditionally been addressed by specialized configurator systems, which guide the user through the configuration process component by component and check whether the user choices are consistent with the set of feasibility constraints. Over the years, however, the limitations of product configurators for mass customization have become apparent. With the growing scale of configuration problems, product configurators have become more difficult for non-experts to use and ultimately do not provide relief against the "mass confusion" caused by the sheer amount of choice. Research in this field has progressively been integrating recommendation technologies into configuration systems, in order to make them more flexible and easy to use. Preference elicitation in product configuration has been attempted as well but still remains a challenge.

We propose a generic framework for preference elicitation in constructive domains, that is able to scale to large combinatorial problems better than existing techniques. Our constructive preference elicitation framework is based on *online structured prediction*, a machine learning technique that deals with sequential decision problems over structured objects. By combining online structured prediction and state-of-the-art constraint solvers we can efficiently learn user utility models and make increasingly better recommendations for complex preference-based constructive problems such as product configuration. In particular, we favor the use of *coactive learning*, an online structured prediction framework for preference learning. Coactive learning is particularly well suited for constructive preference elicitation as it may be seen as a cooperation between the user and the system. The user and the systems interact through "coactive" feedback: after each recommendation, the user provides a modification that makes it slightly better for her preferences. This type of feedback is very flexible and can be acquired both explicitly and implicitly from the user actions. Coactive learning also comes with theoretical convergence guarantees and a set of ready-made extensions for many related problems such as learning in a multi-user setting and learning with approximate constraint solvers.

In this thesis we detail our coactive learning approach to constructive preference elicitation, and propose extensions for scaling up to very large constructive problems and personalizing the utility model. We also applied our framework to two important classes of constructive preference elicitation problems, namely layout synthesis and product bundling. The former is a design process for arranging objects into a given space, while the latter is a kind of product configuration problem in which the object to configure is a package of different products and services. Within the product bundling application, we also performed an extensive validation involving real participants, which highlights the practical benefits of our approach.

Acknowledgements

My deepest gratitude goes to my advisor Andrea Passerini, who tirelessly supported me throughout my whole PhD and single-handedly made me the researcher that I am today. I feel extremely privileged to have had Andrea as my advisor and I will always be indebted to him for his insightful guidance and dedication. The time spent working with Andrea has been invaluable and has helped me grow both as a researcher and as a person.

I also want to thank my colleagues at the TIM SKIL group, and in particular Michele Vescovi, who co-supervised my PhD work and provided important advice about the possible industrial applications of my thesis work. This thesis would not have been possible without the financial support of TIM.

A special thanks goes to my friend and colleague Stefano Teso. Stefano and I co-authored most of the publications that make up this thesis. I can honestly say that I would not have achieved half of what I did without his help. It has only been after meeting Stefano that I truly started to grasp what being a researcher really means.

These years in Trento have been full of intense work and memorable events. I cannot overstate how happy I feel about sharing this experience with the wonderful people of the SML research group: Gianni Pellegrini, Luca Erculiani and Paolo Morettin. Thanks for the many interesting discussions and all the cheerful moments of daily PhD life. I have no words to express my gratitude to them and to all my fellow PhD students at the DISI, who made my time in Trento the very best of my life.

I also wish to thank Paolo Viappiani and Kristian Kersting for hosting me for some time in Paris and Darmstadt respectively. The time with them allowed me to deepen my knowledge and broaden my research interests.

Last but definitely not least, a grateful thought goes to my parents and my family, for their constant trust and support, and especially to my nephews Giorgia, Giulia and Francesco, for whom I hope this thesis may be an inspiration to always pursue their passions in life.

Paolo Dragone

Trento, 16th April 2019

Contents

Chapter 1

Introduction

Decision making is a central aspect of the human experience. We are constantly faced with decision problems about ourselves and our state of affairs, from which brand of cereal to buy at the supermarket, to whether to vote for one party or the others in the general elections. Indeed, some decisions have longer-term consequences and need to be made considering several criteria, which makes them more significant yet more difficult. While in the everyday life an individual usually does not employ analytical tools to decide which brand of cereal to buy, she might want to do so for an expensive item such as a car or an apartment. In these cases, a *decision support system* may be of great help in solving complex decision problems.

In some situations, the goal (or *objective*) of the decision maker can be precisely stated, such as profit maximization or cost minimization, and thus the decision problem can be solved via analytical methods from operations research and game theory [187, 281]. This is usually the case for e.g. companies deciding upon their economic strategy. In other cases, when an individual wishes to buy a product for instance, the decision problem is influenced by the *preferences* of the decision maker. To make a meaningful choice and to be able to solve analytically this type of decision problems, the objective has to be clearly stated as a *preference model*, which can then be algorithmically optimized to find the best decision to make. According to the typical view in microeconomics and decision theory, when faced with a preference-based decision problem, the goal of the decision maker is to maximize her *utility* [107, 186]. Loosely speaking, the notion of utility relates to the degree of desire of an individual towards a product, and represents the trade-off between conflicting objectives, such as quality and price. In decision theory, the utility is a function defined over a set of products, ranking them according to the preferences of the decision maker. The best decision amongst a set of choices is the one determined by maximizing the utility function. While not being universally accepted as a fitting model of the human rationality in the decision-making process, utility maximization is the principle that a decision support system uses to aid a user in making a good decision when faced with a complex decision problem. The utility model, however, is not known a priori, so the decision support system needs to estimate it using specialized *preference elicitation* algorithms [150, 206]. Preference elicitation is an interactive process with which a preference model is progressively built as more preference information is acquired from the user, enabling the system to provide recommendations of increasingly higher quality.

In this thesis, we are interested in preference-based decision problems, and the algorithmic processes to solve them, with the aim of developing efficient and reliable decision support systems. In particular, we will focus on a certain type of decision problems, for which an object does not just have to be chosen among a set of available options, but has instead to be "built" from scratch, assembling its components while maximizing the utility for the user and satisfying a set of feasibility constraints. We call these *constructive* decision problems and the process for solving them *constructive preference elicitation* [85]. Constructive problems arise e.g. in *product configuration* [100], in which a customer wishes to customize a certain type of product by setting up its components one by one. This category also includes preference-based design tasks, such as shaping 3D printed objects, and other combinatorial tasks with unknown objectives to be estimated from the interaction with a user.

1.1 Motivation

With the growing availability of mass produced products, evaluating more and more choices has become increasingly harder for human decision makers. This phenomenon is often referred to as the *paradox of choice* or *product variety paradox*. While theoretically more choice should lead to products of higher quality and higher customer satisfaction, in practice this is not the case due to the greater effort the customer has to put into making a choice over the vast variety of available goods [74, 239]. This problem is exacerbated in the context of *mass customization*, in which products can technically be highly customized through specialized *configuration systems*, yet users are often paralyzed by the amount of available choices, leading to so called *mass confusion* [139]. The development of decision support systems capable of aiding users in making good choices in very complex scenarios is therefore becoming ever more important.

Despite having been theorized decades ago [155, 207], mass customization has yet to be fully realized in practice. Much progress has been made on the industrial side, as well as on the product configuration technology between the customers and the production process. However, less attention has been given to the development of effective user interfaces, which should help users throughout the decision process by providing recommendations and explanations, in order to mitigate the negative effects of the product variety paradox [267]. This lack of awareness in regard to the user needs explains the lag in adoption of mass customization technology.

In the past decade, product configuration systems have seen an increasing blend with recommendation technologies [94, 264]. *Constraint-based recommenders*, in particular, can be used to assemble and recommend configurable products [97, 287]. Typically, these systems also allow the user to interact and state preferences through preference elicitation or *example critiquing*, i.e. stating what should the system do to improve the currently recommended configurations [29, 211, 275]. Critiquing systems based on *soft constraints* [211], in particular, permit the user to state expressive trade-offs over the products attributes.

Most constraint-based critiquing systems, however, do not employ a rigorous utility maximization paradigm, which is necessary to guarantee the eventual convergence to an optimal solution. Also, many existing systems require the direct manipulation of the trade-off objectives, which is impractical for more than a few preference criteria. State-of-the-art preference elicitation techniques, including *regret-based* and *Bayesian* preference elicitation [272, 274], could be integrated into constraint-based configuration systems, to provide a sound way to collect preference information and guarantee the user to reach an optimal solution. These techniques, however, come with their own set of shortcomings, mostly regarding noise resilience and scalability [86].

All the above issues, in one way or another, obstruct the wide spread adoption of mass customization, and have prevented the implementation of fully functional decision support systems to help users in making optimal choices when dealing with complex decision problems. Notwithstanding the plentiful research in this field, not a single method capable of addressing all the above issues at once has been proposed. In this thesis, we aim at filling this gap and develop a complete, reliable and efficient methodology for preference elicitation in complex, constraint-based, combinatorial domains.

1.2 Contributions

In order to overcome the issues exposed in the previous section, a preference elicitation system should exhibit certain properties. Guo and Sanner [128] provided a thorough list of principles that modern preference elicitation techniques should abide to:

1. *Real-time interaction*

   A preference elicitation system should be reactive to the user input and provide recommendations within a few seconds or less, in order to ensure a smooth interaction.

2. *Multi-attribute domains*

   The utility model should adhere to the principles of *multi-attribute utility theory*, which establishes the concept of evaluating feasible choices along several attributes and enables efficient preference elicitation by decomposing the utility over these components.

3. *Low cognitive load*

   The system should provide an interaction method and a type of feedback that is not overwhelmingly difficult for the user to provide. In general, requiring the user to directly manipulate utility values, trade-offs weights, probabilities or any *cardinal* value has been proven to be too cognitively difficult. More suiting types of interactions include pair-wise and set-wise comparisons, or other types of *ordinal* judgment.

4. *Robustness to noise*

   Additionally to providing an interaction with low cognitive effort, a preference elicitation system needs to account for noise in the user feedback, which can be introduced by a number of external factors such as distraction and fatigue. This ensures the system to provide good recommendations despite the uncertainty in the user response.

5. *Scalability (domain size w.r.t. amount of feedback)*

   A preference elicitation algorithm should be able to find a good solution with little user feedback despite a large number of possible choices. Expanding the decision space should not drastically increase the amount of feedback required from the user.

The previous five properties are indeed required by any preference elicitation algorithm able to deal with large catalogs of products. We propose to complement the above list with seven additional properties. While these properties are beneficial for all decision support systems, even non-constructive ones, their simultaneous satisfaction is a necessary condition for the successful implementation of a constructive preference elicitation system. The following is our complementary list of properties:

6. *Constrained combinatorial spaces*

   As for constraint-based recommenders, in a constructive scenario such as product configuration, the preference elicitation algorithm should be able to reason over intentionally specified option spaces, whose instances are made up as combinations of various attributes and are subject to several feasibility constraints.

7. *Hybrid domains*

   Many existing preference elicitation algorithms deal exclusively with categorical attributes. In a constructive scenario, we also require the algorithm to deal with numerical attributes in conjunction to categorical ones. This is needed to ensure full representability of constructive problems, and to allow more complex numerical trade-offs.

8. *Explicit and implicit feedback*

   In a constructive problem feedback might be scarce and hard to acquire, hence preference elicitation algorithms need to be able to adapt to any kind of feedback available, being it explicit or implicit, or even combinations of both.

9. *Contextual information*

   Preference elicitation algorithms should formally take into account contextual information. Besides being helpful for single runs, a preference model with contextual information would be able to *generalize* to similar contexts and would be reusable to similar tasks. By encoding user features as contextual information, for instance, a contextual preference model would be able to exploit previously acquired information from similar users to make good recommendations right from the beginning of the elicitation process, thereby making it easier and quicker to reach satisfactory solutions.

10. *Optimality guarantees*

    Most constraint-based systems provide heuristically "good" recommendations relying on the incremental improvements of the user. We instead require formal guarantees of convergence towards an optimal solution. State-of-the-art preference elicitation techniques do provide such guarantees, but fail to do so in a fully constructive scenario, while realizing all the above properties.

11. *Scalability (domain size w.r.t. inference time)*

    As mentioned, constructive domains are combinatorial, and as such they grow *exponentially* with the number of attributes. Allowing numerical variables can even potentially make the domain infinitely large. While research on constraint solvers has made combinatorial problems more practical, scaling up to a large number of attributes is still a challenge for a preference elicitation system, especially in relation to the requirement of real time interaction. Nevertheless, to solve constructive decision problems, a preference elicitation needs to be able to scale up to a quite large number of attributes, which is not doable with existing techniques.

12. *Expressive trade-offs*

    Preference elicitation techniques should permit the formulation of more complex trade-offs than just a weighted sum of the attribute values. Also, it is important to adapt the set of preference criteria used by the utility model with respect to the particular user needs [211]. Critiquing systems based on soft constraints provide such functionality, yet soft constraints are no longer sufficient when the choice domain includes numerical attributes. Greater expressive power is guaranteed by techniques working with *generalized additive independent* (GAI) models (see Section 2.1.2), which can be formulated over numerical attributes too, but they are usually assumed fixed and the user cannot expand the set of preference criteria as doable in critiquing recommenders. A combination of the expressiveness of GAI models with the flexibility of critiquing systems is needed in the constructive preference elicitation setting.

A *constructive recommender* system is expected to satisfy all the above desiderata. To the best of our knowledge, no preference elicitation nor recommendation technique has been proposed so to satisfy these twelve properties all together. Table 1.1 provides an overview of the properties satisfied by state-of-the-art techniques. Regret-based approaches utilize a multi-attribute representation in conjunction with a constraint solvers, but do not directly model noise in the user feedback. Bayesian approaches do model noisy feedback but do not deal well with constrained domains and are more susceptible to increasing domain size (see e.g. [86, 257]). Constraint-based recommenders are typically combined with critiquing interfaces and soft constraint solvers, which can enlarge the set of relevant preference criteria, but are limited to Boolean criteria and typically do not provide optimality guarantees. We will review in detail all the aforementioned techniques in Chapter 2.

This thesis presents a framework specifically designed for recommending *structured* objects such as configurable products, which satisfies all the above requirements. To do so, we cast the task of preference elicitation over the constrained combinatorial outcome space of a complex decision problem as a specific type of *machine learning* problem. The learning technique we employ is *online structured prediction*, which is itself a combination of two broader methodologies: *online learning* [50] and *structured-output prediction* [14]. Online learning is a paradigm for solving sequential decision problems, in which information about the prediction is revealed only after the prediction has been made. This is analogous to preference elicitation, in which the feedback of a recommendation is observed only after the recom-

	Properties	Regret-based preference elicitation	Bayesian preference elicitation	Recommenders with soft constraints	Constructive preference elicitation
1.	Real-time interaction	✓	✓	✓	✓
2.	Multi-attribute domains	✓	✓	✓	✓
3.	Low cognitive load	✓	✓		✓
4.	Robustness to noise		✓		✓
5.	Scalability (feedback)	✓	✓	✓	✓
6.	Constraints	✓		✓	✓
7.	Hybrid domains	✓	✓	✓	✓
8.	Feedback (expl. & impl.)	✓	✓		✓
9.	Contextual information			✓	✓
10.	Optimality guarantees	✓	✓		✓
11.	Scalability (inf. time)	†		†	✓
12.	Expressive trade-offs	$*$	$*$	‡	✓

Table 1.1: Overview of our desired properties among different techniques in the literature compared to constructive preference elicitation. (✓): property satisfied; (†): depends on the underlying constraint solver; ($*$): only with GAI models (see Section 2.1.2); (‡): only with Boolean criteria.

mendation has been made. Structured-output prediction is a learning method that deals with structured objects, such as sequences, trees and graphs. Structured prediction can also be used in conjunction with constraint solvers, which are the same solvers used by constraint-based recommender systems [87, 259]. By combining online learning and structured prediction, we obtain a learning method that attains the benefits of constraint-based recommenders and state-of-the-art preference elicitation techniques, ensuring robustness to noise, scalability and guaranteed optimality. Chapter 3 covers all the relevant machine learning background.

Whilst constructive preference elicitation can be achieved through any online structured prediction method, this thesis focuses on one particular technique called *coactive learning* [247]. Coactive learning is an online structured prediction framework that can be used to learn a user utility function from weak "coactive" feedback. Coactive feedback consists in receiving from the user an "improvement" to the currently recommended solution. This is particularly suiting for our purposes as it resembles the kind of interaction of a critiquing system, yet being more flexible and with the possibility of being acquired implicitly as well as explicitly from the user manipulations. Also, coactive learning comes with theoretical convergence guarantees under very general assumptions, which apply to the constructive case as well.

The constructive preference elicitation setting will be detailed in Chapter 4. Our technique provides the first ten of the above desired properties out-of-the-box. In its basic formulation, domain scalability is tied to the underlying constraint solver, which is our performance bottleneck, just like regret-based elicitation and constraint-based recommenders. Scaling up to larger constructive domains is therefore still a research problem, which we aim to address in this thesis. One possible approach is to use local search solvers or other heuristics to approximate the utility maximization procedure [82, 93]. This comes at the price of a possible degradation of the recommendation quality and more feedback needed to reach an optimal solution. We will explore this possibility in Chapter 7. In Chapter 5 we will discuss an alternative approach that may, in principle, be able to scale to arbitrarily large spaces by decomposing the constructive problem into "parts" and solve sub-utility maximization one part at the time [84]. This could potentially attain an exponential speedup per iteration with respect to standard constructive preference elicitation, allowing us to scale to much larger constructive problems. Besides the increase in the amount of feedback needed, a drawback of this approach is that it is only able to reach a "local" optimum with respect to the user utility. Nevertheless, the local optimum found by this technique has bounded approximation error, and provides a good trade-off between domain scalability and risk of sub-optimality.

In our constructive preference elicitation framework based on coactive learning, the expressiveness of the utility function is on par with state-of-the-art preference techniques using GAI models and recommenders using soft constraints. As it is, though, coactive learning assumes the utility of the user to be fixed and cannot be extended with more informed preference criteria during the elicitation process. This remains a research problem that needs to be addressed. We aim at doing so in Chapter 6, in which we introduce an extension to coactive learning, combining it with example critiquing [255]. This technique progressively enlarges the feature space that the algorithm works with, thereby making the utility model more expressive. Using this technique, we can personalize the preference criteria of each user utility model, without the need of stating them beforehand, which is advantageous for both generalization and computational speed. This approach might require more effort from the user, yet with clever selection criteria we can ask the user to state a critique only when strictly necessary. We will show how this method is capable of reaching an optimal solution while requiring far fewer features than standard coactive learning.

Further contributions of this thesis consist in extensively testing our technique on several different constructive scenarios, as well as the implementation of a number of applications. In this thesis we present two applications of particular interest, namely automated layout synthesis, a kind of design process, and bundling of products and services, respectively treated in Chapter 7 and 8. The latter has also served as a test bed for an empirical study with real participants, whose results highlight the practical benefits of our framework [83].

This thesis is the result of the following peer-reviewed publications (in chronological order):

[256] "Structured Feedback for Preference Elicitation in Complex Domains". Teso, S.; Dragone, P.; Passerini, A. In *BeyondLabeler Workshop at IJCAI*, 2016.

[82] "Constructive Layout Synthesis via Coactive Learning". Dragone, P.; Erculiani, L.; Chietera, M. T.; Teso, S.; and Passerini, A. In *Constructive Machine Learning Workshop at NIPS*, 2016.

[255] "Coactive Critiquing: Elicitation of Preferences and Features". Teso, S.; Dragone, P.; and Passerini, A. In *AAAI Conference on Artificial Intelligence*, 2017.

[81] "Constructive Recommendation". Dragone, P. In *ACM Conference on Recommender Systems (RecSys)*, 2017.

[85] "Constructive Preference Elicitation". Dragone, P.; Teso, S.; and Passerini, A. In *Frontiers in Robotics and AI, vol. 4.* 2018.

[86] "Constructive Preference Elicitation over Hybrid Combinatorial Spaces". Dragone, P.; Teso, S.; and Passerini, A. In *AAAI Conference on Artificial Intelligence*, 2018.

[84] "Decomposition Strategies for Constructive Preference Elicitation". Dragone, P.; Teso, S.; Kumar, M.; and Passerini, A. In *AAAI Conference on Artificial Intelligence*, 2018.

[87] "Pyconstruct: Constraint Programming meets Structured Prediction". Dragone, P.; Teso, S.; and Passerini, A. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

[93] "Automating Layout Synthesis with Constructive Preference Elicitation". Erculiani, L.; Dragone, P.; Teso, S.; and Passerini, A. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD)*, 2018.

[83] "No More Ready-made Deals: Constructive Recommendation for Telco Service Bundling". Dragone, P.; Giovanni, P.; Vescovi, M.; Tentori, K.; and Passerini, A. In *ACM Conference on Recommender Systems (RecSys)*, 2018.

1.3 Thesis outline

The following is an overview of the thesis structure and a brief summary of each chapter.

Chapter 2: Preferences and recommendations

This is the first of the two chapters discussing the background knowledge needed for developing our constructive preference elicitation framework. This chapter, in particular, details the topics of preference elicitation and recommendation systems. In the first part, the decision-theoretic treatment of preferences is introduced, together with its application to utility elicitation algorithms. The state-of-the-art approaches for preference elicitation are also reviewed. In the second part, the literature on recommendation systems is reviewed, focusing on constraint-based techniques and their application to product configuration.

Chapter 3: Online and structured learning

This chapter introduces three important paradigms of learning that are the basis for our constructive preference elicitation approach, namely online learning, structured-output prediction and coactive learning. The three are reviewed through the lens of the online convex optimization framework, which provides a simple yet powerful method for devising learning algorithms in all three those scenarios. Online convex optimization also provides the proof techniques used to analyze those algorithms.

Chapter 4: Constructive recommendation

In this chapter, we will formally introduce our constructive recommendation technique. We will detail the components of constructive recommenders, their characteristics, and their evaluation. We will also compare the constructive recommendation framework to the alternative techniques introduced in the previous chapters. We will then highlight the research problems that we ought to solve, which will then be addressed in the next two chapters. Finally, we point out several applications of the constructive recommendation framework. This chapter is based on the work published in [256, 81, 86, 85].

Chapter 5: Part-wise domain decomposition

This chapter details our methodology for partitioning the domain of a constructive decision problem with the aim of reducing inference overhead and cognitive effort for the user. We first introduce the formal framework for decomposing the domain and the utility function. Next, we propose and analyze a coactive learning algorithm relying only on partial interaction. The analysis will result in a convergence guarantee to a local optimum with bounded approximation error. Experiments are also presented to validate the algorithm performance. The technique and the results presented in this chapter are based on [84].

Chapter 6: Critiquing and feature elicitation

In this chapter, we introduce a method for augmenting coactive learning with example critiquing to personalize the features used in the utility model and solve the issue of learning with a combinatorial explosion of possibly relevant features. We first review the literature on critiquing systems and then propose a "coactive critiquing" approach, which elicits both preferences and features through coactive learning and example critiquing respectively. We analyze the proposed algorithm showing its eventual convergence to optimality. Finally, we report experimental evidence of the performance of our algorithm compared to more informed baselines. The coactive critiquing method and the experimental results exposed in the chapter are based on [255].

Chapter 7: Automated layout synthesis

This is the first of two chapters on applications of constructive preference elicitation to real-world problems. In this chapter we discuss the development of an automated layout synthesis system based on constructive recommendation, which could be integrated into CAD-like design software as a design-aiding component. We describe our approach and test it on two

Symbol	Meaning
x	Input object / contextual information
y	Output object / recommendation
$\mathcal{X}$	Input space / set of contexts
$\mathcal{Y}$	Output space / set of feasible objects
$\boldsymbol{\phi}(x, y)$	Feature map
$u(x, y)$	Generic utility function
$\boldsymbol{w}$	Generic parameter vector
y^*	True label / best object
$\boldsymbol{w}^*$	Best hypothesis / true weight vector
$u^*(x, y)$	True utility function
T	Time horizon
$t \in [T]$	Iteration index
x_t	Input at iteration t
y_t	Prediction / recommendation at iteration t
$u_t(x, y)$	Utility function at iteration t
$\boldsymbol{w}_t$	Estimate of parameter vector at iteration t
$\textsc{reg}(x_t, y_t)$	Instantaneous regret at iteration t
$\textsc{reg}_T$	Average regret up to iteration T
α	Informativeness parameter (coactive learning)
ξ_t	Slack variables / violations to the α-informative feedback
ζ_t	Utility leak / utility mismatch

Table 1.2: List of symbols and their meaning.

layout synthesis tasks, reporting our findings. The content of this chapter is based on the work from [82, 93].

Chapter 8: Product and service bundling

This chapter describes the application of constructive recommendation to product and service bundling, and, in particular, to the recommendation of telecom service bundles. In this work we implemented a full constructive recommendation system as a web application and tested it with the help of real users. In the chapter we first describe the system structure and detail its components. We then describe the experimental setting and the evaluation protocol. Finally, we present our results and draw interesting conclusions. This chapter is based on the results published in [83].

Chapter 9: Conclusions

This chapter concludes the thesis and provides a recap of the contributions of this thesis, and describes the possible research directions to undertake as future work.

1.4 Remarks on notation

Throughout this thesis, we will attempt to maintain a common underlying notation, despite the vast heterogeneity of exposed concepts. The aim is to make it simpler to relate the many, seemingly unconnected, components that make up the background knowledge that the thesis work is based on. For this reason, in many sections, the notation used in this thesis may differ significantly from the one of the original works.

Throughout the thesis, we will indicate recommended objects as well as predicted outputs with the letter y, while reserving x for input objects and contextual information. Bold letters indicate vectors and vector-valued functions. Sometimes we will employ the notation $\mathbf{y}$ to explicitly point out that the object is represented as a vector, while the notation y implies that the representation of the object is underspecified. Calligraphic letters such as $\mathcal{X}$ and $\mathcal{Y}$ are generally reserved for vector spaces or other sets with an underspecified structure. In many situations we will use abbreviations: $[n] = \{1, \dots, n\}$, $\{i : j\} = \{i, i{+}1, \dots, j{-}1, j\}$, as well as $\bigcup A = \bigcup_{a \in A} a$ and $\bigcap B = \bigcap_{b \in B} b$. Throughout all the chapters, we will denote the usual dot product in $\mathbb{R}^n$ as $\langle \mathbf{a}, \mathbf{b} \rangle = \sum_{i=1}^{n} a_i b_i$, and the Euclidean norm as $\|\mathbf{a}\| = \sqrt{\langle \mathbf{a}, \mathbf{a} \rangle}$. Other norms that we will use will be always explicitly denoted, such as $\|\cdot\|_0$ and $\|\cdot\|_1$ for the ℓ_0 and ℓ_1 norms respectively, and $\|\cdot\|_\infty$ for the infinite norm. Table 1.2 provides a non-exhaustive list of reserved symbols used throughout this thesis.

Part I

Background

Chapter 2

Preferences and recommendations

Understanding preferences has long been a topic of interest in several academic fields, such as psychology [147, 249], statistics [104, 233] and economics [170, 186]. Modeling and reasoning about preferences has also been a great interest of researchers in artificial intelligence [80, 206], who aim at developing decision support systems capable of aiding users in making optimal choices when facing complex decision problems. Such systems aid the user by building a *model* of their preferences, which can then be used to infer the most preferred outcome. To build such a preference model, decision support systems need to collect information about the user preferences, typically by interacting with and asking questions to the user, a process commonly known as *preference elicitation* [35, 37, 56, 119]. Preference elicitation systems use specialized algorithms to estimate a preference model and search for the best option for the user among a large set of candidates, while minimizing the amount of questions asked to the user [52]. In Section 2.1 we will survey the topic of preference handling in AI and describe the most well known preference elicitation algorithms in the literature.

In parallel, especially after the proliferation of e-commerce websites, AI researchers have been developing systems capable of providing *recommendations* to users, as a way of addressing the problem of choice overload [25]. Recommendation systems have then spread to many other sectors and industries, and have been shown to increment user satisfaction and fidelity, as well as increase diversification and overall revenue [221]. Several techniques have been developed in this area over the years, addressing many different recommendation scenarios and applications [2, 220]. Generally speaking, recommendation techniques can be divided into two separate categories [221]: [i] *data-driven* approaches [7], including *collaborative* [92, 153] and *content-based* [172] filtering, which exploit past purchases, ratings and other data sources to "match" users to interesting items; [ii] *knowledge-based* approaches [45], which employ an explicit representation of the domain to select recommendations that comply with the user requirements. *Constraint-based* recommenders [97, 99] are knowledge-based systems that encode domain knowledge through *constraint satisfaction programs* and select recommendations by solving them. Constraint-based recommendation systems are the most closely related to our constructive preference elicitation framework, so we will concentrate our discussion around them in Section 2.2.

In the following sections we will describe the most important aspects of preference elicitation algorithms and constraint-based recommendation systems, highlighting their properties and use cases, and finally pointing out their differences and historical connections.

2.1 Preference handling

The formal treatment of preferences has its roots in decision theory [234] and neighboring branches of economics [186] and statistics [233]. The development of the mathematical framework for describing and manipulating preferences still used today is primarily due to the work of von Neumann and Morgenstern [186], Keeney and Raiffa [150], and Fishburn [103, 104]. In their work, preference elicitation [150] was primarily seen as the process of gathering preference information prior to the construction of a *preference model*, which could then be used to make informed decisions. This process was once carried out by a decision analyst interacting with the *decision maker* (DM). This process was, however, error prone and only viable for small decision problems, due to the inherent difficulty for human decision makers, as well as decision analysts, to handle large preference models. For this reason, preference elicitation has caused much interest in the artificial intelligence community [76, 206], which aimed at developing computational models and preference elicitation algorithms to handle complex decision problems.

One important distinction to be made in preference handling is between decision making *under certainty* and decision making *under uncertainty*. In the former case, the choices of the DM lead to certain outcomes, whereas in the latter the consequences of the DM choices are uncertain. In this thesis, we will be exclusively interested in decision theory under certainty, which encompasses decision problems like purchasing a product, choosing a movie to watch, and other tasks that do not involve uncertain outcomes.

For a preference model to be useful, it needs to be built in agreement with a certain notion of *rationality* of the DM's choices, which also drives the subsequent inference process. The standard definition of rationality in decision theory is based on the *von Neumann–Morgenstern* rationality axioms [186]. The von Neumann-Morgenstern theory assumes economic agents to behave in agreement to the goal of utility maximization. This definition is however by no means universally accepted (see e.g. [229, 170, 36]). Much work in behavioral economics and psychology has addressed this issue, highlighting the fact that the choices of human decision makers are actually better described by a *bounded rationality* model. The bounded rationality model implies that human decision makers are not able to accurately articulate their preferences considering more than a handful of variables at the time [169]. Another problem with the standard notion of rationality is that it assumes the preferences of a DM to be *invariant*, static and revealed through interaction. Many argue, instead, that preferences are *constructed* by the DM throughout the elicitation process [20, 163, 202, 203, 249]. The bounded rationality of human DM is also evinced by the fact that their stated preferences are often erroneous and inconsistent due to external factors such as distraction and fatigue [26, 51, 52, 173, 129]. This makes the elicitation process a laborious and error prone task. Artificial intelligence techniques deal with this problem by accounting for uncertainty over feasible models and by devising elicitation algorithms robust to *noise* in the user response [26, 52, 129, 272]. Despite the criticisms, the von Neumann–Morgenstern utility theory is by far the most used in AI. In this thesis, we will use a subset of the von Neumann–Morgenstern axioms (sufficient for the treatment of decision making under certainty), and we will comply with the standard notion of rationality while allowing a boundedly rational user behavior through noisy feedback.

In the following, we will briefly describe the formal framework for modeling and eliciting preferences typically used in AI. In Section 2.1.1 we will introduce the decision-theoretic formulation of preference relations and utility functions, which are the basic tools to describe and reason about preferences. In Section 2.1.2 we will detail the *multi-attribute utility theory*, which is the most common formulation used by preference elicitation techniques in AI. Next, in Section 2.1.3, we will introduce the classical preference elicitation procedure, pointing out its limitations, and then, in Section 2.1.4, we will review the state-of-the-art techniques for preference elicitation in the literature.

2.1.1 Preference modeling

In the classic decision-theoretic view, the preferences of an individual are represented by a *preference relation* $\succcurlyeq$ over a set $\mathcal{Y}$ of outcomes. For $y, y' \in \mathcal{Y}$, the notation $y \succcurlyeq y'$ intuitively means that y is at least as good as y'. A *rational* agent is expected to maintain a preference relation satisfying the following axioms [186]:

1. Comparability: $\forall\, y, y' \in \mathcal{Y} \qquad y \succcurlyeq y' \ \vee\ y' \succcurlyeq y$
2. Transitivity: $\forall\, y, y', y'' \in \mathcal{Y} \qquad y \succcurlyeq y' \ \wedge\ y' \succcurlyeq y'' \implies y \succcurlyeq y''$

A binary relation with the above properties corresponds to a *total preorder* over the set $\mathcal{Y}$. The total preorder induces a complete directed graph over the elements of $\mathcal{Y}$, whose edges connect y and y' if and only if $y \succcurlyeq y'$. Cycles are admissible in a total preorder, in which case we have that $y \succcurlyeq y' \ \wedge\ y' \succcurlyeq y$, which is usually abbreviated by an *indifference relation* $y \sim y'$, meaning that neither object is preferred to the other. When $y \succcurlyeq y' \ \wedge\ y' \not\succcurlyeq y$, instead, we use a *strict* preference relation $y \succ y'$, meaning that y is strictly preferred to y'.

Under the above comparability and transitivity conditions, when the set of outcomes $\mathcal{Y}$ is finite or countably infinite, it is possible to define an *ordinal utility function* $u : \mathcal{Y} \to \mathbb{R}$ such that:

$$y \succcurlyeq y' \iff u(y) \geq u(y')$$

The same holds for uncountably infinite outcome sets, provided the preference relation $\succcurlyeq$ is *continuous* over the set $\mathcal{Y}$. Continuity of a preference relation over a compact set $\mathcal{Y}$ is achieved if "small" variations to y and y' do not change their order of preference. More formally, for a continuous preference relation, if $y \succ y'$, there exist two balls B and B', around y and y' respectively, such that for every element $z \in B$ and $z' \in B'$, $z \succ z'$. It can be shown that a preference relation is continuous if and only if it can be represented by a continuous utility function [73].

Every utility function u represents a unique preference relation $\succcurlyeq$. The opposite is however not true, since the same preference relation can be represented by any utility function that is a monotonically increasing transformation of u. That is, if f is a monotonically increasing function, $f(u(y))$ represents the same preference relation of $u(y)$. In other words, ordinal utility functions are unique up to monotonically increasing transformations.

2.1.2 Multi-attribute utility theory

Outcomes of complex decision problems can typically be evaluated along different directions. For instance, when buying a house one needs to look at several aspects, such as the square footage, the number of rooms, the proximity to work and schools, the neighborhood, and the cost. These are usually called *attributes* (or *features*) of the objects one is making the decision over. More formally, *multi-attribute utility theory* (MAUT) [150] describes outcomes as vectors $\mathbf{y}$ of m variables (a.k.a. attributes) taking values from domains $\mathcal{Y}_1, \ldots, \mathcal{Y}_m$. The space of possible outcomes is the Cartesian product of the variables domains $\mathcal{Y} = \mathcal{Y}_1 \times \cdots \times \mathcal{Y}_m$. Given an index set $I \subseteq [m]^1$, we can define a *partial* outcome $\mathbf{y}_I \in \mathcal{Y}_I = \times_{i \in I} \mathcal{Y}_i$. We denote the complement of I as $I^- = [m] \setminus I$. Given a partial assignment $\mathbf{y}_I$ and its complement $\mathbf{y}_{I^-}$, we define a *completion* $\mathbf{y} = (\mathbf{y}_I, \mathbf{y}_{I^-}) \in \mathcal{Y}$.

If preferences over multi-attribute domains exhibit sufficient structure, preference models can be compactly encoded and efficiently queried. In a multi-attribute setting, structure is often determined by some kind of independence between attributes. The most basic is the *preferential independence* between a subset I of attributes and its complement:

$$\forall\ \mathbf{y}_I, \mathbf{y}'_I \in \mathcal{Y}_I,\ \ \tilde{\mathbf{y}}_{I^-}, \tilde{\mathbf{y}}'_{I^-} \in \mathcal{Y}_{I^-} \qquad (\mathbf{y}_I, \tilde{\mathbf{y}}_{I^-}) \succcurlyeq (\mathbf{y}'_I, \tilde{\mathbf{y}}_{I^-}) \iff (\mathbf{y}_I, \tilde{\mathbf{y}}'_{I^-}) \succcurlyeq (\mathbf{y}'_I, \tilde{\mathbf{y}}'_{I^-})$$

This means that if a subset of attributes I is preferentially independent of the remaining attributes I^-, then the preferences over attributes I do not depend on the attributes I^-, as long as they are kept fixed. The statement $(\mathbf{y}_I, \tilde{\mathbf{y}}_{I^-}) \succcurlyeq (\mathbf{y}'_I, \tilde{\mathbf{y}}_{I^-})$ is usually abbreviated as $\mathbf{y}_I \succcurlyeq \mathbf{y}'_I$ *ceteris paribus* (all else being equal).

If preferential independence holds for *every subset* of attributes with respect to its complement we say that they are *additively independent.* If additive independence holds, the utility function can be decomposed into m *subutilities* u_i, each dependent on a single attribute y_i, for $1 \leq i \leq m$:

$$u(\mathbf{y}) = \sum_{i=1}^{m} u_i(y_i)$$

We call the latter an *additive* utility. While the assumption leading to a utility in additive form are rather strong, it is often used in practice thanks to the marked simplification it endows on the elicitation process. Elicitation of an additive utility only involves finding the utility value of the best and the worst value of each attribute separately, which can then be scaled and straightforwardly combined thanks to their mutual preferential independence.

^{1}Recall that the notation $[m]$ stands for $\{1, \ldots, m\}$.

Algorithm 1 Classical preference elicitation procedure

```
1: Select a query
2: Ask query to the DM
3: Receive response from the DM
4: Update preference model
5: Repeat steps 1 – 4 until the preference model is complete
6: Recommend best outcome based on preference model
```

A more relaxed independence condition is the *conditional preferential independence.* Let $[m]$ be partitioned into three non-overlapping sets I, I', I''. The sets I, I' are *conditionally* preferential independent given $\mathbf{z} \in \mathcal{Y}_{I''}$ if and only if:

$$\forall\, \mathbf{y}_I, \mathbf{y}'_I \in \mathcal{Y}_I,\ \tilde{\mathbf{y}}_{I'}, \tilde{\mathbf{y}}'_{I'} \in \mathcal{Y}_{I'} \quad (\mathbf{y}_I, \tilde{\mathbf{y}}_{I'}, \mathbf{z}) \succcurlyeq (\mathbf{y}'_I, \tilde{\mathbf{y}}_{I'}, \mathbf{z}) \iff (\mathbf{y}_I, \tilde{\mathbf{y}}'_{I'}, \mathbf{z}) \succcurlyeq (\mathbf{y}'_I, \tilde{\mathbf{y}}'_{I'}, \mathbf{z})$$

The above condition induces a *conditional* preference relation $\succcurlyeq_{\mathbf{z}}$ for all $\mathbf{z}$ over which I and I' are preferential independent, thus we can write $\mathbf{y}_I \succcurlyeq_{\mathbf{z}} \mathbf{y}'_I$ ceteris paribus. When this independence condition holds for some attribute sets, the preference relation can be represented by a *conditional preference network* (CP-net) [28]. CP-nets are directed graphs in which each node represents an attribute, whose incident edges represent its preferential dependencies with other attributes. For each node, a *conditional preference table* completely specifies the conditional preference relation $\succcurlyeq_{\mathbf{z}}$ of the node given every assignment $\mathbf{z}$ to the values of its parent nodes.

Another form of relaxed independence is the *generalized additive independence* (GAI) [12, 103] assumption, under which a utility function can be decomposed into independent subutilities over a collection of K (possibly overlapping) subsets of attributes:

$$u(\mathbf{y}) = \sum_{k=1}^{K} u_i(\mathbf{y}_{I_k})$$

This condition is much more general than additive independence and it has been shown that any *acyclic* CP-net admits a GAI utility function over the collection of attribute sets defined by all attributes with their respective parent nodes [34]. By merging acyclic CP-nets with GAI decomposition it is possible to devise elicitation strategies employing the *minimax regret* criterion (see below) [27]. Other strategies have been developed later on [34, 154].

In Chapter 5 we will employ some of the results in the literature on GAI utilities to develop a part-based elicitation process for constructive preference elicitation.

2.1.3 Classical preference elicitation

Decision processes were once split into two separate phases: an *elicitation* phase, in which a preference model is built, and a *recommendation* phase, in which the model is used to help the DM in making an optimal decision. Preference models such as CP-nets were considered static objects that, once obtained, could be used to infer the answer to different types of queries, similarly to other types of graphical models like Bayesian networks [205]. Querying a preference model might be useful to assess e.g. *dominance* or *indifference* between an object y over y', i.e. whether $y \succ y'$ or $y \sim y'$, or it could be used to perform *outcome optimization*, i.e. finding out the outcome with the highest utility, perhaps given some partial assignment of the attributes. The optimal outcome could then be *recommended* to the decision maker. In order to attain an optimal decision, it was implicitly assumed that the preference model should be elicited *completely*, i.e. finding the true preference relation $y \succcurlyeq y'$ for each pair of items (y, y'), or equivalently finding the true utility value $u(y)$ for each object y.

Preference (or utility) elicitation was an interactive process, between a decision analyst and a decision maker, aimed at constructing a preference model. This process usually consisted in a sequence of "queries" asked to the decision maker in order to gather information about his or her preferences. A typical preference elicitation loop would progress as shown in Algorithm 1. Common types of queries used in the literature include:

- Pair-wise comparison: the DM is asked to state her preference between two objects $\mathbf{y}$ and $\mathbf{y}'$. This query conveys the information $\mathbf{y} \succ \mathbf{y}'$ or vice versa. Sometimes, an indifference answer $\mathbf{y} \sim \mathbf{y}'$ is also allowed.
- Set-wise comparison: the DM is asked to pick the most preferred outcome among a set Q of alternatives. Given an answer $\mathbf{y}$, then we can implicitly deduce that $\mathbf{y} \succcurlyeq \mathbf{y}'$ for $\mathbf{y}, \mathbf{y}' \in Q$, $\mathbf{y} \neq \mathbf{y}'$. Sometimes set-wise queries require the DM to order the objects according to her preferences, in which case the answer to the query provides a much more accurate preference information, resulting in all combinations of pair-wise preference rankings.
- Gambles: the DM is asked whether she would prefer an object $\mathbf{y}$ for sure, or a gamble between $\mathbf{y}'$ with probability p and $\mathbf{y}''$ with probability $1 - p$. A gamble is usually denoted as $\langle \mathbf{y}', p\,; \mathbf{y}'' \rangle$, and the result of a choice between a gamble and a certain outcome determines $\mathbf{y} \succ \langle \mathbf{y}', p\,; \mathbf{y}'' \rangle$ or vice versa. If $\mathbf{y}'$ is the best possible object $\mathbf{y}^\top$ and $\mathbf{y}''$ is the worst possible object $\mathbf{y}^\perp$, then $\langle \mathbf{y}^\top, p\,; \mathbf{y}^\perp \rangle$ is called a *standard gamble* and we have that the result of query between $\mathbf{y}$ and a standard gamble equates to $u(\mathbf{y}) > p$ or vice versa, assuming $u(\mathbf{y}^\perp) = 0$ and $u(\mathbf{y}^\top) = 1$. If the DM is indifferent between the gamble and the certain outcome, $\mathbf{y} \sim \langle \mathbf{y}^\top, p\,; \mathbf{y}^\perp \rangle$ then $u(\mathbf{y}) = p$.

Many approaches to preference elicitation have been proposed over the years, most relying on some assumption about the structural independence of the attributes to simplify the elicitation process and using different types of queries to optimize the amount of information gained per query.

For additive independent utility functions, for instance, the elicitation process is rather simple and straightforward [150]. An additive utility function can be described using m *local value functions* v_i and m *scaling factors* λ_i:

$$u(y) = \sum_{i \in [m]} u_i(y) = \sum_{i \in [m]} \lambda_i v_i(y)$$

Algorithm 2 Modern preference elicitation procedure

```
1: while User is not satisfied do
2:     Select a recommendation based on partial preference model
3:     Make recommendation to the DM
4:     Receive feedback (implicit or explicit) from the DM
5:     Update preference model
```

Since the subutilies are all independent of each other, one can proceed in eliciting each attribute separately. Once a ranking for each attribute has been established, e.g. by local pairwise or set-wise comparisons over the attributes values, let $y_i^\top$ and $y_i^\perp$ be the best and worst attribute values for each attribute $i \in [m]$, and set $v_i(y_i^\top) = 1$ and $y_i(y_i^\perp) = 0$. The value function $v_i(y_i)$ can now be determined *locally* for each value of the attribute, using standard gambles with increasing probabilities. Once the local value functions have been determined, one only needs to elicit the true utility of the best values of each attribute in order to assign the right value to the scaling factors λ_i. Let $\mathbf{y}^\perp = (y_1^\perp, \dots, y_m^\perp)$ and $u(\mathbf{y}^\perp) = 0$, as well as $\mathbf{y}^\top = (y_1^\top, \dots, y_m^\top)$ and $u(\mathbf{y}^\top) = 1$. We can use standard gambles over full objects to determine the true utility of $u(y_i^\top, \mathbf{y}_{i^-}^\perp)$[2]: if $(y_i^\top, \mathbf{y}_{i^-}^\perp) \sim \langle \mathbf{y}^\top, p; \mathbf{y}^\perp \rangle$ then $u(y_i^\top, \mathbf{y}_{i^-}^\perp) = p$. The scaling factors can then be set to:

$$\lambda_i = \frac{u(y_i^\top, \mathbf{y}_{i^-}^\perp)}{\sum_{j \in [m]} u(y_j^\top, \mathbf{y}_{j^-}^\perp)}$$

This ensures that the subutilities functions are properly scaled and that the full utility is defined in the $[0, 1]$ range. Similar elicitation procedures based on local elicitation and global scaling can be devised for GAI models as well [39].

The classical paradigm for preference elicitation has fallen out of fashion over the years due to several practical problems [52]. First and foremost, gathering complete preference information is unattainable for decision problems with more than a handful of attributes. Additive utility functions are often not descriptive enough, resulting in erroneous models that do not really represent the user preferences, thereby leading to suboptimal decisions. Another important factor to take into account is the cognitive effort of decision makers, especially non-expert ones. Queries involving cardinal utility values and probabilities are too difficult for non-expert decision makers. Also, in general, the number of queries asked to the DM should be minimized to reduce the cognitive burden. These problems lead to a shift in perspective in preference elicitation, which now is mostly based on the assumption that algorithms should help the user in finding optimal decisions with only *partial* preference information. Also, a partially elicited preference model could be used to select the *most informative* query to ask to the DM, in order to minimize the number of queries. To further reduce cognitive effort for the user, most recent preference elicitation systems adopt comparison queries, which are deemed to require low cognitive effort [64, 128, 129, 173]. Another advantage of comparison preference feedback is that it can often be inferred *implicitly* from the behavior of a user on an online system. Viappiani and Boutilier [272, 273, 274] showed that, in the context of

[2]With a slight abuse of notation $i^- = [m] \setminus \{i\}$.

preference elicitation with set-wise preference queries, the optimal recommendation set myopically coincides with the optimal query set, thereby merging the querying phase and the recommendation phase into one single process. The new paradigm for preference elicitation and decision making proceeds as in Algorithm 2. The preference elicitation process now recommends the best item according to the current partial preference model and stops only when the user is satisfied with the recommendation. Modern preference elicitation algorithms are designed along these lines [18, 31, 129, 272, 274, 278]. In the next section we will describe two of these state-of-the-art algorithms for *regret*-based and *Bayesian* preference elicitation.

2.1.4 Regret-based and Bayesian elicitation

To make optimal recommendations with only partial preference information, one needs to take into account the *uncertainty* resulting from the lack of a full preference knowledge. The two most used paradigms for handling uncertainty in utility models are based on the *minimax regret* principle and *Bayesian statistics.*

Within a *strict* uncertainty scenario, i.e. without any distributional information about the possible utility functions, a well known decision criterion is the *minimax regret* [234]. Descriptively, minimax regret suggests to choose the possibility $\mathbf{y}$ which minimizes the maximum regret over both $\mathcal{U}$ and $\mathcal{Y}$, i.e. the worst-case loss one would suffer by choosing $\mathbf{y}$. Formally, given a set of utilities $\mathcal{U}$ over a choice space $\mathcal{Y}$, the minimax regret principle imposes:

$$\mathbf{y}^* = \underset{\mathbf{y}\in\mathcal{Y}}{\operatorname{argmin}} \max_{u\in\mathcal{U}} \max_{\mathbf{y}'\in\mathcal{Y}} u(\mathbf{y}') - u(\mathbf{y})$$

In practice, the calculation of the above optimization problem depends on the structure of the utility models $u \in \mathcal{U}$ (often GAI models are used) and the type of interaction with the user. In general, the minimax regret problem can be cast as a *mixed integer program* and can be solved through constrained optimization [201, 238]. This has the side benefit of allowing us to impose arbitrary constraints on the decision space in order to limit the feasible area. As more preference information is acquired throughout the elicitation process, more constraints are imposed on the space $\mathcal{U}$, shrinking the space of feasible utility functions and thus reducing the uncertainty about the decision. The minimax regret has been applied to many elicitation scenarios, with different utility models and types of feedback. One major drawback of this approach that has emerged over the years, is that it fails to take into account uncertainty in the user feedback. Indeed, user feedback is inherently noisy, as human decision makers make mistakes and inaccuracies due to lack of focus and other factors extraneous to the decision process. Noisy feedback leads to inconsistencies, which can make a regret-based system never reach an optimal solution. Assuming noiseless responses is unrealistic, especially for systems involving interaction with non-expert users, who do not fully understand the constraints of the problem and the consequences of their choices.

An alternative approach that does take into account noisy feedback is *Bayesian* preference elicitation. In this case, uncertainty is handled through a probability density function $\pi(u)$ over the space of utility functions $\mathcal{U}$. Given the current *belief* π, the optimal decision would be the one maximizing the *expected utility* $\mathbb{E}_\pi[u(\mathbf{y})]$ with respect to the prior π:

$$\mathbf{y}^* = \operatorname*{argmax}_{\mathbf{y}\in\mathcal{Y}} \mathbb{E}_\pi[u(\mathbf{y})] = \operatorname*{argmax}_{\mathbf{y}\in\mathcal{Y}} \int_{u\in\mathcal{U}} u(\mathbf{y})\pi(u)\mathrm{d}u$$

However, throughout the elicitation, one would rather select the query yielding the most preference information. One popular approach is maximizing the *expected value of information* (EVOI) [26, 52]. The EVOI is the difference between the *expected posterior utility* (EPU) and the maximal expected utility over the current belief. The EPU is the maximal utility in expectation over the *posterior* distribution according to the *response model* of the selected query. A user response model is a distribution $P(r|q,u)$ over the possible responses $r \in R_q$ to a query $q \in \mathcal{Q}$ given utility $u \in \mathcal{U}$. The response model used depends on the type of queries the elicitation process employs. Typical response models for e.g. comparison queries are the Bradley-Terry or Plackett-Luce models [33, 175, 208], which are logistic models of the type:

$$P(r|q,u) = \frac{\exp(\gamma\, u(r))}{\sum_{s\in R_q} \exp(\gamma\, u(s))}$$

For a response $r \in R_q$ to query $q \in \mathcal{Q}$, Bayes rule yields the posterior probability $\hat{\pi}$ over $\mathcal{U}$:

$$\hat{\pi}_{r,q}(u) = \frac{P(r|q,u)\pi(u)}{P(r|q)}$$

where $P(r|q) = \int_{u\in\mathcal{U}} P(r|q,u)\pi(u)\mathrm{d}u$ is the marginal probability of response r given q. The expected posterior utility of a query $q \in \mathcal{Q}$ is:

$$\mathrm{EPU}(q) = \sum_{r\in R_q} P(r|q) \left(\max_{\mathbf{y}\in\mathcal{Y}} \mathbb{E}_{\hat{\pi}_{r,q}}[u(\mathbf{y})] \right)$$

The EVOI of a query $q \in \mathcal{Q}$ is the difference between EPU and the maximal expected utility according to the current belief π; the best query q^* is the one maximizing the EVOI:

$$q^* = \operatorname*{argmax}_{q\in\mathcal{Q}} \mathrm{EVOI}(q) = \operatorname*{argmax}_{q\in\mathcal{Q}} \left(\mathrm{EPU}(q) - \max_{\mathbf{y}\in\mathcal{Y}} \mathbb{E}_\pi[u(\mathbf{y})] \right)$$

The above definition considers a *myopic* optimization of the EVOI [52], while the general *sequential* case would require to optimize the EVOI considering possible future queries as well, i.e. by treating the elicitation as a partially observed Markov decision process (POMDP) [26]. Sequential EVOI maximization is unattainable even for small decision problems, so myopic optimization is used in practice. Even for myopic EVOI optimization, EPU maximization is computationally expensive and often impractical. Viappiani and Boutilier [272] showed that an alternative greedy optimization algorithm achieves bounded approximation error. They also proposed a faster randomized alternative but with no theoretical guarantees. Despite the latter approximate approach, Bayesian preference elicitation still requires a high computational cost, which is a major shortcoming for scaling to very large decision domains.

2.2 Recommendation systems

Recommender systems are computational tools providing suggestions in regard to a decision task the user is faced with. Typical scenarios in which a recommender system is used include buying a product, choosing which movie to stream, listening to a playlist while traveling, reading a news article online. In all these tasks, the choice is among a set of *items*, and the job of the recommender system is to pick a few items from the bunch that are likely to be of interest to the current user. This implies that the recommender system has to provide *personalized* suggestions to each user separately, based on different criteria such as demographics, location, browsing and purchase history, likes, ratings and reviews. It is usually the case that a recommender system gathers all this information into a *user profile*, which is then matched to the features of the items in search of promising recommendations. Indeed, this is a form of *content-based recommendation* [172], in which learning algorithms and similarity measures are used to determine how "similar" an item is to a user profile or to the items the user has liked in the past. Other content-based techniques treat the recommendation task as a *multi-armed bandit* problem [41, 160], in which the recommender system has to *choose* which item to display to the current user, balancing between *exploration* and *exploitation*, in order to maximize some overall expected reward, e.g. click-through rate. Another data-driven approach to recommendation is *collaborative filtering* [92, 153], in which the recommendations for a user are determined on the basis of her behavior with respect to other users' behavior. The standard approach to determine how much a user liked a given item is through *ratings*. The assumption in collaborative filtering is that users rating items in the same way will likely rate other items similarly too. In this way, recommendations are simply based on how other users with similar ratings to the current user have rated items that the current user has not yet rated. This avoids the need for extracting item features entirely, yet requiring much more data from a large number of users in order to make good quality recommendations. Collaborative filtering also suffers from the *cold start* problem for new users [164, 236], as no data is available for them to provide meaningful recommendations. As such, often hybrid techniques are used to address this problem [43, 288]. A radically different approach is taken by *knowledge-based* techniques, which represent the knowledge about the items properties and their relation to the user needs and preferences explicitly. They often rely on some kind of structure in describing the relations between items and their properties and users and their preferences, such as graphs, logical predicates and constraints. Inference algorithms over these structured models can then be used to make recommendations. One particular subset of knowledge-based techniques are the *constraint-based* recommender systems, which cast the recommendation task into a *constraint satisfaction problem* [183, 268]. In the following we will describe constraint-based recommendation in more details, as it is the most related to the constructive recommenders that will be introduced in Chapter 4. In the next section, we will detail the constraint-based recommendation paradigm and its many variants. In Section 2.2.2, we will then describe the *product configuration* problem and the systems that are typically used to solve it [100]. We will also see how constraint-based recommendation can be employed for product configuration tasks, which is a particularly relevant point in the context of constructive recommendation.

2.2.1 Constraint-based recommendation

Constraint-based recommenders [97, 99, 266] are a kind of knowledge-based systems in which domain knowledge as well as user preferences are represented as *constraints*. This formalism is particularly beneficial in recommending complex objects like cars, financial services and insurance policies, bundles, and configurable products such as PC configurations and trips. In all these scenarios, items are infrequently purchased objects, for which it is difficult to maintain a sizable and updated set of ratings for each user, making data-driven approaches ineffective.

In constraint-based recommendation, the problem of selecting a recommendation is cast as a *constraint satisfaction problem* (CSP) [268]. Given a set of m *variables* $Y = \{y_1, \ldots, y_m\}$ with *domains* $\mathcal{Y}_1, \ldots, \mathcal{Y}_m$ and a set of *constraints* C, a constraint satisfaction problem consists in finding one or more *assignments* to all variables Y, i.e. instantiations of the variables $y_i \in \mathcal{Y}_i$ from their respective domains, that also satisfy all the constraints C. Formally, a possible assignment is a tuple $y = (y_1, \ldots, y_m) \in \mathcal{Y} = \mathcal{Y}_1 \times \cdots \times \mathcal{Y}_m$ and a constraint $c \in C$ is a set of valid assignments[3]. A *solution* of the CSP is a feasible assignment $y \in \mathcal{Y} \cap \mathcal{C}$, where $\mathcal{C} = \bigcap C$ is the set of feasible assignments. In a constraint-based recommender, the variables Y encode the attributes of the recommended objects. Another set of variables X may encode given contextual information, e.g. user attributes, which can be used in the definition of constraints. Constraints in constraint-based recommenders can mainly be of three kinds: [i] feasibility constraints, i.e. constraints encoding the realizability of configurations based on the available domain knowledge, business-driven restrictions and compatibility between components; [ii] user requirements, i.e. user-specified constraints that filter the configurations to comply with his or her preferences; [iii] catalog constrains, i.e. constraints encoding the available configurations in the item database. When a constraint-based recommender is used for product configuration (discussed in the next section), the last type of constraints is missing entirely. Solutions to the CSP are valid recommendations. Off-the-shelf CSP solvers are widely available, so the system designer is only concerned with describing the full domain knowledge and translating the specified user requirements into valid constraints.

Most constraint-based systems in the literature also incorporate methods for *ranking* the solutions found by the CSP solver [98, 101]. Multi-attribute utility theory (Section 2.1.2) has often been used for this purpose [102, 101, 142]. Usually, a linear additive utility model would be used, and the weights for each attribute would be set by the designer of system or could be modified manually by the user. Later on, systems based on *soft constraints* [183] for ranking were introduced, which allow to articulate more expressive preference trade-off rules and providing a clearer semantics to the interaction with the system [24, 211, 289]. Soft CSPs are generally understood as standard CSPs with an attached *semiring* whose elements can be assigned to the problem constraints [23]. Instantiation of this paradigm include fuzzy CSPs, probabilistic CSPs and weighted CSPs. While some work uses fuzzy and probabilistic CSPs

[3]Technically, each constraint is defined over a subset of variables $Y_c \subseteq Y$ and determines a set of the valid *partial* assignments, i.e. assignments to the variables Y_c only. For simplicity, we consider constraints as sets of full assignments containing all the combinations of feasible assignments to the variables Y_c with all the values in the domains of the other variables.

for recommendation and preference elicitation [113, 275], most works focus on weighted CSPs [211], in which each constraint is assigned a weight and the objective is to maximize the cumulative weight of the satisfied constraints.

Constraint-based recommender systems typically incorporate a way of collecting user requirements through interaction [97]. A natural way of modeling the interaction with the user in a constraint-based system is through a *dialog*, i.e. a sequence of refinement steps in which the user would assess the available options and *tweak* some of the options, providing additional requirements [165]. This type of systems are often called *conversational* recommender systems [118, 165, 279]. The refinement steps are carried out through so-called *critiques* [47, 55, 181]. Critiques may be either user initiated or elicited by the system [54]. Either way, they are statements of the type "I would like a similar product, but cheaper", i.e. statements providing additional information on the preference of the user, in order to improve the search of a satisfying option. In systems using soft constraints, critiques are usually interpreted as either additional soft constraints or a (manual) changes to the weights of the soft constraint [211, 277]. To the best of our knowledge, the first constraint-based system that combined acquiring soft constraints and *learning* the weights of the soft constraints from user feedback is the work of Rossi and Sperduti [22, 225, 227, 226]. They proposed to acquire ratings as user feedback and learn weights for the soft constraints with an incremental strategy [227]. Later, [110] used *ranking SVMs* [144] to learn from critiques, though they did not use constraint solvers to generate recommendations. In [48], the authors propose for the first time a recommendation system based on soft constraint, while learning a utility function from pair-wise preference feedback with ranking SVMs.

A constraint-based system using regret-based elicitation was proposed by Boutilier et al. [29, 274], acquiring critiques in the form of comparison feedback. Their decision problem did not employ soft constraints, but their approach encompasses GAI models, which have the same expressive power of soft constraints [29, 40]. They were the first to apply a provably optimal decision theoretic criterion for eliciting preferences in a constraint-based recommender.

2.2.2 Product configuration

Product configuration is a long standing research area in artificial intelligence [100, 126]. Product configuration is the task of combining the components of an object in such a way that the final object satisfies a set of constraints. Components can be instantiated in several different ways, and their combination form a full configuration that is subject to feasibility constraints, e.g. compatibility constraints between the components, and user requirements, i.e. based on the user needs and preferences. Due to the combinatorial explosion of the solution space, product configuration tasks are often computer-aided by *configuration systems*, whose job is to help users defining their requirements and finding feasible solutions. Configurator systems have been the main technology to enable and drive the adoption of a *mass customization* paradigm in industrial production [5].

Most configurator systems in the literature are mainly concerned with checking consistency of the user requirements and satisfiability of the overall constraint problem. Most often configuration systems assume to get the user requirements in one go, regardless of how they are obtained [5]. An alternative way to acquire user constraints is, instead, through a step-by-step process in which a user specifies the value (or range of values) for each variable one by one, and at each step the solver uses constraints propagation to check and update the current solution [5, 100]. Since product configuration is mainly used at industrial level, research in this area is mainly focused on the issues of creating, employing and maintaining CSP models for configuration tasks. Only a small subset of works in this field addresses the problem of how to drive the users towards a good solution and improve the interaction with the configuration system [98, 178, 232]. The majority of these works propose some way to combine recommendation techniques to configuration systems, in order to ease the choice of attribute values to the user. Among the others, [98] proposes several methods to address cases in which the problem constraints, together with the user requirements, yield no feasible solutions. In [178], the authors propose to use recommendation techniques to suggest which variable to fill next and with which value. Another work in this direction proposes to proceed with a guided search, aided by content-based filtering techniques, through progressively more constrained partial configuration problems [232]. In general, however, configuration systems are more used in an industrial setting or other areas in which the computer-aided configuration task is handled by a domain expert [123, 263].

Constraint-based recommenders are generally more concerned with providing the user with refined interaction protocols and more robust utility models than product configuration systems [99]. This is especially important for non-expert users, who do not have full information about the domain and do not fully understand the consequences of their choices. As mentioned in the previous section, configuration problems can be handled through a constraint-based system as well. The potential application of constraint-based recommendation to product configuration has long been acknowledged [97, 99, 287]. While the original approaches of constraint-based recommenders and configuration systems were rather different, they are now progressively converging [94, 95, 99, 96, 235, 264, 265].

2.3 Summary

In this section we surveyed two areas in the field of artificial intelligence that deal with the issue of designing computational decision support systems, namely *preference elicitation* and *constraint-based recommendation*. The former has the goal of estimating a preference model through interaction and recommend an *optimal* solution to the decision maker, whereas the latter is aimed at suggesting interesting items (not necessarily optimal in a decision-theoretic sense) based on the requirements stated by the user and the constraints of the problem. The literature of these two areas are quite distinct and come from two once separate communities. Preference elicitation came out of decision theory and was heavily influenced by neighboring fields of statistics and econometrics [104, 150, 234]. On the other hand, constraint-based recommendation was developed within the constraint programming community and was influenced by knowledge representation, formal verification and other logic-based approaches [89, 224, 228, 230, 268]. As in other areas of AI, preference elicitation, a traditionally statistical methodology, and constraint-based systems, as a form of knowledge-based approach, are progressively converging. Indeed, despite their separate origins, these two approaches are very much interdependent. Our work on constructive preference elicitation draws insight from both these approaches, providing a methodology that summarizes the best aspects of both.

Chapter 3

Online and structured learning

Recommendation systems often rely on algorithms that learn to predict which, among the catalog of available items, are interesting suggestions for the users. As such, machine learning techniques are widely used in recommendation. For instance, *matrix factorization* is commonly used in collaborative filtering, while k-nearest neighbor and deep learning are often used in content-based recommendation [204, 271]. Learning-to-rank algorithms [168] learn ranking models to optimize e.g. the click-through rate of search engines and top-k recommendation systems [144, 148]. The same objective can be achieved by *online learning* algorithms such as multi-armed bandit and contextual bandit algorithms [50, 160]. Indeed, online learning algorithms are extensively used in recommendation [2] and beyond [50]. Online learning algorithms operate in a somewhat similar setting to the preference elicitation algorithms seen in Chapter 2. Both run in an iterative fashion, making recommendations (or generally predictions) sequentially and obtaining feedback only after the prediction is made. Their objective are, however, different: while preference elicitation algorithms construct a preference model to find the *optimal* object for a user in a combinatorial space of options, online learning algorithms learn to choose actions in such a way to maximize the *cumulative gain* (e.g. clicks) of the player (e.g. a recommender system). Despite their differences, they are not incompatible, as we will argue in Chapter 4.

Another machine learning topic that is related to ranking [53] is *structured-output prediction* [14]. Structured prediction techniques are used in contexts in which the outputs of the algorithm are objects that exhibit some level of *structure*, such as sets, sequences, matrices, trees or graphs. These tasks are more complex than standard binary classification and regression, so specialized learning algorithms are needed. Interestingly, though, these algorithms have been developed in such a way to actually be *independent* of the particular structure of any given prediction problem. The only requirement for using these algorithms is to have access to an *oracle* capable of making predictions with the particular structure by solving certain optimization problems. We will discuss in Chapter 4 how this mechanism can be exploited in a constructive scenario as well.

In this chapter we will survey the online learning (Section 3.1) and structured-output prediction (Section 3.2) subfields, while linking them together using *online convex optimization*, a powerful tool for deriving online learning algorithms to solve many different prediction tasks. Lastly, we will detail the *coactive learning* framework, which is the technique our work on constructive preference elicitation is based on. Coactive learning is an online structured prediction framework that is used for personalizing search engine results and recommendations. We will argue in Chapter 4 that it may also constitute an effective preference elicitation method for constructive problems.

3.1 Online learning

Online learning [50] deals with sequential decision problems, in which predictions are made in a sequence of consecutive rounds, while information about the quality of a prediction (feedback) is revealed only after the prediction has been made. In this setting, learning occurs between two consecutive predictions, and typically involves updating the parameters of a decision model utilizing the feedback received after the prediction. The most well known online learning problems are the *multi-armed bandit* model and the problem of *prediction with expert advice*. In the multi-armed bandit problem an agent has to iteratively choose which "arm" to pull (i.e. which action to play) based on the information she has about the distribution of the reward of that arm. After the agent plays an action, only the reward of the played actions is revealed and not that of the other actions. As such, an agent has to trade-off *exploration* and *exploitation*, in order to maximize the cumulative reward. In the problem of prediction with expert advice, an agent has to choose a prediction based on the recommendations of several experts, and, once the choice is made, the reward of each expert's recommendation is revealed, but the agent only retains the one from the chosen expert. Also in this case the goal of the agent is to maximize the cumulative reward, though she has full information on the rewards of the other actions so exploration is not an issue.

An important extension to the experts problem is the prediction with *side information* [50]. This model is the most closely related to typical machine learning problems like classification and regression. Algorithm 3 shows the typical schema of an online prediction algorithm [242]. At each time step $t \in [T]$, the learner receives an object x_t from an input domain $\mathcal{X}$. The input object x_t represents the side information the learner has access to before making its prediction. After receiving the input x_t, the learner has to make a prediction $y_t \in \mathcal{Y}$. Both the input and output spaces, $\mathcal{X}$ and $\mathcal{Y}$, are defined on the basis of the task at hand. For instance, in standard classification and regression, inputs are feature vectors from $\mathcal{X} = \mathbb{R}^d$, while the output spaces $\mathcal{Y}$ are $\{0, 1\}$ and $\mathbb{R}$, respectively. We will see in Section 3.2 that more complex scenarios may be encoded using the same formalism.

After making the prediction y_t, the true output y_t^* is revealed. Given the prediction y_t and the true output y_t^*, the learner suffers a loss $\ell(y_t, y_t^*)$, measuring the degree of error of the prediction. For example, for a classification task we would use a 0-1 misclassification loss[1] $\ell(y, y') = \mathbf{1}_{[y \neq y']}$, whereas in regression we would use a squared loss $\ell(y, y') = (y - y')^2$.

[1]The notation $\mathbf{1}_{[\cdot]}$ equals 1 if the condition is satisfied, 0 otherwise.

Algorithm 3 The generic online prediction model.

```
1: for $t = 1, \dots, T$ do
2:     Receive input $x_t \in \mathcal{X}$
3:     Predict output $y_t \in \mathcal{Y}$
4:     Observe true output $y_t^*$
5:     Suffer loss $\ell(y_t, y_t^*)$
```

In this learning scenario, we typically fix a hypothesis class $\mathcal{H}$, from which the learner picks, at each iteration $t \in [T]$, a function $h_t \in \mathcal{H}$ to perform the prediction $y_t = h_t(x_t)$. The objective of the learner is to *compete* with the best hypothesis from $\mathcal{H}$. More formally, the learner needs to minimize its *cumulative regret* (or simply *regret*):

$$\sum_{t=1}^{T} \ell(y_t, y_t^*) - \inf_{h \in \mathcal{H}} \sum_{t=1}^{T} \ell(h(x_t), y_t^*) \tag{3.1}$$

The regret measures the cumulative difference between the loss of the predictors chosen by the learner and the best predictor in $\mathcal{H}$. While we seek that our learner achieves the lowest possible regret, a satisfying condition for online learning algorithms is to have regret growing sub-linearly with the number of iterations T. This condition implies that the *average* regret over the iterations $[T]$ approaches 0 for $T \to \infty$.

In the next section, we will introduce one important tool that is extensively used in online learning to derive low regret algorithms, namely *online convex optimization*. Online convex optimization is a generic framework comprising algorithms adaptable to learning when the output space is a convex subset of $\mathbb{R}^d$ and the loss function is convex. In this setting, we will introduce a simple yet powerful algorithm, the *online gradient descent*. This algorithm will then be instantiated to the online classification case in Section 3.1.2, yielding the well known (online) *perceptron* algorithm.

3.1.1 Online convex optimization

Many online learning problems can be cast as *online convex optimization* ones [242]. This framework comprises a set of iterative optimization algorithms for convex functions, whose properties can be analyzed within the same elegant formalism [132]. Most used loss functions in machine learning are convex or can be made convex through randomization or by employing surrogate losses that are convex. Convexity is a fundamental property of many learning problems that allows us to derive *efficient* online learning algorithms.

Algorithm 4 shows the prototypical online convex optimization algorithm. At each iteration $t \in [T]$, the algorithm predicts a vector $\boldsymbol{w}_t$ from a convex set $\mathcal{B} \subseteq \mathbb{R}^d$. The algorithm then receives a convex loss function $\ell_t : \mathcal{B} \to \mathbb{R}$ and suffers the loss $\ell_t(\boldsymbol{w}_t)$. The goal of the

Algorithm 4 The online convex optimization loop.

```
1: for $t = 1, \dots, T$ do
2:     Predict vector $\boldsymbol{w}_t \in \mathcal{B}$
3:     Observe loss $\ell_t : \mathcal{B} \to \mathbb{R}$
4:     Suffer loss $\ell_t(\boldsymbol{w}_t)$
```

algorithm is to minimize the cumulative regret:

$$\sum_{t=1}^{T} \ell_t(\boldsymbol{w}_t) - \inf_{\boldsymbol{w}\in\mathcal{B}} \sum_{t=1}^{T} \ell_t(\boldsymbol{w}) \tag{3.2}$$

To achieve this goal in online convex optimization we make use of the convexity of the loss function. A function $f : \mathcal{B} \to \mathbb{R}$ is convex if and only if for any vector $\boldsymbol{w} \in \mathcal{B}$ there exists a vector $\boldsymbol{z} \in \mathcal{B}$ such that:

$$\forall \boldsymbol{v} \in \mathcal{B} \qquad f(\boldsymbol{w}) - f(\boldsymbol{v}) \leq \langle \boldsymbol{w} - \boldsymbol{v}, \boldsymbol{z} \rangle \tag{3.3}$$

Any vector $\boldsymbol{z}$ satisfying Equation 3.3 is called a *sub-gradient* of f at $\boldsymbol{w}$. We indicate the set of sub-gradients of f at $\boldsymbol{w}$ with $\partial f(\boldsymbol{w})$. In particular, when f is differentiable at $\boldsymbol{w}$, $\partial f(\boldsymbol{w})$ contains a single point that is the gradient $\nabla f(\boldsymbol{w})$. At non-differentiable points, the set $\partial f(\boldsymbol{w})$ may contain an infinite number of vectors.

With the above notions of convexity and sub-gradient, we can go ahead and derive a simple but effective algorithm for online convex optimization, called *online (sub-)gradient descent* [242]. Algorithm 5 shows the simplest version of the algorithm, but many others have been derived over the years. Starting with $\boldsymbol{w}_1 = \boldsymbol{0}$, at each iteration the vector $\boldsymbol{w}_t$ gets updated by taking a step in the opposite direction of a sub-gradient $\boldsymbol{z}_t$ of the loss function ℓ_t at $\boldsymbol{w}_t$. The parameter $\eta \in \mathbb{R}_+$ is a positive constant step-size.

It can be shown (see e.g. [243]) that the regret suffered by the online gradient descent algorithm is upper bounded by:

$$\frac{1}{2\eta}\|\boldsymbol{w}^*\|^2 + \frac{\eta}{2}\sum_{t=1}^{T}\|\boldsymbol{z}_t\|^2 \tag{3.4}$$

where $\boldsymbol{w}^* = \operatorname{argmin}_{\boldsymbol{w}\in\mathcal{B}} \sum_{t=1}^{T} \ell_t(\boldsymbol{w})$.

We can derive a tighter bound in case the loss functions ℓ_t happen to be *Lipschitz*. A function is Lipschitz when it does not change “too fast” at any given point. More formally, a function is Lipschitz over a set $\mathcal{B}$ if there exists $G \in \mathbb{R}_+$ such that for any two vectors $\boldsymbol{w}, \boldsymbol{v} \in \mathcal{B}$ we have $|f(\boldsymbol{w}) - f(\boldsymbol{v})| \leq G\|\boldsymbol{w} - \boldsymbol{v}\|$. If f is both convex and Lipschitz, then we also have that the norm of sub-gradients of f at any point is bounded by G. Therefore, if all the loss functions ℓ_t are Lipschitz with factor G, at all $t \in [T]$, $\|\boldsymbol{z}_t\| \leq G$ for all $\boldsymbol{z}_t \in \partial\ell_t(\boldsymbol{w}_t)$. The regret bound in this case becomes:

$$\frac{1}{2\eta}\|\boldsymbol{w}^*\|^2 + \frac{\eta}{2}TG^2 \tag{3.5}$$

Algorithm 5 The online gradient descent algorithm.

```
1: Initialize $\boldsymbol{w}_1 = \mathbf{0}$
2: for $t = 1, \dots, T$ do
3:     Predict vector $\boldsymbol{w}_t$
4:     Observe loss $\ell_t$
5:     $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t - \eta \boldsymbol{z}_t$   with $\boldsymbol{z}_t \in \partial \ell_t(\boldsymbol{w}_t)$
```

If we also assume the set $\mathcal{B}$ to be enclosed in a ball of radius B, i.e. $\mathcal{B} \subseteq \{\boldsymbol{v} \in \mathbb{R}^d : \|\boldsymbol{v}\| \leq B\}$, and we know the time horizon T beforehand, we can set $\eta = \frac{B}{G\sqrt{T}}$ and obtain:

$$\begin{aligned}
\frac{1}{2\eta}\|\boldsymbol{w}^*\|^2 + \frac{\eta}{2}TG^2 &\leq \frac{1}{2\eta}B^2 + \frac{\eta}{2}TG^2 \\
&\leq \frac{G\sqrt{T}}{2B}B^2 + \frac{B}{2G\sqrt{T}}TG^2 \\
&\leq BG\frac{\sqrt{T}}{2} + BG\frac{\sqrt{T}}{2} \\
&\leq BG\sqrt{T}
\end{aligned}$$

This proves that online gradient descent has sub-linear cumulative regret, and its average regret decreases as $\mathcal{O}(1/\sqrt{T})$. If the time horizon is not known, the so called *doubling trick* can be used to eliminate the dependency of η from T, worsening the regret bound by a constant multiplicative factor [242]. The same can be achieved with a variant of Algorithm 5 using a variable step size η_t and a projection step [292]. Many other variants of this algorithm have been developed, e.g. to derive logarithmic regret bound in the case of strongly convex functions [134] and to deal with limited bandit feedback [106].

3.1.2 Online perceptron

As mentioned, the online convex optimization framework can be used to derive efficient online learning algorithms. We will now instantiate the online gradient descent algorithm in the simple case of online classification. The resulting algorithm is the famous *(online) perceptron* algorithm. The perceptron was first introduced by [223] and then analyzed by [109, 124, 189]. Many variants of this algorithm have been developed, such as the Winnow algorithm [166] and passive-aggressive algorithms [67], all analyzable within the online convex optimization framework. In the online classification setting, we have $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \{-1, 1\}$. We consider a linear model $\langle \boldsymbol{w}, \boldsymbol{x} \rangle$, with parameters $\boldsymbol{w} \in \mathbb{R}^d$ to be estimated by the algorithm, and use the sign of this model $\text{sign}(\langle \boldsymbol{w}, \boldsymbol{x} \rangle)$ as prediction rule. We define a 0-1 misclassification loss of the type:

$$\ell_{\text{0-1}}(y, y^*) = \mathbf{1}_{[y^* y \leq 0]} = \mathbf{1}_{[y^* \langle \boldsymbol{w}, \boldsymbol{x} \rangle \leq 0]} \tag{3.6}$$

In the above loss, when the sign of y^* and $\langle \boldsymbol{w}, \boldsymbol{x} \rangle$ agree, then $y^* \langle \boldsymbol{w}, \boldsymbol{x} \rangle > 0$ and thus the loss is equal to 0. Otherwise, if the sign of y^* and $\langle \boldsymbol{w}, \boldsymbol{x} \rangle$ do not agree, or $\langle \boldsymbol{w}, \boldsymbol{x} \rangle = 0$, the expression $y^* \langle \boldsymbol{w}, \boldsymbol{x} \rangle \leq 0$ and the algorithm incurs in a loss equal to 1.

Algorithm 6 The perceptron algorithm.

```
1: for $t = 1, \dots, T$ do
2:     Receive input $\boldsymbol{x}_t$
3:     Predict output $y_t = \text{sign}(\langle \boldsymbol{w}_t, \boldsymbol{x}_t \rangle)$
4:     Observe true output $y_t^*$
5:     if $y_t^* \langle \boldsymbol{w}_t, \boldsymbol{x}_t \rangle \leq 0$ then
6:         $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t + y_t^* \boldsymbol{x}_t$
7:     else
8:         $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t$
```

The cumulative 0-1 loss over the online predictions counts the number of mistakes the algorithm makes:

$$|\mathcal{M}| = \sum_{t=1}^{T} \ell_{\text{0-1}}(y_t, y_t^*)$$

where $\mathcal{M} = \{t \in [T] : y_t^* \langle \boldsymbol{w}_t, \boldsymbol{x}_t \rangle \leq 0\}$ is the set of iterations $t \in [T]$ in which the algorithm has made a mistake. It is common in the literature to derive a *mistake bound*, instead of a regret bound, when analyzing an online classification algorithm. We will now derive an algorithm for online classification and its associated mistake bound starting from the regret bound of online gradient descent.

To cast this learning problem into an online convex optimization one, we define a sequence of loss functions $\ell_t(\boldsymbol{w})$ to pass to the online gradient descent algorithm:

$$\ell_t(\boldsymbol{w}) = \ell_{\text{0-1}}(\text{sign}(\langle \boldsymbol{w}, \boldsymbol{x}_t \rangle), y_t^*) = \mathbf{1}_{[y_t^* \langle \boldsymbol{w}, \boldsymbol{x}_t \rangle \leq 0]}$$

Unfortunately, the above ℓ_t based on the 0-1 loss are not convex. We will, therefore, resort to deriving a classification algorithm using a *surrogate convex* loss. A loss $\hat{\ell}_t$ is a convex surrogate of ℓ_t if: [i] $\hat{\ell}_t$ is convex; [ii] $\hat{\ell}_t(\boldsymbol{w})$ is an upper bound for $\ell_t(\boldsymbol{w})$, i.e. $\hat{\ell}_t(\boldsymbol{w}) \geq \ell_t(\boldsymbol{w})$, for all $\boldsymbol{w}$ and all $t \in [T]$[2]. To derive the online perceptron algorithm we define the following loss function:

$$\hat{\ell}_t(\boldsymbol{w}) = \begin{cases} 0 & \text{if } t \notin \mathcal{M} \\ |1 - y_t^* \langle \boldsymbol{w}, \boldsymbol{x}_t \rangle|_+ & \text{if } t \in \mathcal{M} \end{cases} \tag{3.7}$$

where $|a|_+ = \max\{0, a\}$ is the hinge function. When the prediction of the algorithm is correct, it receives a loss $\hat{\ell}_t(\boldsymbol{w}) = 0$, which is convex and satisfies $\hat{\ell}_t(\boldsymbol{w}_t) = \ell_t(\boldsymbol{w}_t) = 0$. When the algorithm makes a mistake, instead, it receives the loss $|1 - y_t^* \langle \boldsymbol{w}_t, \boldsymbol{x}_t \rangle|_+$, commonly known as *hinge loss*, which is convex and $\hat{\ell}(\boldsymbol{w}_t) \geq \ell_t(\boldsymbol{w}_t) = 1$.

We can now apply the online gradient descent algorithm. At iteration $t \in [T]$ we pick the following sub-gradient of the above loss:

$$\boldsymbol{z}_t = \begin{cases} 0 & \text{if } y_t^* \langle \boldsymbol{w}_t, \boldsymbol{x}_t \rangle > 0 \\ y_t^* \boldsymbol{x}_t & \text{if } y_t^* \langle \boldsymbol{w}_t, \boldsymbol{x}_t \rangle \leq 0 \end{cases}$$

[2]For the analysis to go through it is sufficient to have $\hat{\ell}_t(\boldsymbol{w}_t) \geq \ell_t(\boldsymbol{w}_t)$.

Resulting in the update rule:

$$\boldsymbol{w}_{t+1} = \begin{cases} \boldsymbol{w}_t & \text{if } y_t^* \langle \boldsymbol{w}_t, \boldsymbol{x}_t \rangle > 0 \\ \boldsymbol{w}_t + \eta y_t^* \boldsymbol{x}_t & \text{if } y_t^* \langle \boldsymbol{w}_t, \boldsymbol{x}_t \rangle \leq 0 \end{cases}$$

Let $\mathcal{M}_t = \{i \in \mathcal{M} : i < t\}$ be the set of iterations in which the algorithm has made a mistake up to iteration $t \in [T]$. We have that:

$$y_t = \text{sign}(\langle \boldsymbol{w}_t, \boldsymbol{x}_t \rangle) = \text{sign}\left(\sum_{i \in \mathcal{M}_t} \eta y_i^* \boldsymbol{x}_i \right) = \text{sign}\left(\sum_{i \in \mathcal{M}_t} y_i^* \boldsymbol{x}_i \right)$$

The last equality holds because the sign of the model does not depend on the value of the positive constant η. This fact yields the update rule of the perceptron algorithm[3] (Algorithm 6).

To derive a mistake bound for the perceptron algorithm, we can use the tools from the previous section. In particular, we can adapt the bound in Equation 3.4 using the loss defined in Equation 3.7:

$$\sum_{t=1}^{T} \hat{\ell}_t(\boldsymbol{w}_t) - \sum_{t=1}^{T} \hat{\ell}_t(\boldsymbol{w}^*) \leq \frac{1}{2\eta}\|\boldsymbol{w}^*\|^2 + \frac{\eta}{2} \sum_{t \in \mathcal{M}} \|y_t^* \boldsymbol{x}_t\|^2$$

Let $R = \max_{t \in [T]} \|x_t\|$. Also, since the loss $\hat{\ell}_t$ is a convex surrogate of the 0-1 loss, we have $\sum_{t=1}^{T} \hat{\ell}_t(\boldsymbol{w}_t) \geq |\mathcal{M}|$. Therefore, the above bound can be rewritten as:

$$|\mathcal{M}| \leq \sum_{t=1}^{T} \hat{\ell}_t(\boldsymbol{w}^*) + \frac{1}{2\eta}\|\boldsymbol{w}^*\|^2 + \frac{\eta}{2} R^2 |\mathcal{M}| \tag{3.8}$$

Since the update rule does not depend on η, it is equivalent to the update rule using any $\eta > 0$ and the above bound still holds. Setting $\eta = \frac{\|\boldsymbol{w}^*\|}{R\sqrt{|\mathcal{M}|}}$ we obtain:

$$\begin{aligned} |\mathcal{M}| &\leq \sum_{t=1}^{T} \hat{\ell}_t(\boldsymbol{w}^*) + \frac{1}{2} R \|\boldsymbol{w}^*\| \sqrt{|\mathcal{M}|} + \frac{1}{2} R \|\boldsymbol{w}^*\| \sqrt{|\mathcal{M}|} \\ &\leq \sum_{t=1}^{T} \hat{\ell}_t(\boldsymbol{w}^*) + R \|\boldsymbol{w}^*\| \sqrt{|\mathcal{M}|} \end{aligned}$$

Rearranging:

$$|\mathcal{M}| - R\|\boldsymbol{w}^*\| \sqrt{|\mathcal{M}|} - \sum_{t=1}^{T} \hat{\ell}_t(\boldsymbol{w}^*) \leq 0$$

Notice that $R\|\boldsymbol{w}^*\| \geq 0$ and $\sum_{t=1}^{T} \hat{\ell}_t(\boldsymbol{w}^*) \geq 0$. The above inequality can be solved for $\sqrt{|\mathcal{M}|}$

[3]In the literature, the perceptron is often derived using the loss $\hat{\ell}(\boldsymbol{w}) = |-y_t^* \langle \boldsymbol{w}, \boldsymbol{x}_t \rangle|_+$ and setting $\eta = 1$. While this approach allows to derive the same algorithm, this loss is not technically a surrogate of the 0-1 misclassification error, and thus it cannot be straightforwardly applied to derive the mistake bound of the perceptron.

by analyzing the roots of the convex parabola $x^2 - bx - c = 0$. In particular, $x = \frac{b}{2} \pm \sqrt{\frac{b^2}{4} + c}$, therefore we have:

$$\sqrt{|\mathcal{M}|} \leq \frac{R}{2}\|\boldsymbol{w}^*\| + \sqrt{\frac{R^2}{4}\|\boldsymbol{w}^*\|^2 + \sum_{t=1}^{T} \hat{\ell}_t(\boldsymbol{w}^*)}$$

By squaring both sides of the inequality, and by the fact that $\sqrt{a+b} \leq \sqrt{a} + \sqrt{b}$ for $a, b \geq 0$, we obtain:

$$\begin{aligned}
|\mathcal{M}| &\leq \frac{R^2}{2}\|\boldsymbol{w}^*\|^2 + R\|\boldsymbol{w}^*\|\sqrt{\frac{R^2}{4}\|\boldsymbol{w}^*\|^2 + \sum_{t=1}^{T} \hat{\ell}_t(\boldsymbol{w}^*)} + \sum_{t=1}^{T} \hat{\ell}_t(\boldsymbol{w}^*) \\
&\leq \frac{R^2}{2}\|\boldsymbol{w}^*\|^2 + R\|\boldsymbol{w}^*\|\sqrt{\frac{R^2}{4}\|\boldsymbol{w}^*\|^2} + R\|\boldsymbol{w}^*\|\sqrt{\sum_{t=1}^{T} \hat{\ell}_t(\boldsymbol{w}^*)} + \sum_{t=1}^{T} \hat{\ell}_t(\boldsymbol{w}^*) \\
&\leq \frac{R^2}{2}\|\boldsymbol{w}^*\|^2 + \frac{R^2}{2}\|\boldsymbol{w}^*\|^2 + R\|\boldsymbol{w}^*\|\sqrt{\sum_{t=1}^{T} \hat{\ell}_t(\boldsymbol{w}^*)} + \sum_{t=1}^{T} \hat{\ell}_t(\boldsymbol{w}^*) \\
&\leq R^2\|\boldsymbol{w}^*\|^2 + R\|\boldsymbol{w}^*\|\sqrt{\sum_{t=1}^{T} \hat{\ell}_t(\boldsymbol{w}^*)} + \sum_{t=1}^{T} \hat{\ell}_t(\boldsymbol{w}^*)
\end{aligned}$$

For linearly separable data, i.e. if exists $\boldsymbol{w}^*$ such that $y_t^*\langle \boldsymbol{w}^*, \boldsymbol{x}_t \rangle > 1$ for all $t \in [T]$, we have $\sum_{t=1}^{T} \hat{\ell}_t(\boldsymbol{w}^*) = 0$, and thus:

$$|\mathcal{M}| \leq R^2\|\boldsymbol{w}^*\|^2 \tag{3.9}$$

3.2 Structured-output prediction

Algorithms for standard multi-label classification and multi-dimensional regression aim at finding a function $f : \mathbb{R}^d \to \mathbb{R}^m$ that is able to predict the right output $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$ given the input $\boldsymbol{x} \in \mathbb{R}^d$, by estimating the model parameters $\boldsymbol{\theta}$ from data. In doing so, one typically assumes that the output variables correlate well with the input variables and that the output variables are *independent* of each other. The latter assumption is not always true, especially when predicting *structured* objects like sets, sequences, matrices, trees or graphs. Typical problems of structured prediction arise in natural language processing when extracting the parse tree of a sentence [253], or in computer vision when segmenting the image into meaningful parts [135]. In these cases, output variables are interdependent and standard discriminative models can not represent nor learn these dependencies. The most well known approaches to structured prediction are *conditional random fields* (CRFs) [158] and large-margin approaches, a.k.a. *structured SVMs* [68, 252, 269]. These two methods share many commonalities (see e.g. [146]), and they can be seen as generalizations to the structured case of binary classification with logistic regression [179] and linear SVMs [65]. In Section 3.2.1 we will concentrate on describing the standard formulation of large-margin structured classifiers.

Even though many algorithms for learning structured models exists, we will focus our analysis on a simple online method proposed by Collins [60, 62] based on a variant of the perceptron algorithm. This method is commonly known as the *structured perceptron.* In Section 3.2.2 we will describe the structured perceptron and derive a mistake bound using the tools from online convex optimization.

3.2.1 Large-margin structured classifiers

The common approach for modeling structured prediction problems is to use a joint input-output model $F(x, y)$, encoding not only the dependency between input and output, but also the dependency within the output variables. The model $F : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ represents the *score* of the input-output pair $(x, y) \in \mathcal{X} \times \mathcal{Y}$, with a higher score meaning a more "compatible" output to a given input. A structured predictor $f(x)$ can then be defined by finding the output object maximizing the score for the given input x:

$$f(x) = \operatorname*{argmax}_{y \in \mathcal{Y}} \; F(x, y)$$

To be able to use a structured predictor, an efficient *inference oracle* capable of solving this maximization problem must be available. For example, when the prediction task involves sequences, the Viterbi algorithm (and variants) is often used [177], which finds the highest score sequence using dynamic programming.

To learn the model F, one common approach is to define it as a linear model in some joint input-output feature space:

$$F(x, y) = \langle \boldsymbol{w}, \boldsymbol{\phi}(x, y) \rangle \tag{3.10}$$

where $\boldsymbol{w} \in \mathbb{R}^d$ is the parameter vector to be learned from data, while $\boldsymbol{\phi} : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^d$ is a feature function, mapping input and output pairs to d-dimensional feature vectors. In the batch setting, when training labels are available prior to the prediction, the most widely adopted method for learning the model in Equation 3.10 is by *structured support vector machines* (SSVMs) [270, 269]. Modern training algorithms for SSVMs include: *cutting-plane* [145], *exponentiated gradient* [61], *stochastic subgradient* [217, 244], and *block-coordinate Frank-Wolfe* [157, 197]. Given a dataset $(x_i, y_i^*)_{i=1}^n$, all the aforementioned algorithms train a structured predictor that minimizes a *structured loss* $\Delta(y_i, y_i^*)$, which measures the "discrepancy" between the prediction y_i and the structured label y_i^*. The structured loss used depends on the type of structures the model predicts. If the task is sequence prediction, for instance, the loss is usually a Hamming distance between y_i and y_i^*. Most of these algorithms can also work in an online fashion, minimizing the loss:

$$\ell_t(\boldsymbol{w}) = \max_{y \in \mathcal{Y}} \left[\Delta(y, y_t^*) - \langle \boldsymbol{w}, \boldsymbol{\phi}(x_t, y_t^*) - \boldsymbol{\phi}(x_t, y) \rangle \right] \tag{3.11}$$

The above loss is often called *structured hinge loss.* Minimizing the structured hinge loss

Algorithm 7 The structured perceptron.

```
1: Initialize $\boldsymbol{w}_1 = \mathbf{0}$
2: for $t = 1, \ldots, T$ do
3:     Receive input $x_t \in \mathcal{X}$
4:     Predict output $y_t = \mathrm{argmax}_{y \in \mathcal{Y}} \langle \boldsymbol{w}_t, \boldsymbol{\phi}(x_t, y) \rangle$
5:     Observe true output $y_t^*$
6:     $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t + \boldsymbol{\phi}(x_t, y_t^*) - \boldsymbol{\phi}(x_t, y_t)$
```

is usually achieved by first solving the so called *loss-augmented inference* problem, setting y_t to: $y_t = \mathrm{argmax}_{y \in \mathcal{Y}}[\Delta(y, y_t^*) - \langle \boldsymbol{w}_t, \boldsymbol{\phi}(x_t, y_t^*) - \boldsymbol{\phi}(x_t, y) \rangle]$. The same maximization oracle is usually capable of solving the standard inference problem and the loss-augmented one. Learning algorithms then update the model by exploiting the gradient of the structured hinge loss. For instance, the algorithm from [217] is based on the projected subgradient method [292], resulting in the update rule: $\boldsymbol{w}_{t+1} \leftarrow \Pi_{\mathcal{B}}(\boldsymbol{w}_t - \eta_t(\boldsymbol{\phi}(x_t, y_t) - \boldsymbol{\phi}(x_t, y_t^*)))$, where $\Pi_{\mathcal{B}}(\cdot)$ projects a vector onto a convex set $\mathcal{B}$, and η_t is a variable step size.

In this thesis we will focus on a much simpler approach, namely the *structured perceptron* [60], which is closely related to the coactive learning algorithms (Section 3.3) that we will employ in our constructive preference elicitation framework. For the structured perceptron, the loss $\Delta(y, y') = \mathbf{1}_{[y \neq y']}$ is simply a 0-1 loss, thereby making it analyzable along the lines of the perceptron algorithm for the binary classification case (Section 3.1.2).

3.2.2 Structured perceptron

The structured perceptron was introduced by [60] as a way for training sequence models, alternative to CRFs [158]. However, it can be used for training models over arbitrary structures, provided a compatible inference oracle. The original formulation of the structured perceptron [60] was mainly thought for training structured predictors in a batch setting. Here we provide an online version of the structured perceptron, in which examples simply come sequentially throughout the iterations, instead of being ready available in a training set. Algorithm 7 shows the pseudocode of this slightly adapted version of the structured perceptron. After initializing the weights $\boldsymbol{w}_1$ to $\mathbf{0}$, the algorithm starts iterating through the examples. At each iteration $t \in [T]$, the algorithm receives an input $x_t \in \mathcal{X}$ and predicts an output object y_t by maximizing the current scoring function $\langle \boldsymbol{w}_t, \boldsymbol{\phi}(x_t, \cdot) \rangle$ over $\mathcal{Y}$. The algorithm then receives a true structured label y_t^*, with which updates the weights $\boldsymbol{w}_{t+1}$ according to the rule in line 6 of Algorithm 7.

The original analysis was given following the standard empirical risk minimization protocol, i.e. bounding training and generalization errors separately. In this section we present an alternative version of the analysis based on the online convex optimization tools. Our approach yields the same result of [60] but follows an argument similar to the one used in Section 3.1.2 for the perceptron in the binary classification case. We are not aware of any other work in the literature proving the mistake bound of the structured perceptron using this technique.

We will focus on the *separable* (or *realizable*) case, as it is similar to the case in which coactive learning algorithms operate (see Section 3.3). We assume there exists a vector $\boldsymbol{w}^* \in \mathbb{R}^d$, with $\|\boldsymbol{w}^*\| = 1$, separating the data points by a constant *margin* $\delta > 0$, i.e.:

$$\forall\, t \in [T],\ \forall\, y \in \mathcal{Y}, y \neq y_t^* \qquad \langle \boldsymbol{w}^*, \boldsymbol{\phi}(x_t, y_t^*) - \boldsymbol{\phi}(x_t, y) \rangle \geq \delta$$

As for the original formulation, we consider a 0-1 misclassification loss (Equation 3.6), where predictions are made using the rule $y_t = \text{argmax}_{y\in\mathcal{Y}} \langle \boldsymbol{w}_t, \boldsymbol{\phi}(x_t, y)\rangle$. Also, as for the standard perceptron, here we use a different loss depending on whether the algorithm has made a mistake or not. If $t \notin \mathcal{M}$, we use $\hat{\ell}_t(\boldsymbol{w}) = 0$, whereas, if $t \in \mathcal{M}$, we use the structured hinge loss (Equation 3.11) as a surrogate loss, with $\Delta(y, y_t^*) = \mathbf{1}_{[y \neq y_t^*]}$. When the algorithm makes a mistake, we have that:

$$y_t = \underset{y\in\mathcal{Y}}{\text{argmax}}\ \langle \boldsymbol{w}_t, \boldsymbol{\phi}(x_t, y)\rangle = \underset{y\in\mathcal{Y}}{\text{argmax}}\ [\mathbf{1}_{[y\neq y_t^*]} - \langle \boldsymbol{w}_t, \boldsymbol{\phi}(x_t, y_t^*) - \boldsymbol{\phi}(x_t, y)\rangle]$$

Therefore, the gradient of the structured hinge loss is $\nabla\hat{\ell}_t(\boldsymbol{w}_t) = -(\boldsymbol{\phi}(x_t, y_t^*) - \boldsymbol{\phi}(x_t, y_t))$, resulting in the following update rule for the online gradient descent:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \eta\,(\boldsymbol{\phi}(x_t, y_t^*) - \boldsymbol{\phi}(x_t, y_t))$$

This rule is also used when the algorithm makes a correct prediction, in which case $y_t = y_t^*$, hence $\boldsymbol{\phi}(x_t, y_t^*) - \boldsymbol{\phi}(x_t, y_t) = \mathbf{0}$ and thus the update becomes $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t$, the same we get by using $\hat{\ell}_t(\boldsymbol{w}_t) = 0$ for $t \notin \mathcal{M}$. With an argument similar to the one used in the online classification case, we can see that the predictions of the structured perceptron are independent of the step size η. If the algorithm does not make a mistake, then the update is $\boldsymbol{\phi}(x_t, y_t^*) - \boldsymbol{\phi}(x_t, y_t) = \mathbf{0}$ and thus the weights at iteration t are equal to:

$$\boldsymbol{w}_t = \eta \sum_{i\in\mathcal{M}_t} (\boldsymbol{\phi}(x_i, y_i^*) - \boldsymbol{\phi}(x_i, y_i))$$

Hence, the prediction of the algorithm equals:

$$y_t = \underset{y\in\mathcal{Y}}{\text{argmax}}\ \langle \boldsymbol{w}_t, \boldsymbol{\phi}(x_t, y)\rangle = \underset{y\in\mathcal{Y}}{\text{argmax}}\ \eta \sum_{i\in\mathcal{M}_t} \langle \boldsymbol{\phi}(x_i, y_i^*) - \boldsymbol{\phi}(x_i, y_i), \boldsymbol{\phi}(x_t, y)\rangle$$

Since η is a positive constant, the solution of the above optimization problem is the same for any value of η. Using the regret bound of online gradient descent (Equation 3.4), we have:

$$\sum_{t=1}^T \hat{\ell}_t(\boldsymbol{w}_t) - \hat{\ell}_t(\boldsymbol{w}^*) \leq \frac{1}{2\eta}\|\boldsymbol{w}^*\|^2 + \frac{\eta}{2}\sum_{t=1}^T \|\boldsymbol{\phi}(x_t, y_t^*) - \boldsymbol{\phi}(x_t, y_t)\|^2$$

Let $\|\boldsymbol{\phi}(x_t, y_t^*) - \boldsymbol{\phi}(x_t, y_t)\| \leq R$. Expanding $\hat{\ell}_t(\boldsymbol{w}_t)$ we get:

$$\sum_{t\in\mathcal{M}} (1 - \langle \boldsymbol{w}_t, \boldsymbol{\phi}(x_t, y_t^*) - \boldsymbol{\phi}(x_t, y_t)\rangle) - \sum_{t=1}^T \hat{\ell}_t(\boldsymbol{w}^*) \leq \frac{1}{2\eta}\|\boldsymbol{w}^*\|^2 + \frac{\eta}{2}\sum_{t\in\mathcal{M}} R^2$$

Rearranging:

$$|\mathcal{M}| \leq \sum_{t \in \mathcal{M}} \langle \boldsymbol{w}_t, \boldsymbol{\phi}(x_t, y_t^*) - \boldsymbol{\phi}(x_t, y_t) \rangle + \sum_{t=1}^{T} \hat{\ell}_t(\boldsymbol{w}^*) + \frac{1}{2\eta}\|\boldsymbol{w}^*\|^2 + \frac{\eta}{2}R^2|\mathcal{M}|$$

Since y_t is the maximizer of $\langle \boldsymbol{w}_t, \boldsymbol{\phi}(x_t, \cdot\,) \rangle$, we have that $\langle \boldsymbol{w}_t, \boldsymbol{\phi}(x_t, y_t^*) - \boldsymbol{\phi}(x_t, y_t) \rangle \leq 0$, making the above equation:

$$|\mathcal{M}| \leq \sum_{t=1}^{T} \hat{\ell}_t(\boldsymbol{w}^*) + \frac{1}{2\eta}\|\boldsymbol{w}^*\|^2 + \frac{\eta}{2}R^2|\mathcal{M}|$$

This equation is identical to Equation 3.8 and is valid for any positive constant η. We can therefore apply a similar argument to the one we used for the analysis of the perceptron for online classification. Substituting $\eta = \frac{\|\boldsymbol{w}^*\|}{R\sqrt{|\mathcal{M}|}}$ and expanding $\hat{\ell}_t(\boldsymbol{w}^*)$ we get:

$$\begin{aligned} |\mathcal{M}| &\leq \sum_{t \in \mathcal{M}} (1 - \langle \boldsymbol{w}^*, \boldsymbol{\phi}(x_t, y_t^*) - \boldsymbol{\phi}(x_t, y_t) \rangle) + \frac{1}{2}R\|\boldsymbol{w}^*\|\sqrt{|\mathcal{M}|} + \frac{1}{2}R\|\boldsymbol{w}^*\|\sqrt{|\mathcal{M}|} \\ &\leq \sum_{t \in \mathcal{M}} (1 - \delta) + R\|\boldsymbol{w}^*\|\sqrt{|\mathcal{M}|} \\ &\leq (1 - \delta)|\mathcal{M}| + R\|\boldsymbol{w}^*\|\sqrt{|\mathcal{M}|} \end{aligned}$$

The second inequality follows from the separability assumption. Rearranging:

$$\delta|\mathcal{M}| - R\|\boldsymbol{w}^*\|\sqrt{|\mathcal{M}|} \leq 0$$

Solving for $\sqrt{|\mathcal{M}|}$ we get:

$$\sqrt{|\mathcal{M}|} \leq \frac{R\|\boldsymbol{w}^*\|}{\delta}$$

Squaring both sides and setting $\|\boldsymbol{w}^*\| = 1$:

$$|\mathcal{M}| \leq \frac{R^2}{\delta^2}$$

As we can see, we obtained the same mistake bound found in the original formulation [60], but starting from the online gradient descent algorithm.

3.3 Coactive learning

Coactive learning [248] is an online structured prediction framework for learning the utility function of a user and provide high quality recommendations. A coactive learning algorithm interacts with the user through *coactive feedback*, i.e. after providing an object y as a recommendation, the algorithm receives from the user another object $\bar{y}$ that is preferred (even slightly) to y by the user. Given this feedback, a coactive learning algorithm can improve its

Algorithm 8 The preference perceptron.

```
1: Initialize $\boldsymbol{w}_1 = \mathbf{0}$
2: for $t = 1, \dots, T$ do
3:     Receive context $x_t$
4:     Recommend object $y_t = \operatorname{argmax}_{y \in \mathcal{Y}} \langle \boldsymbol{w}_t, \boldsymbol{\phi}(x_t, y) \rangle$
5:     Receive feedback $\bar{y}_t$
6:     $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t + \boldsymbol{\phi}(x_t, \bar{y}_t) - \boldsymbol{\phi}(x_t, y_t)$
```

estimate of the user utility function and provide better recommendations in the future. This paradigm can be seen as a cooperation between the user and the system to pursue the common goal of producing high quality recommendations. Objects recommended by coactive learning are *structured*, as those predicted by the structured perceptron seen in the previous section. Indeed, there are many commonalities between coactive learning algorithms and the structured perceptron, with the main difference being that coactive learning algorithms accept *weak* user feedback rather than *optimal* labels. Also, differently from other online learning frameworks (e.g. [160]), coactive learning never observes *cardinal* utilities, which is a requirement for many tasks involving the interaction with a human, such as preference elicitation (see Section 2.1). Coactive learning has much in common with online convex optimization, as most of online convex optimization algorithms can be adapted to the coactive learning case [247].

Coactive learning has been developed extensively along several directions. The original authors extended the framework by deriving coactive learning algorithms for handling arbitrary convex and strongly convex functions, as well as learning with sparsifying multiplicative updates [247]. Other developments include the possibility of learning multi-task decision problems involving multiple users [117] and approximately solving decision problems through local search [116], among others [214, 216, 215]. Applications of coactive learning range from personalizing a search engine results [246], to improving machine translation systems [250] and learning trajectories in robot manipulators [140, 141].

In the next section we will focus on the simplest coactive learning algorithm, called *preference perceptron*, for which we will derive a regret bound using the original derivation from [248], and we will show that the same bound can be derived starting from the online gradient descent algorithm as well.

3.3.1 Preference perceptron

The preference perceptron is the first [246] and most basic coactive learning algorithm. Our constructive derivations (Chapter 5 and 6) will be based on this algorithm, though extensions using more complex coactive learning techniques are rather straightforward.

The preference perceptron is showed in Algorithm 8. The algorithm starts off by initializing the utility parameters to zero, and then loops over a series of T iterations. At each iteration $t \in [T]$, the algorithm first receives an input $x_t \in \mathcal{X}$, which can be thought as the *context* of the subsequent recommendation, much like in contextual bandits [160]. The algorithm then produces an output object $y_t \in \mathcal{Y}$ that is recommended to the user. The recommendation is selected by optimizing the current estimate of the utility function $u_t(x, y) = \langle \boldsymbol{w}_t, \boldsymbol{\phi}(x, y)\rangle$. The algorithm then receives the improvement $\bar{y}_t$ from the user. This object does not have to be constructed by the user explicitly, but it can be created from implicit feedback collected from logged user activity. Indeed, the original formulation of coactive learning was mainly adopted for optimizing rankings from clicks [247]. On the other hand we also advocate the use of coactive learning with explicit user feedback, like the feedback that one would get from a configurator system (see Section 2.2.2), or a combination of both implicit and explicit, as we will see in Chapter 8. After receiving the feedback $\bar{y}_t$, the algorithm obtains the new weights $\boldsymbol{w}_{t+1}$ with an update step similar to the one we have seen in the structured perceptron. The preference perceptron is essentially identical to a structured perceptron, though it exploits the fact that, in the absence of the true gradient, a vector with positive inner product with the gradient in expectation suffices to achieve regret minimization [209, 247].

We now go ahead with the analysis of the preference perceptron algorithm. The analysis will rely on a realizability assumption similar to the one made for the structured perceptron in the previous section. Let $\succ_x$ be the true preference relation of the user given context $x \in \mathcal{X}$. We assume there exists a true utility function $u^*(x, y) = \langle \boldsymbol{w}^*, \boldsymbol{\phi}(x, y)\rangle$ such that:

$$\forall\, x \in \mathcal{X},\, y, y' \in \mathcal{Y} \qquad y \succ_x y' \iff u^*(x, y) > u^*(x, y')$$

This also implies that there must exist an optimal object y^*_x for $\succ_x$ and it must be equal to:

$$\forall\, x \in \mathcal{X} \qquad y^*_x = \operatorname*{argmax}_{y \in \mathcal{Y}}\ u^*(x, y)$$

This in turn means that:

$$\forall\, x \in \mathcal{X},\, y \in \mathcal{Y} \setminus \{y^*_x\} \qquad \langle \boldsymbol{w}^*, \boldsymbol{\phi}(x, y^*_x) - \boldsymbol{\phi}(x, y)\rangle > 0$$

This equates to the separability for the structured perceptron (Section 3.2.2), though here an explicit notion of margin is not needed.

The goal of coactive learning is to recommend high utility objects across the iterations, so its objective is to minimize the following average regret:

$$\textsc{reg}_T = \frac{1}{T}\sum_{t=1}^{T} u^*(x_t, y^*_t) - u^*(x_t, y_t)$$

The above regret provides a measurable quantity indicating the quality of the recommendations, as opposed to the 0-1 mistake loss used in the structured perceptron (Section 3.2.2).

We will prove that the preference perceptron has a $\mathcal{O}(1/\sqrt{T})$ upper bound on its average regret. To prove this bound, we further need a model of the behavior of the user. The authors in [247] propose a boundedly rational model, in which an improvement $\bar{y}_t$ is selected by utility maximization within a neighborhood of the prediction y_t. To quantify the quality of the improvement $\bar{y}_t$, we use the following α-*informative* feedback model [247]:

$$\forall\, t \in [T] \qquad u^*(x_t, \bar{y}_t) - u^*(x_t, y_t) = \alpha\,(u^*(x_t, y_t^*) - u^*(x_t, y_t)) - \xi_t \tag{3.12}$$

with $\alpha \in (0, 1]$ and $\xi_t \in \mathbb{R}$. The α-informative model describes the "margin" by which the utility of the improvement $\bar{y}_t$ should be greater than the utility of the prediction y_t. This margin is quantified by a fraction α of the regret at iteration t, $u^*(x_t, y_t^*) - u^*(x_t, y_t)$. The slack variables ξ_t allow for violations to this rule. Note that the model can describe feedback of any quality, given appropriate values of α and ξ_t. The α-informative feedback model also captures the intuitive notion that, as the quality of the recommended objects increases (i.e. the regret decreases), providing an informative feedback becomes more difficult.

Let $\|\boldsymbol{\phi}(\cdot)\| \leq R$. The following is the original derivation of the regret bound from [247].

Theorem 3.3.1. *For a user with true utility parameters $\boldsymbol{w}^*$, following the α-informative feedback model, the average regret of the preference perceptron is upper bounded by:*

$$\textit{REG}_T \leq \frac{2R\|\boldsymbol{w}^*\|}{\alpha\sqrt{T}} + \frac{1}{\alpha T}\sum_{t=1}^{T}\xi_t \tag{3.13}$$

Proof. We start by expanding the squared norm of the vector $\boldsymbol{w}_{T+1}$:

$$\begin{aligned}
\|\boldsymbol{w}_{T+1}\|^2 &= \langle \boldsymbol{w}_{T+1}, \boldsymbol{w}_{T+1}\rangle \\
&= \langle \boldsymbol{w}_T, \boldsymbol{w}_T\rangle + \langle \boldsymbol{\phi}(x_T, \bar{y}_T) - \boldsymbol{\phi}(x_T, y_T), \boldsymbol{\phi}(x_T, \bar{y}_T) - \boldsymbol{\phi}(x_T, y_T)\rangle \\
&\quad + 2\langle \boldsymbol{w}_T, \boldsymbol{\phi}(x_T, \bar{y}_T) - \boldsymbol{\phi}(x_T, y_T)\rangle
\end{aligned}$$

Since y_T is the maximizer of $\langle \boldsymbol{w}_T, \boldsymbol{\phi}(x_T, \cdot)\rangle$, the last term is null or negative. We can therefore bound the above expression as:

$$\begin{aligned}
\|\boldsymbol{w}_{T+1}\|^2 &\leq \|\boldsymbol{w}_T\|^2 + \|\boldsymbol{\phi}(x_T, \bar{y}_T) - \boldsymbol{\phi}(x_T, y_T)\|^2 \\
&\leq \|\boldsymbol{w}_T\|^2 + 4R^2 \\
&\leq 4R^2T
\end{aligned} \tag{3.14}$$

The second last inequality follows from the assumption that $\|\boldsymbol{\phi}(\cdot)\| \leq R$ and the last inequality is obtained by unrolling over $t \in [T]$, keeping in mind that $\boldsymbol{w}_1 = \mathbf{0}$. Now, applying the Cauchy–Schwarz inequality to the dot product between $\boldsymbol{w}^*$ and $\boldsymbol{w}_{T+1}$ we get:

$$\langle \boldsymbol{w}^*, \boldsymbol{w}_{T+1}\rangle \leq \|\boldsymbol{w}^*\|\|\boldsymbol{w}_{T+1}\| \leq 2R\|\boldsymbol{w}^*\|\sqrt{T} \tag{3.15}$$

We get the second inequality by applying Equation 3.14 to $\|\boldsymbol{w}_{T+1}\| = \sqrt{\|\boldsymbol{w}_{T+1}\|^2}$. Expanding the dot product $\langle \boldsymbol{w}^*, \boldsymbol{w}_{T+1}\rangle$:

$$\begin{aligned}
\langle \boldsymbol{w}^*, \boldsymbol{w}_{T+1}\rangle &= \langle \boldsymbol{w}^*, \boldsymbol{w}_T\rangle + \langle \boldsymbol{w}^*, \boldsymbol{\phi}(x_T, \bar{y}_T) - \boldsymbol{\phi}(x_T, y_T)\rangle \\
&= \sum_{t=1}^{T}\langle \boldsymbol{w}^*, \boldsymbol{\phi}(x_t, \bar{y}_t) - \boldsymbol{\phi}(x_t, y_t)\rangle
\end{aligned} \tag{3.16}$$

Combining Equation 3.15 and 3.16 we obtain:

$$\sum_{t=1}^{T} \langle \boldsymbol{w}^*, \boldsymbol{\phi}(x_t, \bar{y}_t) - \boldsymbol{\phi}(x_t, y_t) \rangle \leq 2R\|\boldsymbol{w}^*\|\sqrt{T} \tag{3.17}$$

Applying the α-informative feedback (Equation 3.12) to the LHS, we get:

$$\alpha \sum_{t=1}^{T} \langle \boldsymbol{w}^*, \boldsymbol{\phi}(x_t, y_t^*) - \boldsymbol{\phi}(x_t, y_t) \rangle - \sum_{t=1}^{T} \xi_t \leq 2R\|\boldsymbol{w}^*\|\sqrt{T}$$

Rearranging:

$$\alpha \sum_{t=1}^{T} \langle \boldsymbol{w}^*, \boldsymbol{\phi}(x_t, y_t^*) - \boldsymbol{\phi}(x_t, y_t) \rangle \leq 2R\|\boldsymbol{w}^*\|\sqrt{T} + \sum_{t=1}^{T} \xi_t$$

Dividing by αT both sides proves the claim. □

The same bound can be derived from the bound of online gradient descent. Using the loss:

$$\hat{\ell}_t(\boldsymbol{w}) = \langle \boldsymbol{w}, \boldsymbol{\phi}(x_t, y_t) - \boldsymbol{\phi}(x_t, \bar{y}_t) \rangle$$

The resulting update rule for online gradient descent is:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \eta \left(\boldsymbol{\phi}(x_t, \bar{y}_t) - \boldsymbol{\phi}(x_t, y_t) \right)$$

As usual η is a positive constant that can be ignored in the actual update rule of the preference perceptron. Plugging the above update rule into the usual bound of online gradient descent results in:

$$\begin{aligned} \sum_{t=1}^{T} \hat{\ell}_t(\boldsymbol{w}_t) - \hat{\ell}_t(\boldsymbol{w}^*) &\leq \frac{1}{2\eta}\|\boldsymbol{w}^*\|^2 + \frac{\eta}{2} \sum_{t=1}^{T} \|\boldsymbol{\phi}(x_t, \bar{y}_t) - \boldsymbol{\phi}(x_t, y_t)\|^2 \\ &\leq \frac{1}{2\eta}\|\boldsymbol{w}^*\|^2 + 2\eta R^2 T \end{aligned}$$

Rearranging and expanding the loss $\hat{\ell}_t(\boldsymbol{w}^*)$:

$$\sum_{t=1}^{T} \langle \boldsymbol{w}^*, \boldsymbol{\phi}(x_t, \bar{y}_t) - \boldsymbol{\phi}(x_t, y_t) \rangle \leq - \sum_{t=1}^{T} \hat{\ell}_t(\boldsymbol{w}_t) + \frac{1}{2\eta}\|\boldsymbol{w}^*\|^2 + 2\eta R^2 T$$

Since y_t is the maximizer of $\langle \boldsymbol{w}_t, \boldsymbol{\phi}(x_t, \cdot\,) \rangle$, the loss $\hat{\ell}_t(\boldsymbol{w}_t) \geq 0$ and thus we can upper bound the above expression by:

$$\sum_{t=1}^{T} \langle \boldsymbol{w}^*, \boldsymbol{\phi}(x_t, \bar{y}_t) + \boldsymbol{\phi}(x_t, y_t) \rangle \leq \frac{1}{2\eta}\|\boldsymbol{w}^*\|^2 + 2\eta R^2 T$$

Setting $\eta = \frac{\|\boldsymbol{w}^*\|}{2R\sqrt{T}}$ we get:

$$\sum_{t=1}^{T} \langle \boldsymbol{w}^*, \boldsymbol{\phi}(x_t, \bar{y}_t) + \boldsymbol{\phi}(x_t, y_t) \rangle \leq R\|\boldsymbol{w}^*\|\sqrt{T} + R\|\boldsymbol{w}^*\|\sqrt{T} = 2R\|\boldsymbol{w}^*\|\sqrt{T}$$

The above inequality is identical to that in Equation 3.17. Applying the inequality of the α-informative feedback (Equation 3.12):

$$\alpha \sum_{t=1}^{T} \langle \boldsymbol{w}^*, \boldsymbol{\phi}(x_t, y_t^*) - \boldsymbol{\phi}(x_t, y_t) \rangle - \sum_{t=1}^{T} \xi_t \leq 2R\|\boldsymbol{w}^*\|\sqrt{T}$$

Rearranging and dividing by αT we obtain the same bound as Theorem 3.3.1:

$$\frac{1}{T} \sum_{t=1}^{T} \langle \boldsymbol{w}^*, \boldsymbol{\phi}(x_t, y_t^*) - \boldsymbol{\phi}(x_t, y_t) \rangle \leq \frac{2R\|\boldsymbol{w}^*\|}{\alpha\sqrt{T}} + \frac{1}{\alpha T} \sum_{t=1}^{T} \xi_t$$

3.4 Summary

In this chapter we have presented the online convex optimization framework [132, 242], and one simple, yet powerful, algorithm: the online gradient descent [242]. We stated the regret bound of this algorithm without proof and used it to derive an online algorithm for binary classification, the perceptron [189, 223]. We have then overviewed structured-output prediction [14], highlighting the structured perceptron [60], an online algorithm for structured prediction, providing an alternative proof to the original one, using the tools from online convex optimization. Finally, we have introduced coactive learning [247], an online structured prediction framework for preference learning. Within coactive learning, we presented the preference perceptron algorithm [246, 248] and replicated the proof for the original $\mathcal{O}(1/\sqrt{T})$ average regret bound. We have also connected back this approach to the online gradient descent method, providing an alternative proof employing the tools from online convex optimization. One remarkable common denominator is the online gradient descent approach, which has proven effective for deriving algorithms to solve three very different, and seemingly unrelated tasks, namely binary classification, structured prediction and coactive learning.

Part II

Methods

Chapter 4

Constructive recommendation

Constructive preference elicitation refers to the problem of learning to synthesize novel objects according to the preference of one or more users. The act of generating new objects from scratch often involves searching over an exponentially (or even infinitely) large instance space while optimizing the desired objective. When the objective being optimized is unknown a priori and needs to be learned from data, the task becomes a *constructive machine learning* one. The goal in a constructive learning problem is to generate one or more instances that exhibit some desired properties, and it is commonly approached by iteratively refining the output as new data comes along. Typical applications of constructive machine learning are de novo molecule and drug design [66, 115, 120, 194], automatic music composition [63, 151, 198], video-game levels generation [171], recipe completion [69], and more [176, 191, 192, 193]. In general, constructive machine learning encompasses different learning techniques, such as probabilistic context free grammars [151], non-negative matrix factorization [69], Monte Carlo sampling over structured spaces e.g. sequences [198] and graphs [66], and deep learning [120]. In this thesis we will focus on one technique in particular, namely *constrained structured prediction* [87, 259], which will be discussed in Section 4.1.

Constructive recommendation is a kind of constructive learning problem in which the objective to optimize is the unknown *utility* function of the user. To learn a utility model of the user preferences, the constructive recommender has to interact with the user in order to collect preference feedback, a process that we call *constructive preference elicitation.* This is analogous to the standard preference elicitation techniques seen in Section 2.1. While there is a large literature in preference elicitation, state-of-the-art techniques are not suitable for large constructive tasks, as will be discussed in Section 4.2. In order to scale to large combinatorial spaces we use instead online structured prediction algorithms. In particular, we use coactive learning [247] (see Section 3.3) as our preference elicitation framework, which offers a number of advantages over standard preference elicitation techniques, as argued in Section 4.2.1 and 4.3. In Section 4.3 we will also highlight differences between constructive recommendation and other types of recommender systems. In Section 4.4, we will detail the research problems associated with our constructive approach, which will then be investigated in Chapter 5 and 6. Finally, in Section 4.5, we will enumerate some of the possible applications of constructive recommendation, two of which will be covered in detail in Chapter 7 and 8.

4.1 Constrained structured prediction

As seen in Section 3.2, to learn structured models, one has to define a domain of structured input and output objects $\mathcal{X}$ and $\mathcal{Y}$, a joint input-output feature map $\boldsymbol{\phi} : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^d$ and needs an oracle capable of solving an inference problem of the type $\mathrm{argmax}_{y \in \mathcal{Y}} \langle \boldsymbol{w}, \boldsymbol{\phi}(x, y) \rangle$, for a given $\boldsymbol{w} \in \mathbb{R}^d$ and $x \in \mathcal{X}$. This paradigm has been traditionally applied only to structures for which an efficient oracle was known. With the advancement of constrained optimization solvers, however, it has become more and more practical to apply this kind of paradigm to generic constrained combinatorial domains. In essence, constrained structured prediction consists in just that: learning structured models over arbitrary constrained combinatorial spaces, taking care of inference problems through off-the-shelf constraint solvers.

In practice, this boils down to encoding the input and output objects $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ with a collection of variables, with the possibility of imposing arbitrary constraints to the feasible space. Depending on the type of variables, constraints and features $\boldsymbol{\phi}(\cdot)$, inference corresponds to a program from different classes of optimization problems [201]. In the simplest case, when the problem contains only real variables and linear features and constraints, inference reduces to a linear program (LP) which is solvable in polynomial time [149]. Many problems are, however, discrete in nature and a relaxed linear program would not find an optimal solution. In these cases, variables taking only integer values are needed, yielding an integer program (IP). Allowing a mixture of real and integer variables, we get a mixed integer program (MIP). Restricting the features and constraints to be linear in the decision variables, the inference problem can be solved by an integer linear program (ILP) or mixed integer linear program (MILP) [238]. In general, we will stick to the MILP case, which is the best supported nowadays, though algorithms and solvers for problems with non-linear constraints or objective functions exist for certain particular cases such as quadratic programming (QP) [290] or second order cone programming (SOCP) [19].

While using LP or ILP solvers for structured inference is not a new concept (see e.g. [190]), the technology for employing them out-of-box in an efficient way is relatively recent. The use of *mixed* programs is also a recent advancement first proposed by [259], which employed an *optimization modulo theories* (OMT) solver [240, 241] with a mix of $\mathcal{LRA}$ and $\mathcal{LIA}$ theories[1]. Another recent progress is the integration of structured predictors and constraint programming languages. One notable example is Pyconstruct [87], which leverages MiniZinc [188], perhaps the most well-known high-level constraint programming language to date. Similarly to other frameworks like MiningZinc [125] for constraint-based mining, Pyconstruct allows us to define the domain of structured learning problems through a unified declarative language. The MiniZinc language is itself independent of the underlying solver, allowing to choose the best solver for each situation. Pyconstruct, together with the possibility of mixing discrete and continuous variables [259], effectively makes structured predictors *programmable*, allowing the definition of complex learning problems over arbitrary struc-

[1]Within the vast literature on satisfiability (SAT) [21], researchers have developed solvers for handling satisfiability of formulas with respect to certain background *theories*, such as the linear arithmetic over reals ($\mathcal{LRA}$) or over integers ($\mathcal{LRI}$), or combinations thereof. This method is called *satisfiability modulo theories* (SMT) [70]. SMT solvers have also been extended to handle optimization problems, yielding *optimization modulo theories* (OMT) [240].

tures [87].

Constrained structured prediction [87, 259] can also be seen as a method to inject prior knowledge into a learning problem, which is a typical characteristic of statistical relational learning (SRL) frameworks [114]. In particular, constrained structured prediction is related to various hybrid probabilistic relational models such as hybrid *Markov logic networks* [222, 280] and hybrid *ProbLog* [72, 130]. We refer to [259] for a detailed comparison of constrained structured prediction techniques with several SRL frameworks.

4.2 Constructive preference elicitation

By pulling together constrained structured prediction and preference elicitation (see Section 2.1), we can devise algorithms for learning utility functions from user feedback over *structured* combinatorial output spaces. We call this methodology *constructive preference elicitation.* Constrained structured prediction serves as a modeling, inference and learning framework, while preference elicitation provides the interaction paradigm, the definition and manipulation of utility functions and their decision theoretic interpretation.

Recall from Section 2.1 that it is usually assumed that the utility function $u(\cdot)$ comes with a certain structure, in order to simplify the elicitation process. In particular, when the utility can be broken down over independent components, we can select recommendation more efficiently and the utility can be elicited more quickly. In multi-attribute utility theory (Section 2.1.2), output objects are generally represented as (and uniquely determined by) feature (or attribute) vectors $\mathbf{y} \in \mathbb{R}^d$ in some d-dimensional feature space. In this case, a typical assumption made on the structure of the utility function is to be additively independent over the features: $u(\mathbf{y}) = \langle \boldsymbol{w}, \mathbf{y} \rangle$. Let us assume now that the vectors $\mathbf{y}$ are vectorial representations of objects y from some arbitrary output space $\mathcal{Y}$ and that there exists a vector-valued function $\boldsymbol{\phi} : \mathcal{Y} \to \mathbb{R}^d$ to extract features from these objects, i.e. such that $\mathbf{y} = \boldsymbol{\phi}(y)$. If $\boldsymbol{\phi}$ is bijective and its inverse $\boldsymbol{\phi}^{-1}$ is known, we can still go through the elicitation process with the same additive utility function by selecting recommendation via optimization of $u(\mathbf{y})$ over $\mathbb{R}^d$ and translating to $\mathcal{Y}$ through $\boldsymbol{\phi}^{-1}$. Alternatively, if we have access to an oracle that is capable of searching for a recommendation directly over $\mathcal{Y}$, we can avoid passing through $\mathbb{R}^d$ entirely. In this case, we can abstract away the dependency of the utility from the feature vectors and define the utility function over $\mathcal{Y}$ directly: $u(y) = \langle \boldsymbol{w}, \boldsymbol{\phi}(y) \rangle$. At this point the feature function $\boldsymbol{\phi}$ is not required to be bijective anymore, and we can still enjoy the benefits of additivity over the features. This approach has also the advantage of providing a easy way to embed contextual information $x \in \mathcal{X}$ into the utility model by simply making the features context-dependent: $u(x, y) = \langle \boldsymbol{w}, \boldsymbol{\phi}(x, y) \rangle$.

This latter form is equivalent to the scoring function typically used in structured-output prediction (see Section 3.2). Indeed, like a scoring function for structured classification, a utility function assigns higher scores to better objects for the given context. The main difference with the structured prediction case is that the scoring function is learned from a dataset of “perfect” input-output pairs. Unfortunately, such dataset is rarely available in constructive preference elicitation, especially for decision problems over exponentially large combinatorial spaces. As in standard preference elicitation, we need to resort to exploit preferential information acquirable via interaction with the user. As seen in Section 2.1.2, multi-attribute utility theory already encompasses elicitation over combinatorial spaces. In practice, though, existing preference elicitation algorithms often fail to scale to truly constructive problems. We can, however, draw insight from them for devising new, more efficient, preference elicitation techniques based on constrained structured prediction that can also be applied to the constructive case. The first method proposed for constructive preference elicitation is the *set-wise max-margin* framework from Teso et al. [257]. The core idea in set-wise max-margin is to use a max-margin learning procedure, similar to structured support-vector machines [269], encoded as a MILP to learn not one but k maximally distant parameter vectors, with the simultaneous selection of the k objects of the next recommendation set. Just like Bayesian preference elicitation alternatives [129, 272], set-wise max-margin elicits and learns from set-wise comparison feedback. Teso et al. [257] demonstrated empirically that set-margin is much more computationally efficient than existing Bayesian techniques while achieving comparable performance. Another constructive approach for learning from choice set feedback is the *choice perceptron* [86] that, differently from set-wise max-margin, separates learning from inference and updates the utility parameters in an online fashion. This online approach has been shown to perform and scale substantially better than set-wise max-margin over large combinatorial problems [86]. The choice perceptron also comes with theoretical convergence guarantees, which are not provided by set-wise max-margin.

Another online structured prediction technique is *coactive learning*, which is a generic learning framework for learning structured predictors from weak user feedback (see Section 3.3). Being based on structured prediction algorithms, coactive learning can be readily applied to constructive preference elicitation. The manipulative feedback used in coactive learning is also very well suited for certain constructive tasks, like product configuration and design, in which a user is able to provide feedback by directly improving the proposed configuration, an approach similar to critiquing systems (see Section 2.2.1), yet without the need of directly modifying the weights of the utility function.

A constructive preference elicitation system based on coactive learning has the following components. As for standard structured prediction techniques, learning and inference are decoupled into two separate software components. The learning part is handled by a coactive learning algorithm, which drives the elicitation through coactive feedback and learns a utility function of the form $u(x, y) = \langle \boldsymbol{w}, \boldsymbol{\phi}(x, y)\rangle$. As in constrained structured prediction, we model the output space $\mathcal{Y}$ with a collection of variables $y_1, \dots y_m$ with domains $\mathcal{Y}_1, \dots, \mathcal{Y}_m$. The variables y_i represent basic attributes of the objects, similarly to other techniques in multi-attribute utility theory. The output objects $y \in \mathcal{Y}$ are tuples $(y_1, \dots, y_m) \in \mathcal{Y}_1 \times \dots \times \mathcal{Y}_m$, combining all the components into one object. The feasible output space $\mathcal{Y}$ can also be limited by a set of *hard* constraints C, $\mathcal{Y} = (\mathcal{Y}_1 \times \dots \times \mathcal{Y}_m) \cap \mathcal{C}$, where $\mathcal{C} = \cap C$. Input contexts $x \in \mathcal{X}$ are represented as variables as well, though they are given constants at any run of the inference procedure. Context variables $x \in \mathcal{X}$ can be used to define features and constraints, making the feasible output space $\mathcal{Y}(x)$ context-dependent as well, though we will usually avoid to explicitly state this dependency for ease of notation.

The feature map $\boldsymbol{\phi}$ is a collection of real functions $\phi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ defined over the context variables and the output variables. Features in constructive preference elicitation are similar to *soft constraints* used in critiquing systems and constraint-based recommenders (see Section 2.2.1). They encode arbitrarily complex *preference criteria* over the attributes of the objects. Features in constructive preference elicitation are, however, strictly more general than standard soft constraints. The weighted CSP used in constraint-based recommenders associate a weight w_i to each soft constraint c_i and maximize the cumulative weight of the satisfied constraints. Each soft constraint contributes either 0 if $y \notin c_i$ or w_i if $y \in c_i$. Boolean features $\phi(x, y) \mapsto \{0, 1\}$ achieve the same result. Features $\phi(x, y)$, however, can also encode more complex *numerical* dependencies, e.g. *cost functions* of the type $\max\{0, y_j - \lambda\}$, which is 0 as long as the attribute y_j is less than λ and increases linearly with y_j when $y_j \geq \lambda$. This allows us to truly encode arbitrary preference criteria [259]. In fact, a utility function of the type $u(x, y) = \langle \boldsymbol{w}, \boldsymbol{\phi}(x, y)\rangle$ can encode *any* utility function over the attributes of the objects x and y (see Section 4.4.2).

In this setting, the inference problem $y = \text{argmax}_{y\in\mathcal{Y}}\langle \boldsymbol{w}, \boldsymbol{\phi}(x, y)\rangle$ becomes a combinatorial optimization problem over the variables $y_1, \ldots, y_m$. As such, we can use any off-the-shelf constraint optimization solver as *inference oracle*. As seen in Section 4.1, depending on the type of constraints and features, the inference problem may be arbitrarily complex. We will typically restrict to MIP or MILP problems, solvable with available software packages like Gecode, Opturion, or Gurobi[2]. As for standard constrained structured prediction, we can use declarative modeling languages like MiniZinc to encode the domain of the decision problem and the inference procedure [87, 188].

The weights of the utility model can be learned by any coactive learning algorithm. Many variants of the preference perceptron algorithm (see Section 3.3) have been proposed. The authors in [247] developed algorithms for handling batch updates, sparse weight learning with multiplicative exponentiated gradient updates (based on [152]), learning from arbitrary convex and strongly convex losses (based on [292] and [133] respectively). Many extensions to the coactive paradigm have been developed as well, e.g. for handling locally optimal inference [116] and learning a balanced global-local utility model from multiple users [117]. All extensions to coactive learning can be seamlessly integrated into our constructive preference elicitation framework.

4.2.1 Contextual preference elicitation

One important distinction between coactive learning and most of the previous work on preference elicitation is its ability to learn from different *contexts*. The importance of contextual information has long been recognized in the recommendation systems literature [1], and preference elicitation algorithms should take it into account too. The key difference between a contextual and a non-contextual model is that the former can *generalize* to unseen contexts,

[2]Gecode: `gecode.org` Opturion: `opturion.com` Gurobi: `gurobi.com`

carrying over the information acquired in one context to similar ones. This also means that contextual models are *reusable* over different instances of the same decision problem.

Contextual variables can hold any kind of additional information, including information about the user. This can be useful to e.g. initialize the weights of the model to a meaningful starting point learned offline from previously collected data of other users. This model would generalize over similar users, providing a good starting point for a new user, speeding up elicitation and reducing the amount of feedback needed from the user.

Standard preference elicitation algorithms are focused on delivering to the user the *optimal configuration* within the feasible space, while learning a good utility model is a secondary issue. Indeed, to find the best instance in the feasible space a good utility model is not strictly necessary, but rather it suffices to learn a model that ranks first the best object, regardless of the ranking of the other non-optimal ones. The goal of online structured prediction, and of coactive learning as an instance, is instead to learn to approximate the *best function* in the hypothesis class, in our case the space of utility functions.

While similar on the surface, finding the best instance in the feasible space is a much different task from learning a good utility model able to generalize across different contexts. The former is often achievable with few user interactions, while the latter usually requires much more data. This difference in goals often determines the decision problems the two techniques are used for. Coactive learning is mainly used to solve contextual decision problems like personalizing search engine results, providing recommendations for frequently accessed items such as movies, and improving predictive tasks with weak user feedback like machine translation [250]. On the other hand, preference elicitation (often used in conjunction with constraint-based recommender systems) is generally more suitable for infrequently purchased items in a non-contextual environment, such as finding the best apartment or car. In this category fall also many constructive tasks such as product configuration. There are, however, constructive preference elicitation tasks that are also contextual, for instance creating novel personalized recipes, planning a trip, selecting a fashionable outfit, or aiding an expert in a design work. We believe (supported by empirical evidence from [85, 86, 93]) that coactive learning is a viable algorithm for both contextual and non-contextual constructive preference elicitation tasks.

4.2.2 Evaluation

The most direct performance measure for any coactive learning algorithm is its *regret* with respect to the user utility u^*. At any iteration $t \in [T]$ of a coactive learning run, the *instantaneous regret* of the algorithm is:

$$\textsc{reg}(x_t, y_t) = \max_{y_t^* \in \mathcal{Y}} u^*(x_t, y_t^*) - u^*(x_t, y_t) \tag{4.1}$$

We will also be interested in the *cumulative regret* $\sum_{t=1}^{T} \text{REG}(x_t, y_t)$, or equivalently, the *average regret* of the algorithm:

$$\text{REG}_T = \frac{1}{T}\sum_{t=1}^{T} \text{REG}(x_t, y_t)$$

Throughout this thesis, we will use the term *regret* to refer to either the instantaneous and the average regret interchangeably, qualifying any ambiguities that might occur.

The instantaneous regret measures the recommendation quality at each iteration separately, while the average regret measures the overall performance of the system up to iteration T. When the decision problem is non-contextual, the goal of the algorithm is to help the user in finding the optimal object in the feasible space, which equates to minimizing the instantaneous regret of the last iteration, no matter how the algorithm performed in the previous ones. When the problem is contextual, instead, the aim of the algorithm is to propose good solutions (if not optimal) for each context, and its performance in doing so is measured by the average regret.

Coactive learning algorithms are designed to minimize the *average* regret. As seen in Section 3.3, the preference perceptron algorithm enjoys a $\mathcal{O}(1/\sqrt{T})$ upper bound on its average regret, making it converge to 0 for $T \to \infty$. This means that coactive learning is well suited for contextual preference elicitation problems. As for most online learning algorithms, vanishing generalization error bounds can be derived for coactive learning algorithms too [250]. This implies that, the "out-of-sample" instantaneous regret (i.e. at iteration $T+1$) decreases in expectation [49, 109], thereby adding support to the claim that coactive learning is a suitable option for non-contextual preference elicitation as well.

While regret bounds provide worst-case theoretical guarantees, it is useful to test the behavior of the algorithm in the average case through *simulations*. Simulations are a widely used tool to showcase the algorithm performance under certain assumptions on the user behavior. It is usually the case that user choices are simulated through a *feedback model*. Commonly used models for choice set feedback, for instance, are the Bradley-Terry [33], Plackett-Luce [175, 208], or the Thurstone-Mosteller [180] models. In coactive learning we typically employ the α-informative feedback model [247] (see Section 3.3). Given a user utility u^* and a parameter α, improvements $\bar{y}_t$ are selected so to satisfy the α-informative model. Random noise is often used to simulate violations ξ_t to the α-informative model. Simulations can be performed with different parametrizations of α and noise level, in order to compare the performance of the algorithm under different levels of user expertise. Simulations are generally performed in batches using many sampled weight vectors $\boldsymbol{w}^*$ and results are averaged over the users, highlighting the average performance of the algorithm over users with different tastes.

While simulations are effective empirical tools to test the algorithm performance, they rely on assumptions that may not hold true in practice. An alternative solution for testing recommender systems is *offline evaluation* over logged data [245]. In principle, it is possible to use an offline evaluation procedure similar to [160, 161]. This approach would require a dataset of logged coactive interactions $(x_i, y_i, \bar{y}_i)_{i=1}^{n}$, collected with a random selection policy (to avoid bias in the performance estimator). The evaluation algorithm would then "re-run" history. For each example x_i, the algorithm would make a prediction $\hat{y}_i$ with the current coactive model. If $\hat{y}_i = \bar{y}_i$ then the algorithm would receive a loss $\ell_i = 0$ and move to the next example. If $\hat{y}_i = y_i$, the algorithm would receive a loss $\ell_i = 1$ and observe the gradient $\phi(x_i, \bar{y}_i) - \phi(x_i, y_i)$. If $\hat{y}_i$ is neither equal to $\bar{y}_i$ or y_i, the example is discarded. Given the set of retained examples $\mathcal{D}$, the average loss $\frac{1}{|\mathcal{D}|}\sum_{i\in\mathcal{D}} \ell_i$ can then be used to assess the performance of the algorithm and compare different models. Being the problem constructive, the retained examples would be extremely rare, and thus it would only work with a large amount of logged data. In a constructive scenario, it is very difficult and expensive to collect such dataset, so this type of evaluation is often impractical.

The third and final evaluation method we consider is through real user experiments. Such experiments, often called A/B tests, are used to compare two different algorithms or models, and collect evidence on their performance over two separate groups of users. This is perhaps the most effective way to test the real performance of a constructive system, yet it is very expensive compared to simulations. As such, simulations will be the main evaluation tool for the algorithms developed in this thesis. However, we will also show in Chapter 8 the results of an empirical analysis performed with human participants over a fully implemented constructive system.

4.3 Detailed related work

In this section we provide an in depth comparison between our constructive preference elicitation framework based on coactive learning and related work in the literature. In particular, we will be comparing with the three closest approaches, which we already surveyed in Chapter 2 and Chapter 3, namely standard (i.e. regret-based and Bayesian) preference elicitation, constraint-based recommendation and other online learning approaches.

4.3.1 Standard preference elicitation

In Section 2.1.4 we discussed the characteristics of state-of-the-art preference elicitation approaches, namely regret-based and Bayesian preference elicitation. Both these approaches are not suitable for constructive tasks for different reasons [85].

Regret-based approaches natively support the representation of the recommendation task as a constrained optimization program and the possibility of encoding configuration tasks implicitly makes it similar to the constraint-based inference of our constructive recommenders. The lack of support for preference reasoning under noisy feedback, however, makes regret based approaches not viable for non-expert users.

Bayesian preference elicitation is arguably the most general way of dealing with an elicitation problem, especially if represented as a full POMDP. Bayesian decision theory provides an elegant framework to deal with the uncertainty of the utility function, uncertainty of the user feedback, and the most sensible criteria to drive the elicitation process. The most prominent problem with Bayesian preference elicitation is its high computational complexity, which limits its applicability to cases in which the decision catalog contains relatively few items, let alone scale to fully constructive scenarios. Approximate sampling techniques that scale linearly with the number of items in the cataloger [272] have been shown to require too much computational overhead, even on relatively small configuration problems [86, 257]. By replacing expensive Bayesian inference with utility estimation via online structured learning, our constructive preference elicitation technique provides performance comparable to Bayesian preference elicitation at a fraction of the computational cost. Bayesian techniques also cannot straightforwardly take into account implicitly-defined constrained problems like regret-based and constructive approaches do.

Our constructive preference elicitation technique is able to combine the benefits of regret-based and Bayesian approaches, while providing a more efficient means of learning. Also, standard preference elicitation techniques are generally not contextual, and do not build models able to generalize across different instances of the same decision problem. Lastly, all the approaches described in this thesis are based on coactive feedback, which was never considered by any other approach in the literature on preference elicitation so far.

4.3.2 Constraint-based recommenders

While recommendation techniques like collaborative and content-based filtering may seem to fulfill the same goal of the preference elicitation techniques described in the previous section on the surface, there is one fundamental difference: preference elicitation techniques aim at finding the *optimal* object for the user, whereas standard recommendation systems only suggest *any* item that the user may like, without a precise notion of optimality. Indeed, most recommendation techniques in the literature do not employ an explicit preference model, they simply represent the user preferences implicitly through the set of liked or rated items. Some techniques do rank items according to some distance metric or reward model which are designed to maximize click-through rate and similar "success" metrics, rather than a decision-theoretic notion of utility. Many constraint-based approaches adhere to this view too: good recommendations are simply those that satisfy the imposed constraints. The first constraint-based systems to take into account aspects from utility theory are those based on MAUT [101] and soft constraints [289]. Pu and Faltings [211] proposed to represent MAUT subutility functions as soft constraints (see Section 2.1.2 and 2.2.1). This allows us to use constraint solvers with soft constraints to filter and rank solutions on the basis of the hard constraints and the utility model, represented as weighted sum of soft constraints. They also advocate the use of the same technique for both catalog-based recommendation and configuration tasks [276]. This is fundamentally identical to what a constructive recommender system does. There are, however, several key differences with [211] and subsequent work in constraint-based recommendation. As mentioned in Section 4.2, the expressiveness of the features used in constructive preference elicitation is greater than soft constraints, allowing to represent more complex trade-offs and preference criteria. Soft constraint-based systems also typically do not directly measure regret and do not provide optimality convergence guarantees. Perhaps the most important difference though is in the fact that most of these techniques do not *learn* the weights of the utility model, but rather require the user to adjust their numerical value by hand, which is too cognitively complex for models with more than a few soft constraints. As seen in Section 2.2.1, Rossi and Sperduti [225, 227, 226] proposed to represent preferences as soft constraints, and to learn their weights through machine learning. Their technique assumed that a dataset of ratings of objects would be available or that could be collected through interaction. Ratings, however, have been shown not to be an appropriate measure of the user tastes due to inconsistency and noise in their collection [8]. This is especially true when recommendations involve complex objects as for constraint-based recommenders. Their technique also did not provide convergence guarantees either.

Interestingly, Pu and Faltings [211] already used the term *constructive* preference elicitation, though intended with a different meaning to that we attach to the same term. While we use the term to align our preference elicitation method with the larger literature in constructive machine learning, they dubbed their methodology “constructive” in reference to the fact that their method enables users to *construct* their preferences while interacting via critiques. This constructive view is advocated by many experts in psychology and behavioral economics [163]. Indeed, in Chapter 6 we will extend our framework through critiquing, thereby making it “constructive” in this sense as well.

4.3.3 Online learning

Our constructive preference elicitation framework is based on coactive learning, which is closely related to other online learning models. As highlighted in [247], coactive learning is at midway between the bandit and the expert settings (see Section 3.1). The online convex optimization model is the continuous relaxation of the prediction with expert advice. Similar relaxations can be obtained for the bandit setting as well. One major difference between coactive learning and online learning in the expert or bandit setting is that the latter observe *cardinal rewards*, whereas coactive learning only observes ordinal information through the implicit ranking $\bar{y}_t \succcurlyeq y_t$ induced by the coactive improvements. As argued in Section 2.1.3, cardinal utilities and cardinal feedback are not well suited for preference elicitation and they have long been replaced by ordinal feedback like pair-wise or set-wise preference comparisons. This makes coactive learning a more suitable candidate for our constructive preference elicitation scenario. Indeed, the multi-armed bandit model is used in many other settings that do involve cardinal feedback, such as network routing, portfolio optimization and online ad placement. A variant of the multi-armed bandit model is also heavily used in recommendation, namely the *contextual bandit* model [160]. Contextual bandits algorithms make online choices with the help of contextual information available prior to the decision. This is similar to coactive learning, though they differ in the fact that arms in coactive learning are *structured* objects and again in the kind of feedback they accept. Contextual bandits collect cardinal feedback, so they are mainly used in settings in which one single contextual bandit model serves recommendation for many users. The context holds, among other information, user features that are useful to generalize recommendations over similar users. The contextual bandits algorithm then learns a model that optimizes some success metric, e.g. click-through rate, based on the feedback of all users.

Compared to contextual bandits and other online learning frameworks, coactive learning is the most suited for constructive preference elicitation, thanks to the fact that it handles structured objects and structured prediction out-of-the-box, and learns from ordinal feedback instead of cardinal rewards. Recall that our objective is to learn a single, optimal, utility model for each user, so it does not suffice to optimize the click-through rate or similar metrics, we instead need to utilize a proper utility model in order to drive the user towards the optimal choice. Coactive learning is also closely related to other online learning techniques like dueling bandits [91, 286], and learning-to-rank approaches [168]. We refer to [247] for more details.

Another online learning approach related to constructive preference elicitation is the *combinatorial multi-armed bandit* framework [58] and its contextual counterpart [212]. In this setting, at each iteration the agent has to choose a *super arm*, i.e. a set of arms, from a combinatorial space of available super arms. This is similar to a constructive scenario in which structured objects from a combinatorial space are recommended. Also in this case, however, the main difference is that the combinatorial bandit setting assumes a cardinal reward, which is not suitable for preference elicitation. We can, however, envision a constructive recommender system based on contextual combinatorial bandits instead of coactive learning for certain tasks that do not require optimal interactive preference elicitation, such as recipes completion, playlist creation, and compound personalized ad placement.

4.4 Research problems

Thanks to the great expressiveness of constraint optimization tools, and the flexibility of coactive learning we can develop solutions to a large variety of constructive preference elicitation problems. There are, however, several research issues to be tackled, especially in regard to the possibility to scale to large constructive problems and the ability to represent complex preference criteria.

4.4.1 Decomposition of the learning task

One critical aspect in an interactive preference elicitation system is the requirement of being near *real time* [128]. To make the interaction run smoothly, recommendations need to be selected very quickly, without latency. Using an online learning framework such as coactive learning for estimating the utility of a user is certainly an advantage in this regard, as learning between iterations boils down to a simple (and fast) gradient update over the weights. In constructive preference elicitation, the bottleneck in performance is the inference oracle, which often prevents us from scaling to large constructive problems whilst keeping the interaction smooth. One solution is to use approximate oracles [82, 93, 116], which may speedup inference and allow scaling to larger problems, at the price of possibly a higher number of iterations. In some cases, it is also possible to retain optimality guarantees [116]. We will be investigating the trade-offs given by this approach in Chapter 7.

Large constructive problems are impractical also from the cognitive perspective, as reasoning and providing feedback over large structures is very difficult for any user, especially non-expert ones. A common strategy in preference elicitation is to elicit the utility one attribute (or small subsets of attributes) at the time. However, this is not straightforwardly applicable to coactive learning as it is. We will describe an approach to approximate coactive learning through a part-wise decomposition of the learning task in Chapter 5. The usefulness of this approach is twofold: [i] inference would be dealt with one part at the time, bringing a potentially *exponential* speedup to each inference call, though requiring more iterations to pass through all the parts; [ii] the per-iteration cognitive effort of the user drops, as only partial feedback is needed. We will also prove that this approach always reaches a local optimum that has bounded approximation error with respect to standard coactive learning over full objects.

4.4.2 Expressiveness of the utility

As mentioned in Section 3.3, coactive learning works under a quite strong realizability assumption. In particular, this realizability assumption implies that, for coactive learning to work, the user should reason with the same (or at least a subset of the) preference criteria encoded in the features of the utility model. In constructive preference elicitation, this is a concern because to accommodate each user one might need to encode a large number of features, and doing so for such complex decision problems may be very expensive from a design perspective. A solution proposed by [48] is to explicitly encode many combinations of attributes as features, while learning a sparse utility model through a sparsity-inducing norm [261] to keep many features to zero weight and reduce inference time. This approach, however, has the drawback that the number of features increases exponentially with the number of attributes, making it viable only for small problems.

In constructive preference elicitation, the utility has the form $u(x, y) = \langle \boldsymbol{w}, \boldsymbol{\phi}(x, y)\rangle$, where $y = (y_1, \ldots, y_m)$ is a tuple of m variables and $\boldsymbol{\phi} : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^d$ is a feature map containing d arbitrary real functions of the m output variables (plus the context variables). The structure we require for the utility poses no restriction on its expressiveness. Multi-attribute utility theory [150] shows that, without loss of generality, a utility function can be decomposed into a sum of subutilities, each dependent on a subset of attributes, for all subsets of attributes:

$$u(\mathbf{y}) = \sum_{I \subseteq Y} u_I(\mathbf{y}_I)$$

In our case, since features are arbitrary real functions of the objects attributes, we can decompose any utility as a weighted sum of features, each dependent on a subset of attributes (both input and output):

$$u(x, y) = \sum_{I \subseteq X} \sum_{J \subseteq Y} w_{I,J} \phi_{I,J}(x_I, y_J)$$

Depending on the complexity of the features $\phi_{I,J}$ themselves, we can encode arbitrarily complex utility functions. While this allow us to potentially take into account the preference criteria of any user, the exponential number of possibly required feature subsumes an impractical computational complexity. Indeed, in a large constructive problem, representing explicitly all possible features becomes unfeasible. A different approach worth mentioning is that of [78, 77] which uses *kernel machines* to implicitly represent an exponential feature space and efficiently learn a high-dimensional preference model using ranking SVMs [144]. This approach was however limited to Boolean attributes only and inference would require to solve either a non-linear constrained optimization problem in the dual model, or a linear constrained optimization problem with exponentially many features in the primal model, which are both impractical for large constructive problems.

An alternative approach, based on many other constraint-based systems in the literature, is to acquire features as *critiques*. In this case, the utility models starts off with few significant features, while eliciting more through the interaction. This keeps inference time low while adding the possibility to customize the utility model depending on the particular preferences of the user. In Chapter 6 we discuss this approach and propose to integrate it within a coactive learning algorithm, eliciting both preferences, through coactive feedback and weight learning, and features, through example-critiquing [255].

4.5 Applications

A constructive recommender is essentially a constraint-based recommendation system, with the interactive component taken care of by constructive preference elicitation. As such, the constructive recommendation paradigm may be applied to any problem a constraint-based recommender would be suitable for. The constructive preference elicitation techniques, however, shine in contexts in which there is a clear notion of *optimality* of the decision to be made. This fact is more prominent in product recommendation for infrequently purchased items, which are typically more complex objects and have a higher price, such as cars, apartments, electronic and domestic appliances, or infrequently accessed items, such as jobs recommendations. These objects also do not generate much useful preference information, which might even be outdated the next time the same user is going to buy a similar object [99].

On the other hand, frequently purchased items, such as groceries, or frequently accessed content, such as music and movies, are generally simpler and less expensive items, that come with large datasets of purchase, ratings and similar preference information. While our technique, just like coactive learning, would be applicable to recommendation of frequently purchased items, it would probably not perform as well as other data-driven techniques like collaborative filtering and content-based recommendation. There are, however, recommendation tasks involving frequently accessed items that exhibit sufficient structure to benefit from a constructive recommendation treatment, such as recipe completion [69], and composing fashionable outfits [143]. These, in particular, may benefit of the contextual nature of our constructive preference elicitation approach.

Infrequently purchased products are often also *configurable*, such as PC configurations, trips and events (e.g. weddings). Product configuration (see Section 2.2.2) is a complex task that involves the *synthesis* of personalized products, based on the requirements of the user. This is the primary task for which constructive recommender have been designed for. Thanks to the integration of constraint-based technology and effective and efficient preference elicitation methodology, constructive recommenders can be successfully employed in recommending product configurations, while interacting with the user to receive preference information, with the goal of eventually arriving to the optimal configuration. In this category falls also *product and service bundling*, which is typically used for insurance policies and banking. In Chapter 8 we will present a fully implemented constructive system for bundling of telecom services.

Another important category of tasks that a constructive recommendation could be useful for are *design* tasks, such as layout and 3D object design. A constructive recommender could learn from the input of a designer, an architect or an engineer, how to automatically fix, adjust, or even create from scratch, complex structures, in order to ease the work of the expert. This may benefit non-expert users as well, which would be able to design proper structures, e.g. for amateurish 3D printing, without the need to learn the complex details involved. One can imagine that a constructive recommender could be seamlessly integrated into a CAD system or similar software to provide suggestions to the designers. The coactive interaction in this case is very well suited, as feedback would consist mostly of direct manipulations of the objects being created. Also in this case, the preference model needs to be contextual in order to be able to learn to generate editing suggestion for different kinds of objects. In Chapter 7 we will explore in detail the application of constructive preference elicitation to layout synthesis tasks.

Constructive preference elicitation essentially solves a combinatorial optimization problem for which the objective is *unknown* and has to be estimated from the interaction with a user. Many classical combinatorial optimization problems may be extended in this way. For instance, in a travelling salesperson problem (TSP), the edges of the graph may be represented through a number of numerical features, and their respective costs would be linear in the features with unknown weights [116]. Based on how the user changes the recommended path, the edges models can be updated in order to have a better estimate of their cost and provide progressively better paths. A similar formulation may also be used for other combinatorial optimization problem such as for preference-based planning, scheduling, allocation or assignment problems.

4.6 Summary

In this chapter we presented our constructive recommendation methodology, which brings together constrained structured prediction and preference elicitation through coactive learning. This approach is strictly more general than standard constraint-based recommendation systems based on soft constraints and is more efficient than state-of-the-art Bayesian preference elicitation techniques. These advantages make constructive recommendation suitable for a large variety of applications, encompassing product configuration, product and service bundling, design aiding systems and generic preference-based combinatorial optimization. The development of constructive layout synthesis systems will be covered in Chapter 7, while Chapter 8 will detail a fully implemented constructive recommender for product and service bundling.

We have also seen that constructive preference elicitation comes with several research problems. Efficiency of constraint solvers is a bottleneck to overcome in order to scale to larger constructive problems, an issue that will be addressed in Chapter 5. Efficient, near real-time, inference while allowing expressive utility models is a non-trivial challenge given the exponential nature of multi-attribute utility functions. We will address this problem in detail in Chapter 6.

Chapter 5

Part-wise domain decomposition

In the basic formulation of constructive preference elicitation, the algorithm has to solve a constraint optimization problem at each iteration to find the best object according to the learned preference model, which is subsequently recommended to the user. This requirement may lead to scalability issues as the domain size increases. The complexity of the constrained problems increases exponentially with growing number of variables, and the longer inference times may become impractical for a real-time user interaction.

As mentioned in Section 4.4.1 and will be further discussed in Chapter 7, it is possible to successfully elicit user preferences while heuristically selecting a suboptimal recommendation at each iteration [82, 93]. This is a viable method for reducing the inference time, at the cost of increased instantaneous regret and slower convergence rate. For very large problems, however, a good trade-off is hard to find, as the recommendation quality degrades to the point that regret reduction by subgradient descent is unattainable [86].

An alternative approach is to decompose the large objects into several "parts" and proceed with eliciting the user preferences by recommending and updating one part at the time [84]. Elicitation over partial configurations has several connections with existing work in preference elicitation with GAI (generalized additive independence) utility models [38, 39, 40]. Indeed, we will show that a linear utility model $\langle \boldsymbol{w}, \boldsymbol{\phi}(x, y)\rangle$ typically used in constructive preference elicitation is a GAI model over the feature subsets induced by the partition. Here we propose an elicitation technique using coactive learning over decomposed objects.

Consider, for instance, a complex preference-based decision problem, such as organizing a road trip. If there is a long sequence of traveling days, several cities, and a large set of activities to choose from, the decision maker might be more inclined to optimize the trip one day at the time, or perhaps shorter or longer periods, depending on the timescale. Another example could be a preference-based design problem, such as selecting the furniture of a hotel. The owner might prefer the hotel to be a luxurious one or a family-oriented one, there would be a budget to cope with and the kind of furniture used may affect the satisfaction of different kinds of customers. Such a complex allocation problem might be better solved floor by floor or even room by room, freeing the user from having to suggest improvements by looking at all rooms and floors at the same time.

In this chapter we detail our part-wise decomposition approach that will lead to the definition of a coactive learning algorithm, dubbed P^3, that is used to optimize a recommended object one part at the time. The two main advantages of this approach are the decrease in cognitive cost for the user and the speedup of the inference procedure. Solving a partial inference problem may be *exponentially* faster than solving the full inference problem. The number of iterations to reach a satisfying solution may indeed increase, but the cognitive cost for the user per iteration is lower as she is only required to improve a partial configuration by manipulating a subset of the object variables.

This technique does not assume complete partitioning of the objects and indeed it is reasonable to assume that there is instead substantial overlapping between parts. In decomposing overlapping parts, however, one needs to "cut" some of the dependencies between parts, thereby loosing information about these dependencies from the user feedback. Unfortunately, this fact prevents us from providing theoretical guarantees of convergence to the optimal solution for the true user utility. This is the price to pay for learning a full utility function with only partial feedback. Despite this shortcoming, we can prove that this technique converges to a "local optimum", for which the user can not further improve any of the parts separately. This local optimum will have bounded approximation error with respect to the global optimum that standard coactive learning over full objects would have found. Furthermore, we will empirically verify that, in many practical cases, the algorithm converges to the globally optimal solution nonetheless.

This chapter is organized as follows. Section 5.1 will introduce a formalism useful for working with parts and partial configurations. Section 5.2 introduces the P^3 algorithm, and Section 5.3 presents its theoretical analysis. The empirical validation is provided in Section 5.4. Section 5.5 concludes the chapter.

5.1 Partitioning the decision problem

In this section we introduce a formal framework for manipulating partial configurations, which we will later use to describe and analyze the P^3 algorithm. The next two sections will describe how to divide the domain of a decision problem into different parts (Section 5.1.1) and how to decompose the utility function into "subutilities" in a principled way (Section 5.1.2).

5.1.1 Parts and partial configurations

As in the standard constructive preference elicitation setting, here we have two sets $\mathcal{X}$ and $\mathcal{Y}$ of input and output objects respectively. Here we further assume that the structure of the output objects can be partitioned into a finite set of *basic parts* $\mathcal{Q}$. The basic parts represent indivisible and non-overlapping portions of the objects $y \in \mathcal{Y}$, which can be composed to form *parts*. A part p is any non-empty subset $p \subseteq \mathcal{Q}$ of the set of basic parts. Each part $p \subseteq \mathcal{Q}$ determines a space of *partial configurations* $\mathcal{Y}_p$ and induces a mapping $y \mapsto y[p]$ between objects $y \in \mathcal{Y}$ and their corresponding partial configurations $y[p] \in \mathcal{Y}_p$. A part $p \subseteq \mathcal{Q}$

can be thought as a portion of the "blueprint" of the objects, whereas a partial configuration $y[p]$ as the instantiation of part p excerpt from object y. Another useful metaphor may be to consider the objects $y \in \mathcal{Y}$ as full jigsaw puzzles with the same splitting patterns, each one with a different picture on top. Following the jigsaw puzzle metaphor, each part $p \subseteq \mathcal{Q}$ would be a portion of the pattern containing pieces that fit well together, while a partial configuration $y[p]$ would be one such portion for a particular picture y. A partition of this kind can easily be found in many tasks, e.g. a partition of a house into rooms for layout synthesis or a segmentation of months into days and weeks for a scheduling problem.

The above domain partitioning $\mathcal{Q}$ has to undergo some additional assumptions in order to be used in our part-wise elicitation process. First, we need to assume that the set $\mathcal{Q}$ of all basic parts covers the entire object, i.e. $y[\mathcal{Q}] = y$ for all $y \in \mathcal{Y}$. Second, we require that in the domain at hand there can be defined a sensible notion of *part combination*, e.g. by combining rooms into floors in layout synthesis or hours into dayparts in scheduling problems. The combination of parts is formally defined with the operator $\circ : \mathcal{Y}_p \times \mathcal{Y}_q \to \mathcal{Y}_{p \cup q}$. Given two non-overlapping parts $p, q \subseteq \mathcal{Q}$, the operation $y[p] \circ y'[q]$ generates a new partial configuration $y''[p \cup q] \in \mathcal{Y}_{p \cup q}$ from the instances of basic parts contained in p and q over y and y' respectively. For non-overlapping parts, the part combination operation is commutative and associative. Given instead two overlapping parts p and q, there are at most $2^{|p \cap q|}$ different ways to combine two distinct partial configurations. To extend the part combination to this case we select deterministically the instances of the overlapping basic parts belonging to the first argument, overloading the part combination operation as $y[p] \circ y'[q] = y[p] \circ y'[p \setminus q]$. In this way, a proper single-valued combination operation is defined even for overlapping parts. Note that this makes part combination not commutative for overlapping parts. Lastly, we define the complement $\tilde{p}$ of a part p as $\tilde{p} = \mathcal{Q} \setminus p$. Combining any two partial configurations $y[p]$ and $y'[\tilde{p}]$ results in a full object $y'' \in \mathcal{Y}$, and in particular the combination of two complementary partial configurations of the same object $y[p] \circ y[\tilde{p}]$ recovers the full object y.

5.1.2 Utility decomposition

In the previous section we introduced the tools needed for partitioning an object into smaller parts. We are now going to define how a utility function over full objects may be decomposed as well. As in standard constructive preference elicitation, we assume the utility to be a linear function of the type $u(x, y) = \langle \boldsymbol{w}, \boldsymbol{\phi}(x, y) \rangle$, with $\boldsymbol{w} \in \mathbb{R}^d$ being the model parameters, and $\boldsymbol{\phi} : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^d$ being a d-dimensional feature map. In this work we have that each part $p \subseteq \mathcal{Q}$ is associated to a non-empty *feature subset* $I(p) \subseteq [d]$ containing the indices of all features dependent on part p. This means that the feature vectors of two objects differing only on part p will differ only on features in $I(p)$. The feature subsets of different parts may overlap, signaling a dependency between them. Disjoint parts may share features as well, i.e. in general $p \cap q = \varnothing \quad \not\Longrightarrow \quad I(p) \cap I(q) = \varnothing$. Finally, we assume that the features associated to the full set of basic parts correspond to the full set of features, $I(\mathcal{Q}) = [d]$.

Given a part $p \subseteq \mathcal{Q}$ with its feature subset $I(p) \in [d]$, we are going to refer to the feature map portion corresponding to features in $I(p)$ as $\boldsymbol{\phi}[I(p)](\cdot) \in \mathbb{R}^{|I(p)|}$. In the same way, we use the notation $\boldsymbol{w}[I(p)] \in \mathbb{R}^{|I(p)|}$ to break down the weight vector $\boldsymbol{w}$ in the components corresponding to $I(p)$. Given the partial feature map and weight vector, for all parts $p \subseteq \mathcal{Q}$, we can define a *partial utility* function as:

$$u[I(p)](x, y) = \langle \boldsymbol{w}[I(p)], \boldsymbol{\phi}[I(p)](x, y) \rangle = \sum_{i \in I(p)} w_i \phi_i(x, y) \tag{5.1}$$

Here w_i and ϕ_i indicate the i-th component of the weight and feature vectors, respectively.

Decomposition of the utility functions is a common tool for devising tractable preference elicitation procedures in the context of multi-attribute utility theory [150]. The typical approach is to assume a certain structure in the way the utility decomposes over the attributes of the object, allowing for easier component-wise elicitation. In particular, as seen in Section 2.1.2 the two most common assumptions found in the literature are *additive independence* between the attributes of the objects, and the weaker *generalized additive independence* (GAI) [12, 39, 103, 121].

Additive independence between the components of a decision problem is achieved if the utility can be decomposed into a sum of *subutility functions*, each dependent on one component and independent of the others. In our case, let $\mathcal{S}$ be a non-empty collection of parts $p \subseteq \mathcal{Q}$ such that: [i] their union covers the whole object, $\bigcup_{p \in \mathcal{S}} p = \mathcal{Q}$; [ii] their feature subsets do not overlap, $\bigcap_{p \in \mathcal{S}} I(p) = \varnothing$. We can define a subutility $u_p(\cdot)$ for each part $p \in \mathcal{S}$ as the partial utility over p, i.e. $u_p(x, y) = u[I_p](x, y)$. To satisfy the additive independence condition we have to have:

$$u(x, y) = \sum_{p \in \mathcal{S}} u_p(x, y) \tag{5.2}$$

It is easy to verify that, with the above construction, the condition in Equation 5.2 holds, thereby leading to an additively independent utility over the parts p:

$$u(x, y) = \sum_{p \in \mathcal{S}} u_p(x, y) = \sum_{p \in \mathcal{S}} u[I(p)](x, y) = \sum_{p \in \mathcal{S}} \langle \boldsymbol{w}[I(p)], \boldsymbol{\phi}[I(p)](x, y) \rangle = \langle \boldsymbol{w}, \boldsymbol{\phi}(x, y) \rangle$$

Since the union of the parts $p \in \mathcal{S}$ covers the entire object, and the feature subsets are non-overlapping, then each feature is added exactly once, hence the last equality is verified. Learning the utility parameters $\boldsymbol{w}$ in the additive independence setting is rather trivial: one might simply use coactive learning on one part at the time and then, since the parts do not share features, combining the resulting optimal partial configurations would yield an optimal object for the full utility function.

A more complicated problem arises in the case of generalized additive independent models, which assume that decomposition is possible over (possibly overlapping) component subsets. In our case, the utility function is GAI if it can be rewritten as a sum of independent subutilities over the parts in collection $\mathcal{S}$, as for Equation 5.2, while allowing the feature subsets $I(p)$ to overlap. As discussed in the previous section, we expect that in many real-world scenarios the feature subsets of the parts $p \in \mathcal{S}$ *do* overlap, and thus the P^3 algorithm has to

Algorithm 9 An ordering strategy for the parts based on a GAI network [121].

```
1: procedure SelectOrdering($\mathcal{S}$)
2:     Build a GAI network $\mathcal{G}$ from $I(p)$ for all $p \in \mathcal{S}$
3:     Produce $\mathcal{P} = (p_1, \dots, p_K)$ by sorting the nodes in $\mathcal{G}$ in ascending order of degree
4:     return $\mathcal{P}$
```

operate in a GAI setting. Therefore, we need to find a decomposition of the utility satisfying Equation 5.2, while still allowing the feature subsets $I(p)$ to overlap. For this purpose, we adapt a construction commonly used in the literature [39, 103]. Let us define an ordering $\mathcal{P} = (p_1, \dots, p_K)$ of the parts $p_k \in \mathcal{S}$ with $k \in [K]$, $K = |\mathcal{S}|$, and let:

$$J_k = I_k \setminus \left(\bigcup_{j=k+1}^{K} I_j \right) \qquad \forall\, k \in [K] \tag{5.3}$$

where $I_k = I(p_k)$. We define the subutilities for the GAI construction as:

$$u_{p_k}(x, y) = u[J_k](x, y) \qquad \forall\, k \in [K] \tag{5.4}$$

The sets J_k defined in Equation 5.3 are all disjoint from one another. By defining the subutilities as in Equation 5.4, the utility $u(x, y)$ satisfies the GAI condition on the feature subsets associated with the parts $p \in \mathcal{P}$. Adding up the subutilities we can see that the full utility function is recovered:

$$u(x, y) = \sum_{k \in [K]} u_{p_k}(x, y) = \sum_{k \in [K]} u[J_k](x, y) = \sum_{k \in [K]} \langle \boldsymbol{w}[J_k], \boldsymbol{\phi}[J_k](x, y) \rangle = \langle \boldsymbol{w}, \boldsymbol{\phi}(x, y) \rangle$$

This decomposition makes the subutilities $u_{p_k}(\cdot)$ independent of one another, allowing us to elicit each one of them separately, as in the additive independence case. The drawback of this approach is that each J_k excludes all the features that are also included in the following subsets $[I_j]_{k+1}^{K}$, i.e. the features $I_k \setminus J_k$. This means that, when eliciting $u_{p_k}(\cdot)$ separately from the others, we do so by ignoring some of the dependencies between the parts. This has the direct consequence that the optimal object found by combining optimal partial configurations for the subutilies may in the end be suboptimal with respect to the full utility function. It is therefore important to choose an ordering of the parts resulting in a minimal number of broken dependencies. This is not a trivial problem, and in general a domain dependent one. However, we propose a simple generic approach that might provide satisfactory results, or at least a good starting point for further refinement.

Algorithm 9 shows a procedure for selecting an ordering $\mathcal{P}$ of the parts $p \in \mathcal{S}$ by leveraging a GAI network [121]. A GAI network is an undirected graph containing one node for each feature subsets $I(p)$ associated to the parts $p \in \mathcal{S}$, and an edge for each pair of feature subsets sharing one or more features. The procedure returns an ordering of the parts $\mathcal{P} = (p_1, \dots, p_K)$ by sorting them in ascending order of node degree, i.e. number of incident edges, of the corresponding nodes in the GAI network. In this way, subsets that share features with many other feature subsets are placed last, in an effort to avoid to break too many dependencies in the above decomposition (Equation 5.3). This is one of many possible approaches, which is here provided as an example and is employed in some of our experiments. More informed techniques could be devised: in particular, this procedure does not take into account the amount of shared features but only the number of other subsets a feature subset shares at least one feature with.

We will now define some of the properties that our utility decomposition satisfies, which will prove useful in the theoretical analysis in Section 5.3. In the following we are going to refer to the union of a range of parts as $\{p_j : p_k\} = p_j \cup \cdots \cup p_k$ and use the shorthand $y[p_j : p_k] = y[p_j \cup \cdots \cup p_k]$. A direct consequence of the utility decomposition in Equation 5.4 is that, for all $y[p_k] \in \mathcal{Y}_{p_k}$, $z[p_1 : p_{k-1}] \in \mathcal{Y}_{\{p_1 : p_{k-1}\}}$ and $z'_1[p_{k+1} : p_K], z'_2[p_{k+1} : p_K] \in \mathcal{Y}_{\{p_{k+1} : p_K\}}$:

$$u[J_k](x, y[p_k] \circ z[p_1 : p_{k-1}] \circ z'_1[p_{k+1} : p_K]) = u[J_k](x, y[p_k] \circ z[p_1 : p_{k-1}] \circ z'_2[p_{k+1} : p_K]) \quad (5.5)$$

meaning that, given the partial configuration $z[p_1 : p_{k-1}]$, then the contribution of $y[p_k]$ to the partial utility $u[J_k](x, \cdot)$ is independent of the rest of the parts $\{p_{k+1} : p_K\}$. This is similar to the *conditional preferential independence* seen in Section 2.1.2, but for $u[J_k](x, \cdot)$ only. Equation 5.5 in turn entails the following *part-wise linearity* property, for all $y[p_k], y'[p_k] \in \mathcal{Y}_{p_k}$ and $z_1[\tilde{p}_k], z_2[\tilde{p}_k] \in \mathcal{Y}_{\tilde{p}_k}$:

$$\begin{aligned} &u[J_k](x, y[p_k] \circ z_1[\tilde{p}_k]) - u[J_k](x, y'[p_k] \circ z_1[\tilde{p}_k]) \\ &= \\ &u[J_k](x, y[p_k] \circ z_2[\tilde{p}_k]) - u[J_k](x, y'[p_k] \circ z_2[\tilde{p}_k]) \end{aligned} \quad (5.6)$$

In other words, the part-wise linearity property affirms that the difference in contribution to the features J_k of any two partial configurations $y[p_k]$ and $y'[p_k]$ is independent of their complement, as long as it is kept fixed.

One last postulate for our formalism is that the feature map $\phi(x, \cdot)$ is defined in such a way to satisfy the following *part-wise addition rule*, for all $k \in [K]$, all $y, y' \in \mathcal{Y}$, and for any $p, q \in \{p_1 : p_k\}$:

$$\begin{aligned} &u[J_k](y[p \cup q] \circ y'[\tilde{p} \cap \tilde{q}]) \\ &\quad = u[J_k](y[p] \circ y'[\tilde{p}]) + u[J_k](y[q] \circ y'[\tilde{q}]) - u[J_k](y[p \cap q] \circ y'[\tilde{p} \cup \tilde{q}]) \end{aligned} \quad (5.7)$$

This is a form of *inclusion-exclusion* principle commonly used in set theory and combinatorics. To visualize the above property one can imagine that to count the contributions of $y[p \cup q]$ we can add up the separate contributions of $y[p]$ and $y[q]$, but doing so we are counting the contributions of their intersection twice[1]. In Section 5.4 we will point out a way to define the feature vector $\phi(x, \cdot)$ so that this property holds.

[1]The same holds true for the complements, which we add together separately, and then by removing their union we recover their intersection.

5.2 Part-wise coactive learning

In this section we detail our part-wise coactive learning method. We are going to proceed in two stages. We will first describe our proposed learning algorithm, named *part-wise preference perceptron* (P^3), for eliciting user preferences over a decomposed domain within a fixed context $x \in \mathcal{X}$ (Algorithm 10). Next, we will illustrate a part-wise coactive learning procedure (PCL) for reusing the same learned model over different contexts (Algorithm 11). The contexts $x \in \mathcal{X}$, in our case, effectively represent different tasks, i.e. variations of the same problem, over the same type of objects, same features and same partition. For instance, in a preference-based scheduling problem, the partition might be over the time (hours, days, weeks), the output objects are assignments of agents to jobs, while the context might represent the level of expertise of the agents for different types of jobs. In this case, learning a utility function over different context means learning the preferences of the user over different combinations of expertise of the agents.

The rationale behind the split of the proposed approach into two separate algorithms is twofold. First, for complex decision problems, we expect that a user would interact with the algorithm over the same object in the same context for a certain number of iterations, before moving on to the next task. Second, the decoupling of the two algorithms makes Algorithm 11 usable with different learning algorithms, besides the one we propose here. Indeed, following the literature in coactive learning [247], one can derive different learning algorithms also in the part-wise case, e.g. for handling arbitrary convex and strongly-convex losses. For the sake of simplicity, we developed our approach by adapting the simpler preference perceptron to the part-wise case, while acknowledging the possibility of extensions along the aforementioned directions.

5.2.1 Part-wise preference perceptron

Algorithm 10 lists the part-wise preference perceptron algorithm, also dubbed P^3. The algorithm expects to receive as input a context $x \in \mathcal{X}$, a initial object $y_0 \in \mathcal{Y}$ and a initial weight vector $\boldsymbol{w}_1 \in \mathbb{R}^d$. If Algorithm 10 is not used in a multi-task environment, the context would be empty, the initial object would be any initial guess, and the weight vector $\boldsymbol{w}_1$ would be initialized to $\mathbf{0}$. The algorithm also takes as input the sequence of parts $\mathcal{P}$ and the time horizon $T \in \mathbb{N}^+$. We include the time horizon explicitly in the algorithm for correctness, but the algorithm could also work with an undefined time horizon, stopping when an optimal solution is found, or when the user is satisfied with the recommended object.

Along with the current estimate of the user utility $u_t(x, y) = \langle \boldsymbol{w}_t, \boldsymbol{\phi}(x, y)\rangle$, the P^3 algorithm keeps a copy of the currently recommended object y_t, starting from y_0, which is iteratively modified one part at the time. At each iteration $t \in [T]$, the algorithm first selects the index k_t of the part it is going to consider next. We indicate the currently selected part as $p_t = p_{k_t}$ with feature subsets $I_t = I_{k_t}$ and $J_t = J_{k_t}$. The part selection is carried out by the SelectPart function. We left this function underspecified, as it could be implemented in a number of ways. The rationale is to use a part selection strategy attempting to find a sequence of parts

Algorithm 10 The part-wise preference perceptron algorithm (with fixed context).

```
1: procedure P$^3$($x \in \mathcal{X}, y_0 \in \mathcal{Y}, \boldsymbol{w}_1 \in \mathbb{R}^d, \mathcal{P} = (p_k)_{k=1}^K, T \in \mathbb{N}^+$)
2:     for $t = 1, \dots, T$ do
3:         $k_t \leftarrow$ SelectPart($\mathcal{P}$)
4:         $p_t \leftarrow p_{k_t} \qquad I_t \leftarrow I_{k_t} \qquad J_t \leftarrow J_{k_t}$
5:         $y_t[\tilde{p}_t] \leftarrow y_{t-1}[\tilde{p}_t]$
6:         $y_t[p_t] \leftarrow \text{argmax}_{y[p_t] \in \mathcal{Y}_{p_t}} u_t[J_t](x, y[p_t] \circ y_t[\tilde{p}_t])$
7:         User provides improvement $\bar{y}_t[p_t]$ of $y_t[p_t]$, keeping $y_t[\tilde{p}_t]$ fixed
8:         if $u_t[I_t](x, \bar{y}_t[p_t] \circ y_t[\tilde{p}_t]) - u_t[I_t](x, y_t) \leq 0$ then $Q_t \leftarrow I_t$ else $Q_t \leftarrow J_t$ ;
9:         $\boldsymbol{w}_{t+1}[\bar{Q_t}] \leftarrow \boldsymbol{w}_t[\bar{Q_t}]$
10:        $\boldsymbol{w}_{t+1}[Q_t] \leftarrow \boldsymbol{w}_t[Q_t] + \boldsymbol{\phi}[Q_t](x, \bar{y}_t[p_t] \circ y_t[\tilde{p}_t]) - \boldsymbol{\phi}[Q_t](x, y_t)$
11:    Recommend $y_T$ to the user
12:    return $\boldsymbol{w}_{T+1}$
```

minimizing the number of iterations needed to reach an overall good solution. For instance, one could prioritize parts with large feature overlap as they are the most difficult to elicit, or use some bandit-based strategy attempting to optimize heuristically the distance from the current object to the optimum. While not explicitly stated in Algorithm 10, the SelectPart function can take as input whichever piece of information is available to the algorithm in order to make the decision.

Next, the algorithm performs inference on the selected part, while keeping the rest of the object $y_t[\tilde{p}_t]$ fixed. Inference is done over the subutility function $u_{p_k}(x, y) = u_t[J_t](x, y)$, which is independent of all the other subutilies, and thus can be elicited separately. We assume we have access to an oracle capable of solving the optimization problem in line 6 of Algorithm 10. When using a mixed integer linear programming solver, for instance, this optimization problem would be the same as the problem on the full object but including a number of equality constraints on the complementary partial configuration $y_t[\tilde{p}_t]$.

Since the complexity of MILP problems is generally exponential in the number of decision variables, solving the problem restricted to the currently selected part only is *exponentially* faster than solving inference on the full object, provided the parts contain a strict subset of the decision variables of the whole object. This is a clear advantage when the decision problem involves many variables. Of course, the drawback is that each part has to be elicited separately, increasing the number of iterations need as well as the number of separate problems to solve. However, as long as the number of iterations remains polynomial in the number of parts K, using this method can be very beneficial in terms of computational complexity.

On the other hand, more iterations also imply more feedback needed from the user. The user, however, only needs to improve one part of the object at the time, which implies less cognitive effort per iteration. Assuming cognitive effort proportional to the size of the part, the increase in number of iterations does not necessarily mean an increase in overall cognitive effort. However, if parts and their features overlap substantially, the net effect might be an increase of the cognitive effort proportional to the amount of overlap between parts.

Another drawback of decomposing a large optimization problem into a sequence of smaller ones is that the solution found by combining them is an *approximation*, i.e. it might be suboptimal with respect to the one the full problem would find. As we will see in Section 5.3.4 of the theoretical analysis, the suboptimality of the solution also depends on the amount of overlapping features between the parts.

Indeed, there is a trade-off to seek between the benefit in computational complexity and the suboptimality of the solution, as well as the amount of additional user effort. In the theoretical analysis we will formalize this trade-off and highlight how the overlap between features plays a crucial role in the ability of the algorithm to converge to a good solution.

After generating the new partial configuration $y_t[p_t]$, the algorithm shows the $y_t[p_t]$ to the user, along with side information needed to make an informed update over it. Side information might be the (fixed) configuration of the parts of the object that share features with the one being improved, or a summary of the common features that can be affected by a change in that part. The user then produces a new partial configuration $\bar{y}_t[p_t]$ improving upon the recommended partial configuration $y_t[p_t]$ over the feature subset I_t. With this information the algorithm proceeds with updating its current estimate of the utility parameters.

Since inference is carried out over J_t, while feedback is expected over I_t, the update schema follows a certain rule (line 8 of Algorithm 10) to decide whether to update the weights corresponding to J_t or I_t. In particular, if the utility $u_t[I_t](x, \bar{y}_t[p_t] \circ y_t[\tilde{p}_t])$ of the improved object is smaller than or equal to the utility $u_t[I_t](x, y_t)$ of the recommended object, than the update will be executed on I_t, otherwise only the features J_t will be updated. The rationale behind this condition, which relates to whether or not the algorithm made a "mistake" according to $u_t[I_t](\cdot)$, will be made clear in the theoretical analysis. The update in line 10 of Algorithm 10 is the standard update of the preference perceptron, but performed only on the features in the subset Q_t chosen according to the above condition. We indicate the complement of Q_t as $Q_t^{-} = [d] \setminus Q_t$. After the update, the algorithm moves on to the next iteration.

As said, the algorithm can either stop after a fixed number of iterations T, or alternatively continue until the user is satisfied the current solution or until the algorithm reaches a "local optimum". We will qualify the notion of local optimum in the analysis section. For readability, we omitted these possible additional stopping criteria from Algorithm 10. Once the loop stops, the algorithm recommends the full object composed by the algorithm so far, and returns the learned utility parameters to the caller.

5.2.2 Handling variable contexts

We will now move on to describe the PCL algorithm for generic part-wise coactive learning with variable contexts. Algorithm 11 shows the PCL algorithm using P^3 as a subroutine. The algorithm accept as input the collection of parts $\mathcal{S}$ over which the decision tasks can be solved. Again, we have the time horizon $T \in \mathbb{N}^+$ as input for correctness, even though we could let each call of P^3 terminate when other stopping criteria are met. The PCL algorithm starts by initializing the initial weights $\boldsymbol{w}^{(1)}$ to zero and then proceeds in "epochs" $n = 1, 2, \ldots$, each

Algorithm 11 The part-wise coactive learning on variable contexts.

```
1: procedure PCL($\mathcal{S}, T \in \mathbb{N}^+$)
2:     $\boldsymbol{w}^{(1)} = \mathbf{0}$
3:     for $n = 1, 2, \dots$ do
4:         Receive context $x^{(n)} \in \mathcal{X}$
5:         Initialize $y^{(n)}$ based on $x^{(n)}$
6:         $\mathcal{P}^{(n)} \leftarrow$ SelectOrdering$(x^{(n)}, \mathcal{S})$
7:         $\bar{\boldsymbol{w}}^{(n)} \leftarrow \mathrm{P}^3(x^{(n)}, y^{(n)}, \boldsymbol{w}^{(n)}, \mathcal{P}, T)$
8:         $\boldsymbol{w}^{(n+1)} \leftarrow \Pi_{\mathcal{B}}\left(\bar{\boldsymbol{w}}^{(n)}\right)$
```

corresponding to a different elicitation task. At each epoch n, the algorithm receives a context $x^{(n)} \in \mathcal{X}$ and initializes an object $y^{(n)}$, e.g. from a catalog of good guesses for each context. The algorithm then selects an ordering $\mathcal{P}$ with the SelectOrdering function on the basis of $\mathcal{S}$ and the context $x^{(n)}$. Depending on the problem, the order of the parts may need to be chosen differently for each separate context. Next, the PCL algorithm calls the P^3 algorithm with the current context $x^{(n)}$, the starting object $y^{(n)}$, the current estimate of the utility weights $\boldsymbol{w}^{(n)}$, the selected ordering $\mathcal{P}^{(n)}$, and the time horizon T. Once P^3 terminates, PCL collects the updated weights $\bar{\boldsymbol{w}}^{(n)}$ and projects them into a ball $\mathcal{B} = \{\boldsymbol{w} \in \mathbb{R}^d : \|\boldsymbol{w}\| \leq B\}$ of radius B, to obtain a new vector $\boldsymbol{w}^{(n+1)}$ with bounded norm. The projection operator is defined as: $\Pi_{\mathcal{B}}(\boldsymbol{w}) = \mathrm{argmin}_{\boldsymbol{v}\in\mathcal{B}} \|\boldsymbol{w} - \boldsymbol{v}\|^2$. The projection step keeps the norm of the weights $\boldsymbol{w}^{(n)}$ bounded across the epochs and guarantees that all the calls to P^3 converge at the same rate, as will be showed in the theoretical analysis.

5.3 Analysis

In this section we will analyze the convergence properties of the P^3 and PCL algorithms. As we mentioned in the previous section, because the P^3 algorithm performs partial inference and receives only partial feedback, the solution found at the end of the elicitation process may be suboptimal with respect to the solution that standard coactive learning would find. Indeed, this fact is also reflected in this analysis, as we will only be able to prove convergence to a "local optimum". We start the analysis (Section 5.3.1) by defining what we mean by local optimum and highlighting the conditions needed to guarantee the convergence of the P^3 algorithm to local optimality, which involve minimizing a quality measure that we call *conditional regret*. Second, in Section 5.3.2, we will prove a sub-linear upper bound on the average conditional regret, for both the non-contextual and contextual cases. In Section 5.3.3, we will then show how, under certain conditions, the P^3 algorithm converges to a locally optimal solution. In Section 5.3.4, we will provide a comparative analysis of the part-wise performance of the local optimum found by P^3 against the global optimum found by standard coactive learning. Finally, in Section 5.3.5 we will recap the analysis and point out differences with the original version of the analysis [84].

5.3.1 Realizability and local optimality

As in standard coactive learning, we assume the user holds a stationary, unobserved true utility function $u^* : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$. Recall from Section 3.3 that in coactive learning we make the realizability assumption:

$$\forall\, x \in \mathcal{X},\ y, y' \in \mathcal{Y} \qquad y \succcurlyeq_x y' \iff u^*(x, y) \geq u^*(x, y') \tag{5.8}$$

for any true preference relation $\succcurlyeq_x$ in context $x \in \mathcal{X}$. Here, instead, we make a "part-wise" realizability assumption, that is:

$$\begin{gathered} \forall\, x \in \mathcal{X},\ p \in \mathcal{P},\ \hat{y}[\tilde{p}] \in \mathcal{Y}_{\tilde{p}},\ y[p], y'[p] \in \mathcal{Y}_p \\ y[p] \circ \hat{y}[\tilde{p}] \succcurlyeq_x y'[p] \circ \hat{y}[\tilde{p}] \iff u^*(x, y[p] \circ \hat{y}[\tilde{p}]) \geq u^*(x, y'[p] \circ \hat{y}[\tilde{p}]) \end{gathered} \tag{5.9}$$

In other words, we assume that the utility $u^*(\cdot)$ is decomposable along the parts $p \in \mathcal{P}$ and is expressive enough to be able to discern two given partial configurations for each part $p \in \mathcal{P}$ in all contexts $x \in \mathcal{X}$ and given any complementary partial configuration $\hat{y}[\tilde{p}]$. As might be expected, this condition is actually *weaker* than the realizable case over full objects (Equation 5.8), as the preference relations $\succcurlyeq_x$ as well as the utility $u^*(x, \cdot)$ only need to be consistent within each part p when the complement $\hat{y}[\tilde{p}]$ is kept fixed. The true utility of the user does not need to be consistent over full objects, which is something that can be expected of a user when faced with a complex decision problem.

Making the part-wise realizability assumption (Equation 5.9) instead of its counterpart on full objects (Equation 5.8) implies that the usual notion of regret over full configurations (Equation 4.1) is ill-posed. If the utility function is inconsistent with the true preferences of the user outside of the single parts $p \in \mathcal{P}$, then finding a global optimum with respect to the full utility $u^*(x, \cdot)$ has no real meaning, as it would be inconsistent with the true preference relation $\succcurlyeq_x$, and therefore not a globally optimal object for the user preferences[2]. What P^3 can do, however, is find a "locally" optimal configuration, as we will prove shortly. This result is in line with previous work on coactive learning with approximate inference [116]. In [116], the authors propose a coactive learning approach for solving intractable combinatorial problems using a locally optimal inference oracle and local user improvements. They prove that the method converges to a local optimum with a sublinear regret bound. Their method, however, still requires (approximate) inference and (local) feedback over full configurations.

This analysis is dedicated to showing that, given a sufficient number of iterations, the P^3 algorithm converges to a configuration that is a local optimum for the true user preferences, according to the following definition.

Definition 5.3.1. An object y^* is *locally optimal* for $u^*(x, \cdot)$ if and only if:

$$\forall\, p \in \mathcal{P} \quad \nexists\, y[p] \in \mathcal{Y}_p \qquad u^*(x, y[p] \circ y^*[\tilde{p}]) > u^*(x, y^*[p] \circ y^*[\tilde{p}]) \tag{5.10}$$

[2]Unless the feature subsets of the parts $p \in \mathcal{P}$ do not overlap, in which case the conditions (5.8) and (5.9) are actually equivalent and the decision problem degenerates into one over additive independent utility.

In other words, y^* is locally optimal if and only if there is no part-wise modification yielding an object y with higher utility than y^*. This means that if an object y^* is locally optimal, the user can not find a part-wise improvement for that object, and neither can the P^3 algorithm.

To prove the local optimality of P^3, we will first define a "local" qualitative measure for the recommended objects, akin to the regret in Equation 4.1 but defined on partial configurations instead. We call this measure the *conditional regret* $\textsc{creg}_p(y)$ of an object $y \in \mathcal{Y}$ with respect to part $p \in \mathcal{P}$. We will later show that P^3 has sublinear cumulative conditional regret in the number of iterations T, a necessary condition for proving convergence to a local optimum.

For the sake of readability, we will henceforth drop the context x from all expressions, as it is constant for any run of Algorithm 10. We will also replace the notation $u^*[I_t](x, y_t[p_t] \circ y_t[\tilde{p}_t])$ in favor of the lighter $u^*[I_t](y_t)$.

Definition 5.3.2. The *conditional regret* of an object $y \in \mathcal{Y}$ over part $p \in \mathcal{P}$ is:

$$\textsc{creg}_p(y) = u^*(y^*[p] \circ y[\tilde{p}]) - u^*(y)$$

where $y^*[p] = \text{argmax}_{y'[p] \in \mathcal{Y}_p} \; u^*(y'[p] \circ y[\tilde{p}])$.

Definition 5.3.3. A configuration y is *conditionally optimal* for part p if $\textsc{creg}_p(y) = 0$.

Lemma 5.3.1. *For a user with utility $u^*(\cdot)$, [A] a configuration y^* is locally optimal for $u^*(\cdot)$ if and only if [B] y^* is conditionally optimal for $u^*(\cdot)$ all parts $p \in \mathcal{P}$.*

Proof. The proof proceeds by contradiction.

[A $\implies$ B] Suppose y^* is locally optimal but not conditionally optimal for some part $p \in \mathcal{P}$. This implies that the conditional regret $\textsc{creg}_p(y^*) > 0$, and thus there exists an object y such that $u^*(y[p] \circ y^*[\tilde{p}]) > u^*(y^*)$. This, however, violates the local optimality condition (Equation 5.10), leading to a contradiction.

[B $\implies$ A] Assume y^* is conditionally optimal for all parts $p \in \mathcal{P}$ but not locally optimal. Then there exist a part $q \in \mathcal{P}$ and a partial configuration $y[q]$ such that $u^*(y[q] \circ y^*[\tilde{q}]) > u^*(y^*)$. This, however, implies that $\textsc{creg}_q(y^*) > 0$, which contradicts the assumption of conditional optimality with respect to all basic parts. □

The above lemma provides a measurable criterion for local optimality: by minimizing the conditional regret with respect to all the parts, the algorithm will eventually converge to a conditionally optimal configuration with respect to all parts, which is consequently also locally optimal. The following section of the analysis is devoted to prove that the P^3 algorithm enjoys a $\mathcal{O}(1/\sqrt{T})$ upper bound on its *average* conditional regret $\frac{1}{T}\sum_{t=1}^{T} \textsc{creg}_{p_t}(y_t)$, which implies convergence to conditional optimality with respect to all basic parts for $T \to \infty$.

Under the part-wise realizability assumption, we will be satisfied with finding a local optimum object. In the case of full realizability, instead, we will show (Section 5.3.4) that the local optimum y^* found by P^3 is competitive with its best improvement over $u^*(\cdot)$, with bounded approximation error.

5.3.2 Conditional regret bound

In this section we derive an upper bound on the conditional regret of the P^3 algorithm. We start by proving a bound for P^3 for the first epoch of PCL, which is also equivalent to the non-contextual case. We will then derive another bound, with the same convergence rate, for P^3 for epoch number $n > 1$. All the definitions and derivations below apply to any run of P^3 regardless of the epoch, so we will generally omit the epoch number superscript for readability.

To derive the bound, we will rely on a part-wise feedback model, adapted from the standard α-informative model, quantifying the amount of information obtained from the partial improvements provided by the user.

Definition 5.3.4. The feedback of a user with utility $u^*(\cdot)$ is *partially* α-informative if, for any selected part $p_t \in \mathcal{P}$:

$$u^*[I_t](\bar{y}_t[p_t] \circ y_t[\tilde{p}_t]) - u^*[I_t](y_t) = \alpha\big(u^*[I_t](y^*_t[p_t] \circ y_t[\tilde{p}_t]) - u^*[I_t](y_t)\big) - \xi_t \tag{5.11}$$

In other words, the user feedback is partially α-informative if, given a recommended partial configuration $y_t[p_t]$, the partial improvement $\bar{y}_t[p_t]$ has a partial utility higher than $y_t[p_t]$ of at least a fraction $\alpha \in (0, 1]$ of the conditional regret, modulo a slack $\xi_t \in \mathbb{R}$. The partial α-informative feedback quantifies the *partial utility gain* (LHS of Equation 5.11) over the feature subset I_t corresponding to part p_t. Intuitively, this is a reflection of the fact that the user can potentially modify any of the features associated to the part p_t when providing the improvement $\bar{y}_t[p_t]$.

Since part p_t only affects features in I_t, under the partially α-informative feedback model, the improvement $\bar{y}_t[p_t]$ satisfies $u^*[I_t](\bar{y}_t[p_t] \circ y_t[\tilde{p}_t]) > u^*[I_t](y_t) - \xi_t$, for $\bar{y}_t[p_t] \neq y_t[p_t]$. This means that, aside from the noise ξ_t in the user feedback, the algorithm mistakenly recommended y_t in place of the better solution $\bar{y}_t[p_t] \circ y_t[\tilde{p}_t]$, and thus the weights $\boldsymbol{w}_{t+1}[I_t]$ should be adjusted to avoid this mistake in the future: $u_{t+1}[I_t](\bar{y}_t[p_t] \circ y_t[\tilde{p}_t]) > u_{t+1}[I_t](y_t)$. However, since inference is done over J_t and not over I_t, it might happen that the above condition is already satisfied at iteration t and the algorithm did not actually make a mistake according to the partial utility $u_t[I_t](\cdot)$. When this happens, the only safe update the algorithm can perform on the weights is over the features J_t, because it is always the case that $u_t[J_t](\bar{y}_t[p_t] \circ y_t[\tilde{p}_t]) \leq u_t[J_t](y_t)$, given that $y_t[p_t]$ is the maximizer of $u_t[J_t](\,\cdot\, \circ\, y_t[\tilde{p}_t])$. Therefore, the choice of the subset of weights to be updated is between I_t and J_t, and the chosen subset Q_t is selected according to the condition in line 8 of Algorithm 10. When $Q_t = J_t$, there is a mismatch between the subset of updated weights and the subset of weights according to which the user selects the improvement. While needed to ensure coherence of the partial utility with respect to the partial configurations $y_t[p_t]$ and $\bar{y}_t[p_t]$, this uneven update schema can make the algorithm “miss” some of the utility gain from the user feedback, thereby slowing down the convergence rate of the algorithm. This fact will be quantified in conditional regret bound by the average of the terms $[\zeta_t]_{t=1}^T$, which we will introduce next.

Let us distinguish two sets of iterations based on the condition in line 8 of Algorithm 10:

$$\mathcal{I}_T = \{t \in [T] : u_t[I_t](\bar{y}_t[p_t] \circ y_t[\tilde{p}_t]) - u_t[I_t](y_t) \leq 0\}$$
$$\mathcal{J}_T = \{t \in [T] : u_t[I_t](\bar{y}_t[p_t] \circ y_t[\tilde{p}_t]) - u_t[I_t](y_t) > 0\}$$

If $t \in \mathcal{I}_T$ then $Q_t = I_t$, whereas if $t \in \mathcal{J}_T$ then $Q_t = J_t$. Considering the dot product between $\boldsymbol{w}^*$ and $\boldsymbol{w}_{t+1}$ at each iteration $t \in [T]$, we have:

$$\begin{aligned}
\langle \boldsymbol{w}^*, \boldsymbol{w}_{t+1} \rangle &= \langle \boldsymbol{w}^*[Q_t], \boldsymbol{w}_{t+1}[Q_t] \rangle + \langle \boldsymbol{w}^*[\bar{Q_t}], \boldsymbol{w}_{t+1}[\bar{Q_t}] \rangle \\
&= \langle \boldsymbol{w}^*[Q_t], \boldsymbol{w}_t[Q_t] \rangle + \langle \boldsymbol{w}^*[\bar{Q_t}], \boldsymbol{w}_t[\bar{Q_t}] \rangle \\
&\quad + \langle \boldsymbol{w}^*[Q_t], \boldsymbol{\phi}[Q_t](\bar{y}_t[p_t] \circ y_t[\tilde{p}_t]) - \boldsymbol{\phi}[Q_t](y_t) \rangle \\
&= \langle \boldsymbol{w}^*, \boldsymbol{w}_t \rangle + \langle \boldsymbol{w}^*[Q_t], \boldsymbol{\phi}[Q_t](\bar{y}_t[p_t] \circ y_t[\tilde{p}_t]) - \boldsymbol{\phi}[Q_t](y_t) \rangle \\
&= \langle \boldsymbol{w}^*, \boldsymbol{w}_t \rangle + \big(u^*[Q_t](\bar{y}_t[p_t] \circ y_t[\tilde{p}_t]) - u^*[Q_t](y_t[p_t])\big)
\end{aligned} \tag{5.12}$$

We call the second summand of the last equality the *utility step*, which is the quantity the estimate $\boldsymbol{w}_t$ moves at iteration t with respect to $\boldsymbol{w}^*$. If this quantity is positive, $\boldsymbol{w}_{t+1}$ will move closer to $\boldsymbol{w}^*$, whereas if it is negative, $\boldsymbol{w}_{t+1}$ will move apart from $\boldsymbol{w}^*$.

When $t \in \mathcal{I}_T$, the utility step equals the partial utility gain:

$$u^*[I_t](\bar{y}_t[p_t] \circ y_t[\tilde{p}_t]) - u^*[I_t](y_t) \tag{5.13}$$

which directly follows the partial α-informative feedback (Equation 5.11).

When $t \in \mathcal{J}_T$, instead, the utility step reduces to:

$$u^*[J_t](\bar{y}_t[p_t] \circ y_t[\tilde{p}_t]) - u^*[J_t](y_t)$$

which, however, can be rewritten in terms of partial utility gain over I_t as:

$$\big(u^*[I_t](\bar{y}_t[p_t] \circ y_t[\tilde{p}_t]) - u^*[I_t](y_t)\big) - \big(u^*[I_t \setminus J_t](\bar{y}_t[p_t] \circ y_t[\tilde{p}_t]) - u^*[I_t \setminus J_t](y_t)\big) \tag{5.14}$$

Combining Equation 5.13 and Equation 5.14, we can compactly re-define the utility step for all iterations $t \in [T]$ as:

$$u^*[I_t](\bar{y}_t[p_t] \circ y_t[\tilde{p}_t]) - u^*[I_t](y_t) - \zeta_t \tag{5.15}$$

where

$$\zeta_t = \begin{cases} 0 & \text{if } t \in \mathcal{I}_T \\ u^*[I_t \setminus J_t](p_t[\bar{y}_t] \circ y_t[\tilde{p}_t]) - u^*[I_t \setminus J_t](y_t) & \text{if } t \in \mathcal{J}_T \end{cases} \tag{5.16}$$

In Equation 5.15 we have rephrased the utility step into the difference between the partial utility gain and a quantity ζ_t, which we call the *utility leak* at iteration t. Note that the utility leak ζ_t can be positive, negative or null. If the leak is null or negative, updating the weights only on J_t is actually beneficial, avoiding a step back or even increasing the step in utility.

Finally, let us assume that $\|\boldsymbol{\phi}(y)\|_\infty \leq D$ and define $S = \max_{p\in\mathcal{P}} |I(p)|$. We are now ready to state and prove the bound on the average conditional regret of the P^3 algorithm in the stand-alone case or within the execution of PCL for $n = 1$.

Theorem 5.3.2. *For a user with true utility parameters $\boldsymbol{w}^*$, under the partial α-informative feedback assumption, the average conditional regret incurred by the P^3 algorithm is upper bounded by:*

$$\frac{1}{T}\sum_{t=1}^{T} \textit{CREG}_{p_t}(y_t) \leq \frac{2DS\|\boldsymbol{w}^*\|}{\alpha\sqrt{T}} + \frac{1}{\alpha T}\sum_{t=1}^{T}(\xi_t + \zeta_t) \tag{5.17}$$

Proof. The proof proceeds by first bounding the term $\|\boldsymbol{w}_{T+1}\|$ for both $T \in \mathcal{I}_T$ and $T \in \mathcal{J}_T$. We will then expand and bound the term $\langle \boldsymbol{w}^*, \boldsymbol{w}_{T+1}\rangle$ using the Cauchy-Schwarz inequality. Finally, we apply the partial α-informative feedback assumption to the resulting expression, yielding the final statement.

Let us begin by expanding the term $\|\boldsymbol{w}_{T+1}\|^2$ in the general case:

$$\begin{aligned}
\|\boldsymbol{w}_{T+1}\|^2 &= \|\boldsymbol{w}_{T+1}[Q_T^{\text{-}}]\|^2 + \|\boldsymbol{w}_{T+1}[Q_T]\|^2 \\
&= \|\boldsymbol{w}_T[Q_T^{\text{-}}]\|^2 + \|\boldsymbol{w}_T[Q_T] + \boldsymbol{\phi}[Q_T](\bar{y}_T[p_T] \circ y_T[\tilde{p}_T]) - \boldsymbol{\phi}[Q_T](y_T)\|^2 \\
&= \|\boldsymbol{w}_T[Q_T^{\text{-}}]\|^2 + \|\boldsymbol{w}_T[Q_T]\|^2 + \|\boldsymbol{\phi}[Q_T](\bar{y}_T[p_T] \circ y_T[\tilde{p}_T]) - \boldsymbol{\phi}[Q_T](y_T)\|^2 \\
&\quad + 2\langle \boldsymbol{w}_T[Q_T], \boldsymbol{\phi}[Q_T](\bar{y}_T[p_T] \circ y_T[\tilde{p}_T]) - \boldsymbol{\phi}[Q_T](y_T)\rangle \\
&\leq \|\boldsymbol{w}_T\|^2 + \|\boldsymbol{\phi}[Q_T](\bar{y}_T[p_T] \circ y_T[\tilde{p}_T]) - \boldsymbol{\phi}[Q_T](y_T)\|_\infty^2 |Q_T|^2 \\
&\quad + 2\big(u_T[Q_T](\bar{y}_T[p_T] \circ y_T[\tilde{p}_T]) - u_T[Q_T](y_T)\big)
\end{aligned}$$

The last inequality follows from the fact that $\|\boldsymbol{z}\| \leq d\|\boldsymbol{z}\|_\infty$ for $\boldsymbol{z} \in \mathbb{R}^d$.

If $T \in \mathcal{I}_T$, then $Q_T = I_T$ and thus we have:

$$u_T[I_T](\bar{y}_T[p_T] \circ y_T[\tilde{p}_T]) - u_T[I_T](y_T) \leq 0$$

Hence, the above inequality reduces to:

$$\begin{aligned}
\|\boldsymbol{w}_{T+1}\|^2 &\leq \|\boldsymbol{w}_T\|^2 + \|\boldsymbol{\phi}[I_T](\bar{y}_T[p_T] \circ y_T[\tilde{p}_T]) - \boldsymbol{\phi}[I_T](y_T)\|_\infty^2 |I_T|^2 \\
&\leq \|\boldsymbol{w}_T\|^2 + 4D^2S^2
\end{aligned}$$

If $T \in \mathcal{J}_T$, instead, by the optimality of y_T with respect to $u_T[J_T]$ we always have that:

$$u_T[J_T](\bar{y}_T[p_T] \circ y_T[\tilde{p}_T]) - u_T[J_T](y_T) \leq 0$$

Thus the above inequality becomes:

$$\begin{aligned}
\|\boldsymbol{w}_{T+1}\|^2 &\leq \|\boldsymbol{w}_T\|^2 + \|\boldsymbol{\phi}[J_T](\bar{y}_T[p_T] \circ y_T[\tilde{p}_T]) - \boldsymbol{\phi}[J_T](y_T)\|_\infty^2 |J_T|^2 \\
&\leq \|\boldsymbol{w}_T\|^2 + 4D^2S^2
\end{aligned}$$

where the last inequality follows from $J_T \subseteq I_T$, therefore $|J_T| \leq |I_T| \leq S$.

As the two inequalities for $T \in \mathcal{I}_T$ and $T \in \mathcal{J}_T$ coincide, we can unroll the above expression to obtain:

$$\|\boldsymbol{w}_{T+1}\|^2 \leq 4D^2S^2T \tag{5.18}$$

Here we used the fact that $\boldsymbol{w}_1 = \mathbf{0}$, thus $\|\boldsymbol{w}_1\|^2 = 0$. Now, applying the Cauchy-Schwarz inequality we have:

$$\langle \boldsymbol{w}^*, \boldsymbol{w}_{T+1} \rangle \leq \|\boldsymbol{w}^*\| \|\boldsymbol{w}_{T+1}\| \leq 2DS\|\boldsymbol{w}^*\|\sqrt{T} \tag{5.19}$$

From the derivation in Equation 5.12 and the utility step in Equation 5.15 we have:

$$\langle \boldsymbol{w}^*, \boldsymbol{w}_{T+1} \rangle = \langle \boldsymbol{w}^*, \boldsymbol{w}_T \rangle + \big(u^*[I_T](\bar{y}_T[p_T] \circ y_T[\tilde{p}_T]) - u^*[I_T](y_T)\big) - \zeta_T \tag{5.20}$$

Expanding the above equation we obtain:

$$\langle \boldsymbol{w}^*, \boldsymbol{w}_{T+1} \rangle = \sum_{t=1}^{T} \big(u^*[I_t](\bar{y}_t[p_t] \circ y_t[\tilde{p}_t]) - u^*[I_t](y_t)\big) - \sum_{t=1}^{T} \zeta_t \leq 2DS\|\boldsymbol{w}^*\|\sqrt{T}$$

Rearranging:

$$\sum_{t=1}^{T} \big(u^*[I_t](\bar{y}_t[p_t] \circ y_t[\tilde{p}_t]) - u^*[I_t](y_t)\big) \leq 2DS\|\boldsymbol{w}^*\|\sqrt{T} + \sum_{t=1}^{T} \zeta_t$$

By Equation 5.11, we can substitute the LHS of the above expression with:

$$\alpha \sum_{t=1}^{T} \big(u^*[I_t](y_t^*[p_t] \circ y_t[\tilde{p}_t]) - u^*[I_t](y_t)\big) - \sum_{t=1}^{T} \xi_t \leq 2DS\|\boldsymbol{w}^*\|\sqrt{T} + \sum_{t=1}^{T} \zeta_t$$

Rearranging and dividing by T both sides we obtain:

$$\frac{1}{T}\sum_{t=1}^{T} \big(u^*[I_t](y_t^*[p_t] \circ y_t[\tilde{p}_t]) - u^*[I_t](y_t)\big) \leq \frac{2DS\|\boldsymbol{w}^*\|}{\alpha\sqrt{T}} + \frac{1}{\alpha T}\sum_{t=1}^{T}(\xi_t + \zeta_t) \tag{5.21}$$

Since $u^*[I_t](y_t^*[p_t] \circ y_t[\tilde{p}_t]) - u^*[I_t](y_t) = u^*(y_t^*[p_t] \circ y_t[\tilde{p}_t]) - u^*(y_t)$, the LHS equates the conditional regret of y_t for part p_t, and thus Equation 5.21 proves the theorem. □

The above theorem is a slightly reworked version of the original bound, first published in [84], using the non-strict α-informative feedback model. Let us now add to the analysis a bound on the conditional regret of any run of P^3 in the general contextual case, within PCL for $n > 1$.

Corollary 5.3.3. *For a user with true utility parameters $\boldsymbol{w}^*$, under the partial α-informative feedback assumption, the average conditional regret incurred by the P^3 algorithm, at epoch $n > 1$ of the PCL algorithm, is upper bounded by:*

$$\frac{1}{T}\sum_{t=1}^{T} \textit{CREG}_{p_t}(y_t) \leq \frac{2DS\|\boldsymbol{w}^*\|}{\alpha\sqrt{T}} + \frac{2B\|\boldsymbol{w}^*\|}{\alpha T} + \frac{1}{\alpha T}\sum_{t=1}^{T}(\xi_t + \zeta_t) \tag{5.22}$$

Proof. In this proof we will employ the superscript indicating the epoch number. Recall from Equation 5.18 that the squared norm of $\boldsymbol{w}_{T+1}$ is bounded by:

$$\|\bar{\boldsymbol{w}}_{T+1}^{(n)}\|^2 \leq \|\bar{\boldsymbol{w}}_T^{(n)}\|^2 + 4D^2S^2 \leq \|\bar{\boldsymbol{w}}_1^{(n)}\|^2 + 4D^2S^2T$$

Here we are considering the case $n > 1$, thus $\|\bar{\boldsymbol{w}}_1^{(n)}\|^2$ is not necessarily equal to $\mathbf{0}$. However, we have that $\bar{\boldsymbol{w}}_1^{(n)} = \boldsymbol{w}^{(n-1)} \in \mathcal{B}$, and thus:

$$\|\bar{\boldsymbol{w}}_{T+1}^{(n)}\|^2 \leq 4D^2S^2T + B^2$$

Using the Cauchy-Schwarz inequality and applying the above bound we have:

$$\langle \boldsymbol{w}^*, \bar{\boldsymbol{w}}_{T+1}^{(n)} \rangle \leq 2DS\|\boldsymbol{w}^*\|\sqrt{T} + B\|\boldsymbol{w}^*\| \tag{5.23}$$

Unrolling the LHS as in Equation 5.3.2:

$$\langle \boldsymbol{w}^*, \bar{\boldsymbol{w}}_1^{(n)} \rangle + \sum_{t=1}^{T} \big(u^*[I_t](\bar{y}_t[p_t] \circ y_t[\tilde{p}_t]) - u^*[I_t](y_t)\big) - \sum_{t=1}^{T} \zeta_t \leq 2DS\|\boldsymbol{w}^*\|\sqrt{T} + B\|\boldsymbol{w}^*\|$$

Since $\langle \boldsymbol{w}^*, \bar{\boldsymbol{w}}_1^{(n)} \rangle \geq -\|\boldsymbol{w}^*\|\|\bar{\boldsymbol{w}}_1^{(n)}\| \geq -B\|\boldsymbol{w}^*\|$, rearranging the above bound we have:

$$\sum_{t=1}^{T} \big(u^*[I_t](\bar{y}_t[p_t] \circ y_t[\tilde{p}_t]) - u^*[I_t](y_t)\big) \leq 2DS\|\boldsymbol{w}^*\|\sqrt{T} + 2B\|\boldsymbol{w}^*\| + \sum_{t=1}^{T} \zeta_t$$

Applying the partial α-informative feedback model and dividing by αT proves the claim. □

The above result shows that only a term vanishing as $\mathcal{O}(1/T)$ is added to the bound for $n > 1$, implying the same convergence rate of $\mathcal{O}(1/\sqrt{T})$ of the non-contextual case.

To further qualify the utility leak that appears in both Theorem 5.3.2 and Corollary 5.3.3, we can find a bound the terms ζ_t depending on the number of iterations $\mathcal{J}_T$ and on the amount of overlapping in the partition $\mathcal{P}$. Let us define the constant $\nu \in [0, 1]$ as:

$$\nu = \max_{k \in [K]} \frac{|I_k \setminus J_k|}{|I_k|}$$

The average utility leak can then be bounded by:

$$\begin{aligned}\sum_{t=1}^{T} \zeta_t &= \sum_{t=1}^{T} u^*[I_t \setminus J_t](\bar{y}_t[p_t] \circ y_t[\tilde{p}_t]) - u^*[I_t \setminus J_t](y_t) \\ &\leq 2D\|\boldsymbol{w}^*\| \sum_{t \in \mathcal{J}_T} |I_t \setminus J_t| \leq 2D\|\boldsymbol{w}^*\| \sum_{t \in \mathcal{J}_T} \nu S \leq 2\nu DS\|\boldsymbol{w}^*\||\mathcal{J}_T|\end{aligned}$$

We can see here the close dependency between the utility leak and the ratio ν of overlapping features. We will see in Section 5.3.4 how ν crucially affects P^3 approximation error too.

5.3.3 Local optimality convergence

Given the result from Theorem 5.3.2 and Corollary 5.3.3, then the conditional regret for each part approaches zero with increasing T. This fact alone, however, still does not prove that the algorithm will eventually converge to a local optimum. In P^3, inference is made on one part p_k at the time over the features J_k, ignoring the other features $I_k \setminus J_k$. From the user perspective, the features $I_k \setminus J_k$ are still affected, even if the algorithm ignores them. This means that a partial configuration $y_t[p_t]$ that was once conditionally optimal, may become suboptimal later when the other parts have changed. In principle, this may happen even if the conditional regret is consistently 0 from some point onward. In this case, even if the user does not provide any feedback anymore (as she cannot improve the parts further), the algorithm might keep changing the object at each iteration to adjust it after the changes made on the other parts in the previous iterations, never reaching a stable local optimum. Next, we will show that this situation can never occur, thanks to our particular utility decomposition.

Theorem 5.3.4. *Let $\tau_1 \le \cdots \le \tau_K \le \hat{\tau}_1 \le \cdots \le \hat{\tau}_K \le T$ such that $p_{\tau_k} = p_{\hat{\tau}_k} = p_k$ and $\textsc{creg}_{p_t}(y_t) = 0$ for all $t \ge \tau_1$. The configuration y_T is a local optimum.*

Proof. The proof proceeds by strong induction. We will first show that $y_t[p_1] = y_{\tau_1}[p_1]$ for all $t \ge \tau_1$. At iteration τ_1, the part p_1 is selected and the partial configuration $y_{\tau_1}[p_1]$ is inferred from the partial utility defined over the features J_1. Since $J_1 = I_1 \setminus \bigcup_{j=2}^{K} I_j$, inference is done over the weights $\boldsymbol{w}_t[J_1]$, which depend exclusively on $p_1 \setminus \{p_2 \colon p_k\}$. Changes on any other part do not affect the weights $\boldsymbol{w}_t[J_1]$. Since $\textsc{creg}_{p_{\tau_1}}(y_{\tau_1}) = 0$ by assumption, the user will not provide an improvement, and thus no weight update is performed. If p_1 is selected again at $t > \tau_1$, the weights $\boldsymbol{w}_t[J_1]$ will be equal to $\boldsymbol{w}_{\tau_1}[J_1]$, since no other part affects those weights. The algorithm will therefore make the same prediction $y_t[p_1] = y_{\tau_1}[p_1]$. Given that by assumption $\textsc{creg}_{p_t}(y_t) = 0$ the weights $\boldsymbol{w}_t[J_1]$ will not be updated as well. Hence, $y_t[p_1] = y_{\tau_1}[p_1]$ and $\boldsymbol{w}_t[J_1] = \boldsymbol{w}_{\tau_1}[J_1]$ for all $t \ge \tau_1$.

By strong induction, suppose that $y_t[p_j] = y_{\tau_j}[p_j]$ for all $j = 1, \ldots, k-1$ and for all $t \ge \tau_{k-1}$. At iteration τ_k, the part p_k is selected and the partial configuration $y_{\tau_k}[p_k]$ is inferred over the features J_k. Since $J_k = I_k \setminus \bigcup_{j=k+1}^{K} I_j$, the weights $\boldsymbol{w}_{\tau_k}[J_k]$ are only affected by changes to $\{p_1 \colon p_k\} \setminus \{p_{k+1} \colon p_K\}$. By inductive hypothesis, the partial configurations $y_t[p_j]$, $1 \le j \le k-1$, do not change for $t \ge \tau_{k-1}$. This means that inference over J_k at $t \ge \tau_k > \tau_{k-1}$ only depends on p_k itself. By assumption $\textsc{creg}_{p_k}(y_{\tau_k}) = 0$, so the weights $\boldsymbol{w}_{\tau_k}[J_k]$ will not change. If p_k is selected again at $t > \tau_k$, the partial configurations $y_t[p_j]$, $1 \le j \le k-1$, will not have changed by inductive hypothesis. The weights $\boldsymbol{w}_t[J_k]$ too will not have changed, thus the partial configuration $y_t[p_k] = y_{\tau_k}[p_k]$. As $\textsc{creg}_{p_k}(y_t) = 0$, the weights $\boldsymbol{w}_t[J_k] = \boldsymbol{w}_{\tau_k}[J_k]$ will not change as well.

This proves that for $k \in [K]$ and any $t > \tau_k$, $y_t[p_k] = y_{\tau_k}[p_k]$. This implies that none of the partial configurations will change for $t \ge \tau_K$. At $\hat{\tau}_1 > \tau_K$, the entire object will not have changed $y_{\hat{\tau}_1} = y_{\tau_K}$. Since $\textsc{creg}_{p_1}(y_{\hat{\tau}_1}) = 0$, it means that $y_{\hat{\tau}_1}$ is conditionally optimal with respect to p_1. Likewise, at $\hat{\tau}_k$, $y_{\hat{\tau}_k} = y_{\tau_K}$ and since $\textsc{creg}_{p_k}(y_{\hat{\tau}_k}) = 0$ the object $y_{\hat{\tau}_k}$ is conditionally optimal with respect to p_k. At iteration $\hat{\tau}_K$, the object $y_{\hat{\tau}_K} = y_{\tau_K}$ is conditionally optimal for all parts $[p_k]_{k=1}^{K}$, therefore $y_T = y_{\hat{\tau}_K} = y_{\tau_K}$ is a local optimum. $\square$

The above theorem proves that if the conditional regret is null at all iterations after τ_1, the P^3 algorithm reaches a local optimum after two consecutive ordered rounds over the parts $p \in \mathcal{P}$. At iteration $t = \tau_K$, if $\sum_{t=\tau_1}^{\tau_K} \text{CREG}_{p_k}(y_t) = 0$ and if the conditions of Theorem 5.3.4 hold in hindsight, the algorithm has actually already reached a local optimum y_{τ_K}, but it needs to double check that all the partial configurations are still conditionally optimal for the user. Since the user provides feedback over the full I_t, changes made to the parts for $t \leq \tau_K$ may still make other partial configurations conditionally suboptimal. Once the algorithm reaches $\hat{\tau}_k$, it can be sure that the object will never change again and all the parts are conditionally optimal for the user, hence the object is a local optimum. This fact can actually be used as a stopping criterion in practice: if the algorithm completes two full runs over the parts and the user has never been able to improve any of them, that means that the current object is a local optimum and the algorithm may stop. We actually employed this stopping criterion in our implementation, but we left it out from Algorithm 10 for simplicity.

5.3.4 Approximation error

As said at the beginning of the analysis, the P^3 algorithm operates with a part-wise realizability assumption, for which there is not a clear notion of regret over full objects. If we assume full realizability, instead, we can see how the local optimum y^* found by P^3 compares with the global optimum y^{**} that would be found by standard coactive learning. In particular, the following theorem states that P^3 has bounded regret (over full objects) with respect to u^*. Recall that K is the number of parts, while $S = \max_{p \in \mathcal{P}} |I(p)|$ and $\|\phi(y)\|_\infty \leq D$.

Theorem 5.3.5. *For a user with true utility parameters $\boldsymbol{w}^*$, the regret of the local optimum y^* found by the P^3 algorithm is upper bounded by:*

$$u^*(y^{**}) - u^*(y^*) \leq 4\nu DSK\|\boldsymbol{w}^*\| \tag{5.24}$$

The above theorem effectively gives us a bound on the *approximation error*, in terms of *full utility*, of P^3 with respect to the best object that could be found with full inference. Notably, this regret bound depends on ν, hinting to the fact that low feature overlap leads to low approximation error. In the extreme case, if there is no overlapping between the parts, then $\nu = 0$, implying that the P^3 algorithm would converge to the global optimum. This is in line with the fact that with no overlap, the utility degenerates to the additively independent case.

If we assume that the norm of the features are bounded by a constant $\|\phi(\cdot)\| \leq R$, as done in standard coactive learning, and we have $\nu = \varepsilon\frac{R}{4DSK}$ for some constant $\varepsilon \in [0, 1]$, then the regret bound in Equation 5.24 becomes:

$$u^*(y^{**}) - u^*(y^*) \leq \varepsilon R\|\boldsymbol{w}^*\|$$

The regret can be normalized between $[0, 1]$ by the constant upper bound $2R\|\boldsymbol{w}^*\|$, yielding:

$$\frac{u^*(y^{**}) - u^*(y^*)}{2R\|\boldsymbol{w}^*\|} \leq \frac{\varepsilon}{2} \qquad\qquad \frac{u^*(y^*)}{R\|\boldsymbol{w}^*\|} \geq \frac{u^*(y^{**})}{R\|\boldsymbol{w}^*\|} - \varepsilon$$

The last inequality affirms that P^3 finds an object whose normalized full utility is no more than ε worse than that of the optimal object. This means that P^3 is an ($\textsc{opt}-\varepsilon$)–approximation algorithm for standard coactive learning. Note that the constant $\varepsilon = \frac{4\nu DSK}{R}$ is computable a priori by the system designer, providing an effective performance guarantee.

Let us now proceed with the proof of Theorem 5.3.5. In order to prove it, though, we will first need to show the following lemma, which quantifies the change in utility of an object y' if the complement for its part-wise contributions $u[J_k](\cdot)$ were to change from y' to y. This quantity is equal to the part-wise contributions of y, complemented by y', to the complementary subutilities $u[J_k \setminus I_k](\cdot)$.

Lemma 5.3.6. *Let $q_k = \{p_1{:}\,p_{k-1}\} \cup \{p_{k+1}{:}\,p_K\}$. The following equality holds for any pair of objects $y, y' \in \mathcal{Y}$:*

$$\sum_{k=1}^{K} u^*[J_k](y'[p_k] \circ y[\tilde{p}_k]) - u^*[J_k](y') = \sum_{k=1}^{K} u^*[I_k \setminus J_k](y[p_k \setminus q_k] \circ y'[q_k]) - u^*[I_k \setminus J_k](y')$$

Proof. Let us start by showing that, since J_k contains only features dependent on $p_1, \ldots, p_k$, the following equality holds:

$$u^*[J_k](y'[p_k] \circ y[\tilde{p}_k]) - u^*[J_k](y') = \sum_{j=1}^{k-1} \left(u^*[J_k](y[p_j \setminus q_j] \circ y'[q_j]) - u^*[J_k](y')\right) \quad (5.25)$$

The above expression follows from the part-wise addition rule (Equation 5.7) by noticing that, for $k \in [K]$, since J_k depends only on $p_1, \ldots, p_k$, we have:

$$\begin{aligned} &u^*[J_k](y'[p_k] \circ y[\tilde{p}_k]) - u^*[J_k](y') \\ &\quad = u^*[J_k](y'[p_k] \circ y[p_1{:}p_{k-1}] \circ y'[p_{k+1}{:}p_K]) - u^*[J_k](y') \\ &\quad = u^*[J_k](y[\{p_1{:}p_{k-1}\} \setminus p_k] \circ y'[p_k{:}p_K]) - u^*[J_k](y') \end{aligned} \quad (5.26)$$

Let us now apply the part-wise addition rule (Equation 5.7) to the non-overlapping parts $\{p_1{:}p_{k-2}\} \setminus \{p_{k-1}{:}p_k\}$ and $p_{k-1} \setminus (\{p_1{:}p_{k-2}\} \cup p_k)$:

$$\begin{aligned} &u^*[J_k](y[\{p_1{:}p_{k-1}\} \setminus p_k] \circ y'[p_k{:}p_K]) \\ &\quad = u^*[J_k](y[\{p_1{:}p_{k-2}\} \setminus \{p_{k-1}{:}p_k\}] \circ y'[p_{k-1}{:}p_K]) \\ &\qquad + u^*[J_k](y[p_{k-1} \setminus (\{p_1{:}p_{k-2}\} \cup p_k)] \circ y'[\{p_1{:}p_{k-2}\} \cup \{p_k{:}p_K\}]) - u^*[J_k](y') \end{aligned}$$

Recurring on the first summand of the RHS:

$$\begin{aligned} &u^*[J_k](y[\{p_1{:}p_{k-1}\} \setminus p_k] \circ y'[p_k{:}p_K]) \\ &\quad = \sum_{j=1}^{k-1} u^*[J_k](y[p_j \setminus (\{p_1{:}p_{j-1}\} \cup \{p_{j+1}{:}p_k\})] \circ y'[q_j]) - \sum_{j=1}^{k-2} u^*[J_k](y') \end{aligned}$$

Plugging the above expression in Equation 5.26:

$$\begin{aligned}
&u^*[J_k](y'[p_k] \circ y[\tilde{p}_k]) - u^*[J_k](y') \\
&= \left(\sum_{j=1}^{k-1} u^*[J_k](y[p_j \setminus (\{p_1{:}p_{j-1}\} \cup \{p_{j+1}{:}p_k\})] \circ y'[q_j]) - \sum_{j=1}^{k-2} u^*[J_k](y') \right) - u^*[J_k](y') \\
&= \sum_{j=1}^{k-1} (u^*[J_k](y[p_j \setminus (\{p_1{:}p_{j-1}\} \cup \{p_{j+1}{:}p_k\})] \circ y'[q_j]) - u^*[J_k](y'))
\end{aligned}$$

Since the features J_k are only affected by parts $\{p_1{:}p_k\} \setminus \{p_{k+1}{:}p_K\}$, we have that:

$$\begin{aligned}
&u^*[J_k](y[p_j \setminus (\{p_1{:}p_{j-1}\} \cup \{p_{j+1}{:}p_k\})] \circ y'[q_j]) \\
&\quad = u^*[J_k](y[p_j \setminus (\{p_1{:}p_{j-1}\} \cup \{p_{j+1}{:}p_K\})] \circ y'[q_j]) = u^*[J_k](y[p_j \setminus q_j] \circ y'[q_j])
\end{aligned}$$

which proves Equation 5.25. As part p_j only affects features in I_j, Equation 5.25 equates to:

$$u^*[J_k](y'[p_k] \circ y[\tilde{p}_k]) - u^*[J_k](y') = \sum_{j=1}^{k-1} (u^*[J_k \cap I_j](y[p_j \setminus q_j] \circ y'[q_j]) - u^*[J_k \cap I_j](y'))$$

Summing up for all $k \in [K]$ we get:

$$\begin{aligned}
&\sum_{k=1}^{K} (u^*[J_k](y'[p_k] \circ y[\tilde{p}_k]) - u^*[J_k](y')) \\
&\quad = \sum_{k=1}^{K} \sum_{j=1}^{k-1} (u^*[J_k \cap I_j](y[p_j \setminus q_j] \circ y'[q_j]) - u^*[J_k \cap I_j](y'))
\end{aligned}$$

By unrolling the above expression we can see that each $j \in [K]$ occurs together with all $k = j+1, \ldots, K$, adding up all intersections $J_{j+1} \cap I_j, \ldots, J_K \cap I_j$. This means that we end up summing all features in I_j but those in J_j, hence $I_j \setminus J_j$. Simplifying we get:

$$\sum_{k=1}^{K} u^*[J_k](y'[p_k] \circ y[\tilde{p}_k]) - u^*[J_k](y') = \sum_{k=1}^{K} u^*[I_k \setminus J_k](y[p_k \setminus q_k] \circ y'[q_k]) - u^*[I_k \setminus J_k](y')$$

thereby proving the claim. □

We are now ready to prove Theorem 5.3.5.

Proof. [Theorem 5.3.5] Let us first define the *part-wise optimum* $\hat{y}$, which is the result of the combination of the partial configurations obtained by optimizing u^* over the parts p_k

separately, for all $k \in [K]$:

$$\begin{aligned} \hat{y} &= y'_1[p_1] \circ \cdots \circ y'_K[p_K] \\ y'_k[p_k] &= \operatorname*{argmax}_{y[p_k] \in \mathcal{Y}_{p_k}} u^*[J_k](y[p_k] \circ y'_{k-1}[\tilde{p}_k]) \end{aligned}$$

Since J_1 depends only on p_1, the first partial configuration $y'_1[p_1]$ can be computed using any object as complement. If parts overlap, we choose the combination yielding the maximum utility. The object $\hat{y}$ is the best object we can achieve with our part-wise inference procedure knowing u^*. As such, it follows that for all $p_k \in \mathcal{P}$:

$$u^*[J_k](\hat{y}) - u^*[J_k](y^{**}[p_k] \circ \hat{y}[\tilde{p}_k]) \geq 0 \tag{5.27}$$

Since y^* is a local optimum, by Lemma 5.3.1, we also have that, for all $p_k \in \mathcal{P}$:

$$u^*[I_k](y^*) - u^*[I_k](\hat{y}[p_k] \circ y^*[\tilde{p}_k]) \geq 0 \tag{5.28}$$

Applying part-wise linearity to Equation 5.27, we get:

$$u^*[J_k](\hat{y}[p_k] \circ y^{**}[\tilde{p}_k]) - u^*[J_k](y^{**}) \geq 0$$

Changing signs and direction of the inequality:

$$u^*[J_k](y^{**}) - u^*[J_k](\hat{y}[p_k] \circ y^{**}[\tilde{p}_k]) \leq 0$$

We can bound the above inequality using Equation 5.28:

$$u^*[J_k](y^{**}) - u^*[J_k](\hat{y}[p_k] \circ y^{**}[\tilde{p}_k]) \leq u^*[I_k](y^*) - u^*[I_k](\hat{y}[p_k] \circ y^*[\tilde{p}_k])$$

Decomposing I_k into J_k and $I_k \setminus J_k$ we get:

$$\begin{aligned} u^*[J_k](y^{**}) - u^*[J_k](\hat{y}[p_k] \circ y^{**}[\tilde{p}_k]) \leq\ & u^*[J_k](y^*) - u^*[J_k](\hat{y}[p_k] \circ y^*[\tilde{p}_k]) \\ & + u^*[I_k \setminus J_k](y^*) - u^*[I_k \setminus J_k](\hat{y}[p_k] \circ y^*[\tilde{p}_k]) \end{aligned}$$

Rearranging:

$$\begin{aligned} u^*[J_k](y^{**}) - u^*[J_k](y^*) \leq\ & u^*[J_k](\hat{y}[p_k] \circ y^{**}[\tilde{p}_k]) - u^*[J_k](\hat{y}[p_k] \circ y^*[\tilde{p}_k]) \\ & + u^*[I_k \setminus J_k](y^*) - u^*[I_k \setminus J_k](\hat{y}[p_k] \circ y^*[\tilde{p}_k]) \end{aligned}$$

Applying again the part-wise linearity to the above expression we get:

$$\begin{aligned} u^*[J_k](y^{**}) - u^*[J_k](y^*) \leq\ & u^*[J_k](y^{**}) - u^*[J_k](y^{**}[p_k] \circ y^*[\tilde{p}_k]) \\ & + u^*[I_k \setminus J_k](y^*) - u^*[I_k \setminus J_k](\hat{y}[p_k] \circ y^*[\tilde{p}_k]) \end{aligned}$$

Summing over $k \in [K]$:

$$\begin{aligned}\sum_{k\in[K]} (u^*[J_k](y^{**}) - u^*[J_k](y^*)) \leq& \sum_{k\in[K]} (u^*[J_k](y^{**}) - u^*[J_k](y^{**}[p_k] \circ y^*[\tilde{p}_k])) \\ &+ \sum_{k\in[K]} (u^*[I_k \setminus J_k](y^*) - u^*[I_k \setminus J_k](\hat{y}[p_k] \circ y^*[\tilde{p}_k]))\end{aligned}$$

We can now apply Lemma 5.3.6 to the first summation of the RHS of the above inequality:

$$\begin{aligned}\sum_{k\in[K]} u^*[J_k](y^{**}) - u^*[J_k](y^*) \leq& \sum_{k\in[K]} u^*[I_k \setminus J_k](y^{**}) - u^*[I_k \setminus J_k](y^*[p_k \setminus q_k] \circ y^{**}[q_k]) \\ &+ \sum_{k\in[K]} u^*[I_k \setminus J_k](y^*) - u^*[I_k \setminus J_k](\hat{y}[p_k] \circ y^*[\tilde{p}_k])\end{aligned}$$

Now we can bound both the sums in the RHS with:

$$\begin{aligned}\sum_{k\in[K]} u^*[J_k](y^{**}) - u^*[J_k](y^*) \leq& \sum_{k\in[K]} u^*[I_k \setminus J_k](y^{**}) - u^*[I_k \setminus J_k](y^*[p_k \setminus q_k] \circ y^{**}[q_k]) \\ &+ \sum_{k\in[K]} u^*[I_k \setminus J_k](y^*) - u^*[I_k \setminus J_k](\hat{y}[p_k] \circ y^*[\tilde{p}_k]) \\ \leq& \, 2D\|\boldsymbol{w}^*\| \sum_{k\in[K]} |I_k \setminus J_k| + 2D\|\boldsymbol{w}^*\| \sum_{k\in[K]} |I_k \setminus J_k| \\ \leq& \, 4D\|\boldsymbol{w}^*\| \sum_{k\in[K]} \nu S \\ \leq& \, 4\nu DSK\|\boldsymbol{w}^*\|\end{aligned}$$

which concludes the proof. □

5.3.5 Remarks

To recap, we sought to prove that the P^3 algorithm converges to a local optimum. Lemma 5.3.1 showed that we can do that by finding an object that is conditionally optimal with respect to all the parts $p \in \mathcal{P}$, i.e. by minimizing the conditional regret for all parts. Theorem 5.3.2 and Corollary 5.3.3 proved an upper bound on the average conditional regret over the iterations $t \in [T]$ for the non-contextual and contextual case respectively. Theorem 5.3.4 proves that, if $\text{CREG}_{p_t}(y_t) = 0$ for $t \geq \tau_1$, the algorithm reaches a local optimum after two complete iterations over the parts $p_k \in \mathcal{P}$. From Theorem 5.3.2 we have that the average conditional regret approaches 0, implying that the conditions for Theorem 5.3.4 will eventually hold, thereby proving that the P^3 algorithm will eventually converge to a local optimum for $T \to \infty$. Finally, with Theorem 5.3.5 we proved that the local optimum found by the P^3 algorithm enjoys a regret with respect to the global optimum bounded by a constant dependent on the maximal ratio ν of overlapping feature within the feature subsets. This effectively means that P^3 constitutes an $(\text{OPT} - \varepsilon)$–approximation algorithm for standard coactive learning (with ε

defined in Section 5.3.4).

The analysis presented in this chapter is based on the one first published in [84]. Here we extended the original analysis by providing a clearer notion of part-wise realizability, extending the proofs to the non-strict partial α-informative feedback, adding a conditional regret bound on the contextual case, explicitating the dependency of the utility leak with the overlap of the feature subsets, and proving the bound on the approximation error of the P^3 algorithm.

5.4 Experiments

We now apply our decomposition strategy to constructive problems by dividing the underlying constraint optimization problems into parts. The parts $p \in \mathcal{S}$ are subsets of decision variables and feature subsets $I(p)$ simply contain all the features dependent on at least one decision variable in p. We restrict to linear features only, which makes the part decomposition satisfy the part-wise addition rule defined in Section 5.1.2.

We report in this section the same experiments made in the original contribution [84]. The MILP solver used was Gecode[3] and the full experimental setup is available on Github[4].

We ran PCL on three constructive preference elicitation tasks of increasing complexity, comparing different degrees of user informativeness. According to our experiments, informativeness is the most critical factor. The three problems involve rather large configurations, which can not be handled by coactive interaction via complete configurations. For instance, in [196] the user is tasked to solve relatively simple SAT instances over three variables and (at most) eight clauses; in some cases users were observed to show signs of cognitive overload. In comparison, our simplest realistic problem involve 35 categorical variables (with 8 possible values) and 74 features, plus additional hard constraints. As a consequence, Coactive Learning can not be applied as-is, and part-wise interaction is necessary.

We employed a user simulation protocol similar to that of [255]. First, for each problem, we sampled 20 vectors $\boldsymbol{w}^*$ at random from a standard normal distribution. Then, upon receiving a recommendation $y_t[p_t]$, an improvement $\bar{y}_t[p_t]$ is generated by solving the following problem:

$$\begin{aligned} \operatorname*{argmin}_{y[p_t] \in \mathcal{Y}_{p_t}} \quad & u^*[I_t](y[p_t] \circ y_t[\tilde{p}_t]) \\ \text{s.t.} \quad & u^*[I_t](y[p_t] \circ y_t[\tilde{p}_t]) - u^*[I_t](y_t) \\ & \qquad \geq \alpha(u^*[I_t](y^*[p_t] \circ y_t[\tilde{p}_t]) - u^*[I_t](y_t)) \end{aligned}$$

This formulation clearly satisfies the partial α-informativeness assumption (Equation 5.11).

[3]`gecode.org` [4]`github.com/unitn-sml/pcl`

Synthetic setting

We designed a simple synthetic problem inspired by spin glass models, see Figure 5.1 for a depiction. In this setting, a configuration y consists of a 4×4 grid. Each node in the grid is a binary 0-1 variable. Adjacent nodes are connected by an edge, and each edge is associated to an indicator feature that evaluates to 1 if the incident nodes have different values (green in the figure), and to -1 otherwise (red in the figure). The utility of a configuration is simply the weighted sum of the values of all features (edges). The parts $p \in \mathcal{P}$ consist of all the non-overlapping 2×2 sub-grids of y, for a total of 4 parts (indicated by dotted lines in the figure).

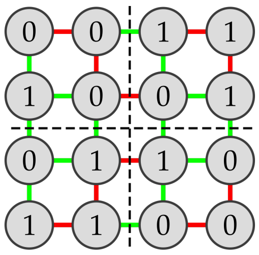

Figure 5.1: Example of grid configuration.

Since the problem is small enough for inference of complete configurations to be practical, we compared PCL to standard Coactive Learning, using the implementation of [255]. In order to keep the comparison as fair as possible, the improvements fed to CL were chosen to match the utility gain obtained by PCL. We further report the performance of three alternative part selection strategies: random, smallest (most independent) part first, and UCB1.

The results can be found in the first column of Figure 5.2. We report both the regret (over *complete* configurations) and the cumulative runtime of all algorithms, averaged over all users, as well as their standard deviation. The regret plot shows that, despite being restricted to work with 2×2 configurations, PCL does recommend *complete* configurations of quality comparable to CL after enough queries are made. Out of the three part selection strategies, random performs best, with the other two more informed alternatives (especially smallest first) quite close. The runtime gap between full and part-wise inference is already clear in this small synthetic problem; complete inference quickly becomes impractical as the problem size increases.

Training planning

Generating personalized training plans based on performance and health monitoring has received a lot of attention recently in sport analytics (see e.g. [105]). Here we consider the problem of synthesizing a week-long training plan y from information about the target athlete. Each day includes 5 time slots (two for the morning, two for the afternoon, one for the evening), for 35 slots total. We assume to be given a fixed number of training activities (7

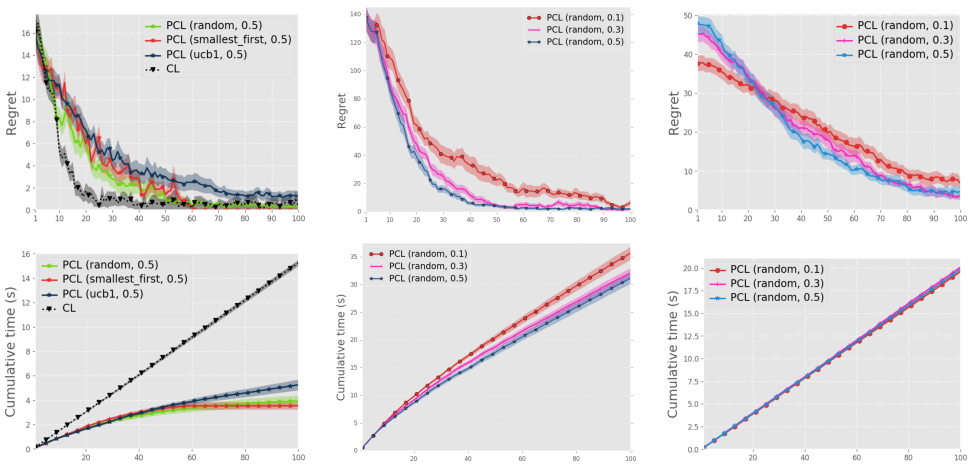

Figure 5.2: Regret over *complete* configurations (top) and cumulative runtime (bottom) of PCL and CL on our three constructive problems: synthetic (left), training planning (middle), and hotel planning (right). The x-axis is the number of iterations, while the shaded areas represent the standard deviation. Best viewed in color.

in our experiments: walking, running, swimming, weight lifting, push-ups, squats, abs), as well as knowledge of the slots in which the athlete is available. The training plan y associates an activity to each slot where the athlete is available. Our formulation tracks the amount of improvement (e.g. power increase) and fatigue over five different body parts (arms, torso, back, legs, and heart) induced by performing an activity for one time slot. Each day defines a part.

The mapping between training activity and improvement/fatigue over each body part is assumed to be provided externally. It can be provided by the athlete or medical personnel monitoring his/her status. The features of y include, for each body part, the total performance gain and fatigue, computed over the recommended training plan according to the aforementioned mapping. We further include inter-part features to capture activity diversity in consecutive days. The fatigue accumulated in 3 consecutive time slots in any body parts does not exceed a given threshold, to prevent injuries.

In this setting, CL is impractical from both the cognitive and computational points of view. We ran PCL and evaluated the impact of user informativeness by progressively increasing α from 0.1, to 0.3, to 0.5. The results can be seen in Figure 5.2. The plots show clearly that, despite the complexity of the configuration and constraints, PCL can still produce very low-regret configurations after about 50 iterations or less.

Understandably, the degree of improvement α plays an important role in the performance of PCL and, consequently, in its runtime (users at convergence do not contribute to the runtime), at least up to $\alpha = 0.5$. Recall, however, that the improvements are part-wise, and hence α quantifies the degree of *local* improvement: part improvements may be very informative on their own, but only give a modest amount of information about the full configuration.

However, it is not unreasonable to expect that users to be very informative when presented with reasonably sized (and simple) parts. Crucially PCL allows the system designer to define the parts appropriately depending on the application.

Hotel planning

Finally, we considered a complex furniture allocation problem: furnishing an entire hotel. The problem is encoded as follows. The hotel is represented by a graph: nodes are rooms and edges indicate which rooms are adjacent. Rooms can be of three types: normal rooms, suites, and dorms. Each room can hold a maximum number of furniture pieces, each associated to a cost. Additional, fixed nodes represent bathrooms and bars. The type of a room is decided dynamically based on its position and furniture. For instance, a normal room must contain at most three single or double beds, no bunk beds, and a table, and must be close to a bathroom. A suite must contain one bed, a table and a sofa, and must be close to a bathroom and a bar. Each room is a part, and there are 15 rooms to be allocated.

The feature vector contains 20 global features plus 8 local features per room. The global features include different functions of the number of different types of rooms, the total cost of the furniture and the total number of guests. The local features include, instead, characteristics of the current room, such as its type or the amount of furniture, and other features shared by adjacent rooms, e.g. whether two rooms have the same type. These can encode preferences like "suites and dorms should not be too close", or "the hotel should maintain high quality standards while still being profitable". Given the graph structure, room capacities, and total budget, the goal is to furnish all rooms according to the user's preferences.

This problem is hard to solve to optimality with current solvers; part-based inference alleviates this issue by focusing on individual rooms. There are 15 rooms in the hotel, so that at each iteration only 1/15 of the configuration is affected. Furthermore, the presence of the global features implies dependencies between all rooms. Nonetheless, the algorithm manages to reduce the regret by an order of magnitude in around a 100 iterations, starting from a completely uninformed prior. Note also that as for the training planning scenario, an alpha of 0.3 achieves basically the same results as those for alpha equal to 0.5.

5.5 Summary

In this chapter we addressed the issue of scalability of constructive preference elicitation to large constructive domains by partitioning the problem domain and eliciting the utility function one part at the time. We derived the P^3 algorithm, based on the preference perceptron [247], which only performs inference and updates on partial configurations, speeding up the per-iteration inference time substantially. The user feedback also consists in partial improvements, making it cognitively affordable to tackle large constructive problems. We provided an extensive theoretical analysis demonstrating that, despite working only with partial configuration, the P^3 algorithm is capable of reaching a locally optimal solution with bounded approximation error with respect to standard coactive learning. The theoretical analysis presented in this chapter is based on that of the original publications [84], but extended in a number of directions. Finally, we empirically validated the performance of the P^3 algorithm on three constructive scenarios of increasing complexity, showing that it is capable of reducing the regret almost to optimality, despite the worst-case approximation error saying otherwise.

Chapter 6

Critiquing and feature elicitation

In Chapter 4 we introduced our framework for constructive preference elicitation based on coactive learning, while in Chapter 5 we derived a new technique to scale up to larger constructive problems. In this chapter we will tackle another important problem that arises in a constructive scenario, namely how to handle an exponentially large feature space. In constructive preference elicitation, the size of the output domain $\mathcal{Y}$ is exponential in number of attributes used to represent the objects, but so is the number of possibly relevant features. As explained in Chapter 4, features represent the "preference criteria" with which a user can assess the quality of a recommended object. Features may be arbitrarily complex formulas of the attributes of the objects. In Section 4.4.2 we have also seen that functions of the form $u(x, y) = \langle \boldsymbol{w}, \boldsymbol{\phi}(x, y) \rangle$ can represent any possible utility with an exponential number of features. One way to picture the size of the possible formulas is to consider a set of operators, such as logical conjunction and disjunction or arithmetic sum and multiplication, and define each feature by combining an arbitrary number of attributes using these operators. In this case, the size of the feature space is $\mathcal{O}(s^{(n+l)})$ where s is the maximum number of allowed attributes to combine for each feature, n is the number of attributes, and l is the number of operators. Enumerating all the possible features beforehand would make the complexity of the inference problem exponential and thus unfeasible [200]. One possible approach to solve this problem, adopted by [48], is to select the most relevant ones by learning a sparse preference model [261]. While sparsification may mitigate the problem of intractable inference, this approach still suffers from several drawbacks. First, learning with these many features is harder and more computationally demanding. Also, it is still possible that relevant but unanticipated preference criteria might be excluded from the feature space. Another approach that we have mentioned in Section 4.4.2, is the kernel-based technique from [77, 78]. While feature augmentation and kernel methods have long been used to improve generalization [237], in this case they pose a problem on the inference procedure, as the kernelized model from [77] would require solving either a non-linear optimization problem in the dual variables, or a linear optimization problem with an exponential number of features.

While a good feature set is important for generalization, and elicitation with a more expressive utility has more chances to reach good solutions, ultimately the features need to reflect the best trade-offs for the user, which are hard to determine beforehand. The work of Pu and Faltings [211] clearly highlights the problem. They argue that the set of preference criteria of a user is not completely determined in advance. Users are likely to discover relevant preference criteria throughout the elicitation process [203]. Also, the weight a user assigns to a certain criterion is likely to change as she uncovers new available options that she did not know about. For this reason they propose an interaction schema that allows the user to express their preference criteria and trade-offs through soft constraints.

In this chapter, we present a method inspired by [211]. We take however a slightly different approach and propose to *elicit* the relevant features for the users, in parallel with the elicitation of their preferences. In particular, we devised an algorithm that allows the users to *critique* some of the examples. Like soft constraints for [211], critiques are essentially formulas of the attributes of the objects that can be interpreted as features. Unlike [211], we give a precise *semantics* to our critiques. Using our system, the user does not state any critique that comes to mind, but rather she is asked to state a critique when certain conditions occur. In practice, critiques in our system are *explanations* for ambiguous improvements elicited via standard coactive learning. When a user provides an improvement that strongly disagrees with the current utility model of the algorithm, chances are that the algorithm does not possess the right feature that "explains" that improvement, together with the other evidence it has already collected. At this point the algorithm can ask the user to provide a critique, i.e. a formula explaining why the feedback is actually an improvement over the recommendation. This formula can then be attached to the current utility model and make it more expressive. This process allows to gradually enlarge the feature space of the preference model, while asking only the necessary critiques and minimizing the effort for the user. Crucially, unlike [211], this approach provably converges to an optimal solution when the feature space becomes expressive enough. As an additional benefit, the inference procedure can be solved much faster, thanks to the fact that, at least at the first iterations, the algorithm works with many fewer features than those needed by standard coactive learning.

The rest of the chapter is organized as follows. In Section 6.1 we review the literature on critiquing recommendation systems. Section 6.2 then exposes the details of our proposed critiquing algorithm, also pointing out when and how to elicit critiques. In Section 6.3 we will proceed in analyzing our algorithm, proving that it will converge to an optimal solution despite the fact that it starts off in a lower-dimensional feature space than the space of the user true utility. In Section 6.4, the empirical results will show how the algorithm reaches optimal solutions while eliciting only a fraction of the total number of the user features. Section 6.5 concludes the chapter.

6.1 Critiques

Example critiquing is a type of feedback that has been extensively employed in interactive recommendation systems [54, 181]. When using a critiquing-based recommender, after receiving a set of recommendations, a user is allowed to reply with a *critique*, i.e. a suggestion on how to improve the proposed solutions. This resembles in many ways the coactive interaction, but instead of providing a new better object, the user provides a *direction* of improvement. For instance, when presented with a product, the user may state a critique of the type “I would like a similar but cheaper product”. In this case, the attribute to improve is the price, specifically by lowering it. The critique then acts as a sort of filter for the following recommendation, in which similar products are retrieved but incorporate the specified suggestion.

Many critiquing systems proposed in the literature do not employ an explicit preference model. They use, instead, the current set of recommended options as an implicit representation of the user preferences, which is incrementally refined through critiques [44, 47, 218, 219]. Few other critiquing systems, instead, do keep an explicit representation of the user preferences [274, 279, 289]. Most closely related to our approach are those systems that represent the user preferences as *soft constraints*. Recall from Section 2.2.1 that soft constraints are arbitrary formulas over objects attributes with attached weights, and recommendations are selected based on the cumulative weight of the formulas satisfied by the object. As pointed out in Chapter 4, the feature functions used in our constructive preference elicitation framework are comparable to soft constraints, with the added property of not necessarily being Boolean valued.

The critiquing method we propose follows the same intuition behind the approach of Viappiani et al. (2006) [277]: critiques can be expressed as soft constraints, and the preference model can be extended as new critiques are collected. This approach has been extensively used by many constraint-based recommenders [55, 275, 279, 287]. These systems, however, most often lack a principled way to determine proper values to the weights associated to the soft constraints, assigning them heuristically or even rely on the user to manually adjust them based on their preferences [210, 211, 287]. To the best of our knowledge, the first attempt to use machine learning techniques to estimate user preferences encoded as soft constraints is the work by Rossi and Sperduti [22, 225], and, in particular, in [227] they propose a system that allows the user to state preferences as soft constraints and then learns the weights of the soft constraints from collected user ratings. Another early form of learning in this context appears in the system developed by Viappiani et al. (2007) [279], in which they use a probabilistic modeling and learning of critiquing suggestions, to help the user stating relevant critiques. Our approach has many similarities with [227], but they differ in several important ways, as explained in Chapter 4.

The idea of eliciting preferences and features simultaneously in an incremental fashion was already proposed by Boutilier et al. [30, 31]. Notably, they propose to use *concept learning* [10] to generate new constraints from examples collected throughout the interaction in a minimax regret setting [31]. In Chapter 4 we discussed the disadvantages of regret-based elicitation as opposed to our coactive learning approach, but it is worth noting that our work draws from [31] the fundamental idea that it is not necessarily required to elicit the full feature set of the user, but a small subset may be enough to reach an optimal solution. While in this work we assume for simplicity that the user is capable to provide an arbitrarily complex critique, we can relax this assumption by generating critiques through concept learning, as done by Boutilier et al. [31]. We will discuss this possibility in Section 6.2.3.

Algorithm 12 The critiquing preference perceptron algorithm [255].

```
1: procedure CPP($T \in \mathbb{N}^+$)
2:     $\boldsymbol{w}_1 \leftarrow 0$
3:     for $t = 1, \dots, T$ do
4:         Receive context $x_t$ from the user
5:         $y_t \leftarrow \text{argmax}_{y \in \mathcal{Y}} \langle \boldsymbol{w}_t, \boldsymbol{\phi}_t(x_t, y) \rangle$
6:         User provides improvement $\bar{y}_t$
7:         if NeedCritique$(x_t, y_t, \bar{y}_t)$ then
8:             Receive critique $\rho$ from the user
9:             $\boldsymbol{\phi}_t \leftarrow \boldsymbol{\phi}_t \circ [\rho]$
10:            $\boldsymbol{w}_t \leftarrow \boldsymbol{w}_t \circ [0]$
11:        $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t + \boldsymbol{\phi}_t(x_t, \bar{y}_t) - \boldsymbol{\phi}_t(x_t, y_t)$
12:        $\boldsymbol{\phi}_{t+1} \leftarrow \boldsymbol{\phi}_t$
```

6.2 Coactive critiquing

We detail here our approach, called *coactive critiquing*, that extends the coactive learning framework [247] to allow feature elicitation through critiquing feedback [255]. In particular, we present an algorithm, dubbed *critiquing preference perceptron* (CPP), that learns user preferences as in the standard preference perceptron (see Section 3.3), and is able to expand the feature space of the utility function from the user critiques. Algorithm 12 shows an updated version of the critiquing preference perceptron algorithm, originally presented in [255], that also includes contexts in the utility function.

6.2.1 Critiquing preference perceptron

The algorithm starts with an initial set of features $\boldsymbol{\phi}_1$[1] and traverses increasingly more expressive feature spaces $\boldsymbol{\phi}_t$, $t = 1, \dots, T$, as critiques are collected. At each iteration $t \in [T]$, the algorithm receives a context x_t and recommends an object y_t, by performing inference over the current feature space $\boldsymbol{\phi}_t$. The algorithm then receives an improvement $\bar{y}_t$ from the user as in standard coactive learning. At this point, the algorithm checks whether or not a critique is needed (line 7 of Algorithm 12). If the condition is met, the algorithm queries the user for a critique ρ, which is appended to the current feature vector $\boldsymbol{\phi}_t$. The weight vector is also zero-padded accordingly. Finally, the algorithm updates the weights as in the standard preference perceptron (line 11).

We assume the user critique ρ to be a real function of the type $\rho : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$, such that $\rho(x_t, \bar{y}_t) \geq \rho(x_t, y_t)$. This constraint effectively *explains* why the user prefers $\bar{y}_t$ over y_t. When collecting critiques we make a "local rationality" assumption: given two objects y_t and $\bar{y}_t$, the user is always able to state at least one critique to differentiate between the two. The user explicitly created $\bar{y}_t$ to be better than y_t, so it is reasonable to assume the user can also

[1]We will interchangeably use the notation $\boldsymbol{\phi}_t(\cdot)$ to refer to both the set of features held by the algorithm at iteration $t \in [T]$ and the feature map defined over this set.

state at least one reason why this is actually an improvement. In the ideal scenario, the user would respond with a critique that best explains the difference between the objects, i.e. the critique with maximum difference $\rho(x_t, \bar{y}_t) - \rho(x_t, y_t)$. However, the user could reply with a suboptimal critique, a fact that we address in the theoretical analysis and in our experiments.

In the next two sections we cover the conditions for the NeedCritique function (line 7) and the methods a coactive critiquing system can use to elicit critiques in a user friendly manner.

6.2.2 When to ask for critiques

Asking critiques allows the algorithm to build up an increasingly richer feature space, which is not only useful, but needed to reach an optimal solution. However, each critique comes with a cost in cognitive effort for the user, so asking them should be limited to only those cases in which they are really needed and actually useful. That is why a proper design of the NeedCritique function (line 7) is critical for any coactive critiquing system. If this procedure is too lazy, it may hurt the ability of the model to properly represent the user preferences, and thus advancing towards an optimal solution. On the other hand, if the NeedCritique is too eager, many unnecessary critiques might be elicited, incurring in higher user effort.

We propose a simple, yet effective, method that is a good trade-off between representation power of the utility model and cognitive effort for the user. We design the NeedCritique so to ask the user to provide a critique whenever the provided improvement $\bar{y}_t$ disagrees with the other preference information collected so far. In other words, a critique is asked when the current ranking constraint $\bar{y}_t \succcurlyeq y_t$ cannot be included in the utility model without violating the ranking constraints observed in the previous iterations. This condition can be tested by checking the existence of a weight vector $\boldsymbol{w}$ that can correctly rank all the pairwise preference examples in the dataset $\mathcal{D}$, i.e. more formally:

$$\nexists \boldsymbol{w} \; \forall (y_t, \bar{y}_t) \in \mathcal{D} \quad \langle \boldsymbol{w}, \boldsymbol{\phi}_t(x_t, \bar{y}_t) - \boldsymbol{\phi}_t(x_t, y_t) \rangle > 0 \tag{6.1}$$

For noiseless users, this criterion is guaranteed to elicit critiques only when the user preferences collected so far are not representable anymore with the current set of features. Noisy users, on the other hand, could state contradictory preference constraints, in which case the algorithm may end up asking one critique for every iteration. To avoid this case, one might reduce the number of examples used in the check, either by sampling or by discarding older examples. This also has the positive side effect of reducing the computational overhead of the check.

While this simple criterion is effective in practice, it might be suboptimal because it is not guaranteed (not even in the noiseless case) that the critique provided by the user will make the condition in Equation 6.1 false in the next iteration. More sophisticated strategies could be devised to address this issue and to tackle the problem of noisy ranking constraints. We experimented with a strategy based on estimating the likelihood of inconsistencies being due to lack of features or noise, but we did not see an improvement over the simpler criterion in Equation 6.1, therefore we will report our results with the latter.

6.2.3 How to acquire critiquing feedback

The primary focus of this work is to devise an extension to coactive learning to allow critiquing feedback, so we assume there exist an interface that allows the users to generate fairly articulated critiques that reflect their preferences. Here we mention some of the methods that can be used to acquire critiquing feedback in our setting.

First and foremost, simple interfaces used in other rule-based critiquing systems may be employed in our setting as well [44, 46, 165, 211, 266, 277]. We argue that in our case it might even be easier for users to interact with such interface, since they do not have to provide generic critiques, which might be difficult to think about when the objects have many components, but instead they only need to compare two specific objects and provide a critique to set them apart.

The specificity of the critiques to only two comparable objects, allows us to think about more complex interfaces. For instance, one could use a concept learning approach similar to that of Boutilier et al. (2010) [31]. They collect feedback from membership queries (e.g. "Is this object *safe*?") that can be used to learn "concepts" (e.g. the concept "safe") which can then be incorporated in the utility model. We can use a similar "labeling" system, but instead we could ask the user to assign a distinguishing label, that is a label for which $\bar{y}_t$ is a positive instance and y_t is a negative instance (or vice versa), e.g. "$\bar{y}_t$ is safe" and "y_t is not safe". In this case, it might even be better to allow the user to differentiate the objects by their degree of a property or lack thereof, e.g. "$\bar{y}_t$ is *safer* than y_t". Soft constraints can be learned from these critiquing examples using inductive logic programming [71] or constraint learning [213]. We can easily imagine this type of interaction being carried out through natural language, since labels of this kind might be extracted from text through some simple form of semantic parsing [162]. This is especially true if we think about the coactive critiquing interaction as taking place through a conversation of a user with a dialogue system [122, 283].

6.3 Analysis

As in standard coactive learning, we assume the user to behave according to a true utility function $u^* : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$. In this work, we relax the assumption of existence of a "universal" feature map $\boldsymbol{\phi}(\cdot)$, used to elicit the preference of any user, but rather we posit that each user behaves according to her own true feature map $\boldsymbol{\phi}^* : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^{d'}$. This feature map contains a number of features unknown to the algorithm (including their weights). The utility function of each user has therefore the form $u^*(x, y) = \langle \boldsymbol{w}^*, \boldsymbol{\phi}^*(x, y) \rangle$, with $\boldsymbol{w}^*$ and $\boldsymbol{\phi}^*(x, y)$ of unknown but finite dimension.

For the rest of this analysis we will assume that the set of features held by the algorithm to be a subset of the user features. We can assume, without loss of generality, that the initial features $\boldsymbol{\phi}_1$ are included in the user features $\boldsymbol{\phi}^*$ as well, with zero weight if not relevant.

The goal of this analysis is to show that CPP enjoys a $\mathcal{O}(1/\sqrt{T})$ bound on its average regret:

$$\frac{1}{T}\sum_{t=1}^{T}\text{REG}(x_t, y_t) = \frac{1}{T}\sum_{t=1}^{T}\langle \boldsymbol{w}^*, \boldsymbol{\phi}^*(x_t, y_t^*) - \boldsymbol{\phi}^*(x_t, y_t)\rangle$$

To derive the bound, we first need to transform the temporary feature maps $\boldsymbol{\phi}_t$ into vectors of the same dimension of $\boldsymbol{\phi}^*$. We can do so by "masking" those features that are not owned by the algorithm at iteration t. More formally, let $\mathcal{F}_t$ be the set of features collected up to iteration t, and let $\boldsymbol{z}_t$ be a 0-1 vector masking the features not owned by the algorithm at iteration t, i.e. $\boldsymbol{z}_t \in \{0,1\}^{d'}$ whose components $z_{t,i}$ are equal to:

$$z_{t,i} = \begin{cases} 1 & \text{if } \phi_i^* \in \mathcal{F}_t \\ 0 & \text{if } \phi_i^* \notin \mathcal{F}_t \end{cases}$$

We can now redefine $\boldsymbol{\phi}_t : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^{d'}$ as:

$$\boldsymbol{\phi}_t(x, y) = \boldsymbol{z}_t \odot \boldsymbol{\phi}^*(x, y) \tag{6.2}$$

where $\odot$ denotes the Hadamard (element-wise) product. Given the above definition, we can also derive the following property:

$$\begin{aligned} \boldsymbol{\phi}^*(x, y) - \boldsymbol{\phi}_t(x, y) &= \boldsymbol{\phi}^*(x, y) - \boldsymbol{z}_t \odot \boldsymbol{\phi}^*(x, y) \\ &= (\boldsymbol{1} - \boldsymbol{z}_t) \odot \boldsymbol{\phi}^*(x, y) \end{aligned} \tag{6.3}$$

where $\boldsymbol{1}$ is a vector of the same dimension containing 1 for all components.

In coactive critiquing we observe feedback over the weights through the standard coactive feedback, whereas feedback over the features is observed through critiques. While inference and learning are carried out over the current set of features $\boldsymbol{\phi}_t(\cdot)$, the user improvements will be generated over the full $\boldsymbol{\phi}^*(\cdot)$, according to the usual α-informative feedback model. This mismatch causes the CPP algorithm to "miss" some utility gain over the feature it still does not possess. More specifically, we quantify this utility mismatch as:

$$\zeta_t = \langle \boldsymbol{w}^*, \boldsymbol{\phi}^*(x_t, \bar{y}_t) - \boldsymbol{\phi}^*(x_t, y_t)\rangle - \langle \boldsymbol{w}^*, \boldsymbol{\phi}_t(x_t, \bar{y}_t) - \boldsymbol{\phi}_t(x_t, y_t)\rangle \tag{6.4}$$

Using Equation 6.2 and Equation 6.3 we can rewrite the ζ_t quantity as follows:

$$\begin{aligned} \zeta_t &= \langle \boldsymbol{w}^*, \boldsymbol{\phi}^*(x_t, \bar{y}_t) - \boldsymbol{\phi}^*(x_t, y_t)\rangle - \langle \boldsymbol{w}^*, \boldsymbol{\phi}_t(x_t, \bar{y}_t) - \boldsymbol{\phi}_t(x_t, y_t)\rangle \\ &= \langle \boldsymbol{w}^*, \boldsymbol{\phi}^*(x_t, \bar{y}_t) - \boldsymbol{\phi}^*(x_t, y_t) - \boldsymbol{z}_t \odot \left(\boldsymbol{\phi}^*(x_t, \bar{y}_t) - \boldsymbol{\phi}^*(x_t, y_t)\right)\rangle \\ &= \langle \boldsymbol{w}^*, (\boldsymbol{1} - \boldsymbol{z}_t) \odot \left(\boldsymbol{\phi}^*(x_t, \bar{y}_t) - \boldsymbol{\phi}^*(x_t, y_t)\right)\rangle \end{aligned} \tag{6.5}$$

The quantity ζ_t appears on the following regret bound, explaining the slow down in the convergence rate of the algorithm due to the fact that the algorithm starts with a lower dimensional feature space than the one used by the user. Finally, let $\|\boldsymbol{\phi}^*(\cdot)\| \leq R$.

Theorem 6.3.1. *For a user with true utility parameters $\boldsymbol{w}^*$, under the α-informative feedback assumption*[2] *(Equation 3.12), the average regret incurred by the CPP algorithm is upper bounded by:*

$$\frac{1}{T}\sum_{t=1}^{T} \textit{REG}(x_t, y_t) \leq \frac{2R\|\boldsymbol{w}^*\|}{\alpha\sqrt{T}} + \frac{1}{\alpha T}\sum_{t=1}^{T}(\xi_t + \zeta_t) \tag{6.6}$$

Proof. We begin by finding an upper bound to the following dot product by applying the Cauchy-Schwarz inequality:

$$\langle \boldsymbol{w}^*, \boldsymbol{w}_{T+1}\rangle \leq \|\boldsymbol{w}^*\|\|\boldsymbol{w}_{T+1}\| = \|\boldsymbol{w}^*\|\sqrt{\langle \boldsymbol{w}_{T+1}, \boldsymbol{w}_{T+1}\rangle} \tag{6.7}$$

By unrolling the term $\langle \boldsymbol{w}_{T+1}, \boldsymbol{w}_{T+1}\rangle$:

$$\begin{aligned}
\langle \boldsymbol{w}_{T+1}, \boldsymbol{w}_{T+1}\rangle &= \langle \boldsymbol{w}_T, \boldsymbol{w}_T\rangle + 2\langle \boldsymbol{w}_T, \boldsymbol{\phi}_T(x_T, \bar{y}_T) - \boldsymbol{\phi}_T(x_T, y_T)\rangle + \\
&\qquad \langle \boldsymbol{\phi}_T(x_T, \bar{y}_T) - \boldsymbol{\phi}_T(x_T, y_T), \boldsymbol{\phi}_T(x_T, \bar{y}_T) - \boldsymbol{\phi}_T(x_T, y_T)\rangle \\
&\leq \langle \boldsymbol{w}_T, \boldsymbol{w}_T\rangle + \|\boldsymbol{\phi}_T(x_T, \bar{y}_T) - \boldsymbol{\phi}_T(x_T, y_T)\|^2 \\
&\leq \langle \boldsymbol{w}_T, \boldsymbol{w}_T\rangle + 4R^2 \\
&\leq 4R^2 T
\end{aligned}$$

The second inequality follows from the optimality of y_T with respect to $\boldsymbol{w}_T$, while the last equality follows from the fact $\|\boldsymbol{\phi}_T(x, y)\| \leq \|\boldsymbol{\phi}^*(x, y)\| \leq R$. Combining the above inequality with Equation 6.7 we get:

$$\begin{aligned}
\langle \boldsymbol{w}^*, \boldsymbol{w}_{T+1}\rangle &\leq \|\boldsymbol{w}^*\|\sqrt{4R^2 T} \\
&= 2R\|\boldsymbol{w}^*\|\sqrt{T}
\end{aligned} \tag{6.8}$$

Expanding the LHS with the update rule (line 11) we get:

$$\langle \boldsymbol{w}^*, \boldsymbol{w}_{T+1}\rangle = \langle \boldsymbol{w}^*, \boldsymbol{w}_T\rangle + \langle \boldsymbol{w}^*, \boldsymbol{\phi}_T(x_T, \bar{y}_T) - \boldsymbol{\phi}_T(x_T, y_T)\rangle$$

Applying Equation 6.2 and Equation 6.3 on the remaining term we get:

$$\begin{aligned}
\langle \boldsymbol{w}^*, \boldsymbol{w}_{T+1}\rangle &= \langle \boldsymbol{w}^*, \boldsymbol{w}_T\rangle + \langle \boldsymbol{w}^*, \boldsymbol{z}_T \odot (\boldsymbol{\phi}^*(x_T, \bar{y}_T) - \boldsymbol{\phi}^*(x_T, y_T))\rangle \\
&= \langle \boldsymbol{w}^*, \boldsymbol{w}_T\rangle + \langle \boldsymbol{w}^*, \boldsymbol{\phi}^*(x_T, \bar{y}_T) - \boldsymbol{\phi}^*(x_T, y_T)\rangle - \\
&\qquad \langle \boldsymbol{w}^*, (\boldsymbol{1} - \boldsymbol{z}_T) \odot (\boldsymbol{\phi}^*(x_T, \bar{y}_T) - \boldsymbol{\phi}^*(x_T, y_T))\rangle
\end{aligned}$$

Substituting the definition of ζ_t (Equation 6.5) and unrolling the recursion we get:

$$\begin{aligned}
\langle \boldsymbol{w}^*, \boldsymbol{w}_{T+1}\rangle &= \langle \boldsymbol{w}^*, \boldsymbol{w}_T\rangle + \langle \boldsymbol{w}^*, \boldsymbol{\phi}^*(x_T, \bar{y}_T) - \boldsymbol{\phi}^*(x_T, y_T)\rangle - \zeta_T \\
&= \sum_{t=1}^{T}\langle \boldsymbol{w}^*, \boldsymbol{\phi}^*(x_t, \bar{y}_t) - \boldsymbol{\phi}^*(x_t, y_t)\rangle - \sum_{t=1}^{T}\zeta_t
\end{aligned}$$

[2]Recall the α-informative model: $u^*(x_t, \bar{y}_t) - u^*(x_t, y_t) = \alpha(u^*(x_t, y_t^*) - u^*(x_t, y_t)) - \xi_t$.

Applying the definition of the α-informative feedback (Equation 3.12):

$$\langle \boldsymbol{w}^*, \boldsymbol{w}_{T+1}\rangle = \alpha \sum_{t=1}^{T} \langle \boldsymbol{w}^*, \boldsymbol{\phi}^*(x_t, y_t^*) - \boldsymbol{\phi}^*(x_t, y_t)\rangle - \sum_{t=1}^{T} \xi_t - \sum_{t=1}^{T} \zeta_t$$

Plugging the above form into Equation 6.8 we get:

$$\alpha \sum_{t=1}^{T} \langle \boldsymbol{w}^*, \boldsymbol{\phi}^*(x_t, y_t^*) - \boldsymbol{\phi}^*(x_t, y_t)\rangle - \sum_{t=1}^{T} \xi_t - \sum_{t=1}^{T} \zeta_t \leq 2R\|\boldsymbol{w}^*\|\sqrt{T}$$

Rearranging:

$$\begin{aligned}\alpha \sum_{t=1}^{T} \langle \boldsymbol{w}^*, \boldsymbol{\phi}^*(x_t, y_t^*) - \boldsymbol{\phi}^*(x_t, y_t)\rangle &\leq 2R\|\boldsymbol{w}^*\|\sqrt{T} + \sum_{t=1}^{T} (\xi_t + \zeta_t) \\ \alpha \sum_{t=1}^{T} \text{REG}(x_t, y_t) &\leq 2R\|\boldsymbol{w}^*\|\sqrt{T} + \sum_{t=1}^{T} (\xi_t + \zeta_t)\end{aligned}$$

Dividing by αT we obtain the claim. □

Note that:

$$\begin{aligned}\frac{1}{T}\sum_{t=1}^{T} \zeta_t &\leq \frac{\|\boldsymbol{w}^*\|}{T} \sum_{t=1}^{T} \|(\mathbf{1} - \boldsymbol{z}_t) \odot (\boldsymbol{\phi}^*(x_t, \bar{y}_t) - \boldsymbol{\phi}^*(x_t, y_t))\| \\ &\leq \frac{\|\boldsymbol{w}^*\|}{T} \sum_{t=1}^{T} \|\mathbf{1} - \boldsymbol{z}_t\|_0 \|\boldsymbol{\phi}^*(x_t, \bar{y}_t) - \boldsymbol{\phi}^*(x_t, y_t)\|_\infty \\ &\leq \frac{2D\|\boldsymbol{w}^*\|}{T} \sum_{t=1}^{T} \|\mathbf{1} - \boldsymbol{z}_t\|_0\end{aligned}$$

With $D = \max_{y \in \mathcal{Y}} \|\boldsymbol{\phi}^*(x, y)\|_\infty$. This means that, as new features get acquired, the norm $\|\mathbf{1} - \boldsymbol{z}_t\|_0$ decreases with t, and thus the average $\frac{1}{T}\sum_{t=1}^{T} \zeta_t$ is guaranteed to converge to 0.

6.4 Experiments

We evaluated the CPP algorithm over two constructive settings, a synthetic task and a travel plan recommendation problem. All experiments were run by sampling 20 complete user weight vectors from a standardized normal distribution and simulating their behavior according to these vectors. Improvements were simulated using the α-informative feedback model as in coactive learning (Section 3.3). Critiques were simulated by selecting the most discriminative feature for a given pair $(y_t, \bar{y}_t)$, accounting for some noise in the selection. For both settings, the problem domain was encoded via MiniZinc [188] and inference was solved using Gecode. Please refer to [255] for a complete description of the experimental setup.

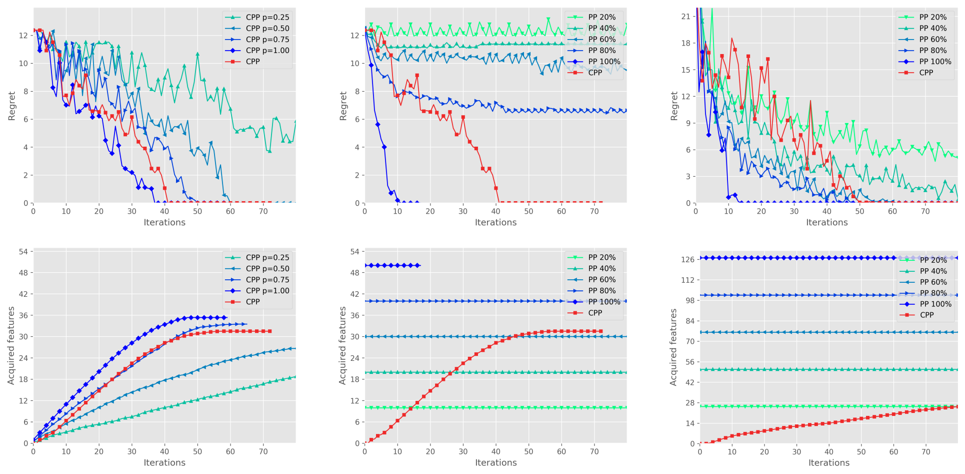

Figure 6.1: Left: comparison of CPP for different choices of NeedCritique procedure; median utility loss at the top, average number of acquired features at the bottom. Middle: comparison between CPP and PP on the synthetic problem. Right: comparison between CPP and PP on the trip planning problem. Best viewed in color.

The first experimental setting we considered is a synthetic constructive problem, in which configurations are represented by all points in a 100×100 lattice (10^4 feasible configurations). Features are represented as rectangles inside the lattice, and 50 of them were sampled uniformly at random. The corresponding formula in the feature vector $\boldsymbol{\phi}$ is an indicator function ($\phi(y) = \{-1, 1\}$) of whether the point corresponding to the configuration y is included in the rectangle. A positive weight w_i associated to a feature ϕ_i corresponds to a preference of the user for points inside the corresponding rectangle, whereas a negative weight corresponds to the user disliking points inside the rectangle.

The synthetic setting was used for two experiments. We first tested the performance of our formulation for the function NeedCritique, which, as said, consists in querying the user for a critique when the current feature space cannot represent all the ranking pairs in $\mathcal{D}$. This strategy was compared against a random strategy consisting in asking critique queries at random iterations, according to a binomial distribution with $p \in \{0.25, 0.5, 0.75, 1\}$. The left plots of Figure 6.1 show the regret (top) and the acquired features (bottom) of CPP using the different possible criteria for NeedCritique. The results show that the separability criterion is a good trade-off, performing almost as well as asking a critique at each iteration, while asking for roughly 25% less critiques. We employed this criterion in all the following experiments.

In the second experiment, we compared CPP to the standard preference perceptron (PP) algorithm (see Section 3.3.1). CPP started with 2 randomly selected user features and acquired the others throughout the elicitation process, whereas PP^p started with a fixed percentage $p\%$ of (randomly selected) user features and did not acquire more. The middle plots in Figure 6.1 report the regret and the acquired features of CPP and PP^p, with $p \in \{20, 40, \ldots, 100\}$.

The plots show that PP^{100} clearly converges much faster than all other settings, which was expected given that it uses all the user features. CPP also converges, albeit in a few more iterations, while PP does not converge at all if not provided with the entire set of features. Notice that CPP manages to converge to an optimal solution by eliciting only roughly 60% of the user features.

The same experimental setting was used for a more realistic (and complicated) scenario consisting of an interactive travel plan problem. A trip is represented as a sequence of time slots, each one fillable with some activity in some city or by traveling between cities. The algorithm is in charge or recommending a trip y between a subset of the cities, along with the activities planned for each time-slot in each city. A subset of activities is available in each city. The trip has a maximum of usable time-slots. In our experiments the trip length was fixed to 10. User features include, e.g. the time spent in each city and the time spent doing each activity, the number of visited cities, etc. The total number of acquirable features is 92. As for the synthetic experiment, we ran CPP and PP^p, averaging the results over 20 randomly selected users. The results are shown in the right plots of Figure 6.1. Even in this complex scenario CPP outperforms PP^{40} and is competitive against the much more informed PP^{60} and PP^{80}, by converging in roughly the same amount of iterations. This experiment shows that CPP is very effective despite using a fraction of the user information. Indeed, it ends up using less than 20% of the acquirable user features, even less of PP^{20} and PP^{40} that fail to converge after 100 iterations.

6.5 Summary

In this chapter we introduced and detailed our coactive critiquing approach. This methodology aims at relaxing the assumption of a universal feature map ϕ made by standard coactive learning, allowing to *customize* the utility model to the different needs of each user. Coactive critiquing extends coactive learning by dynamically expanding the feature space through example critiquing. As critiques are elicited from the user, the utility becomes increasingly more expressive, up to the point of enabling the algorithm to converge to an optimal solution. Our proposed algorithm, the critiquing preference perceptron, is able to converge while asking critiques only when necessary, i.e. when the current feature space of the algorithm does not suffice to represent all the ranking pairs collected so far. In this chapter, we also proved that our algorithm enjoys a $\mathcal{O}(1/\sqrt{T})$ bound on its average regret, only slowed down by a term dependent on the features the algorithm misses, which vanishes to zero as new critiques are acquired. Finally, we presented experimental evidence that shows that the critiquing preference perceptron is competitive with standard coactive learning, while requiring many fewer features to reach an optimal solution.

Part III

Applications

Chapter 7

Automated layout synthesis

In this chapter we apply the constructive preference elicitation framework to the class of design problems called *layout synthesis*. Layout synthesis refers to the task of generating layouts, i.e. arrangements of objects within a fixed 2D or 3D space. Examples of layout synthesis tasks are designing the interior of a room, create the blueprint of an apartment or plan the disposition of buildings in a block- or city-sized urban space. Many of these tasks can be seen as preference-based decision problems over a constrained combinatorial space. Indeed, the solutions to all these problems are restricted by functional and structural requirements, while at the same time largely dependent on the designer or end user style and taste. Solving problems of such complexity is not an easy task, and even experts may find certain layout synthesis problems especially challenging. Automated layout synthesis systems can greatly enhance the productivity of experts in their design work, as well as improving the outcome quality for non-experts users.

Layout synthesis tasks are combinatorial in nature, and as such can be expressed very naturally as constructive preference elicitation problems. A constructive layout synthesis problem can be defined by representing layouts as a collection of variables describing the properties of the objects involved (position, size, type, etc.), and by imposing constraints to encode the normative and engineering requirements, as well as any applicable human design guideline [6, 199] (visibility, accessibility, *etc.*).

We propose to use coactive feedback to learn user preferences in constructive layout synthesis tasks. Using a kind of *manipulative* feedback, the coactive interaction appears to be a natural choice for an interactive layout synthesis system, as it could be seamlessly integrated within CAD-like software.

The concept of incrementally improving layouts through manipulative interaction has already been proposed in previous work [182, 184]. We also borrow from previous work the idea of representing layout synthesis tasks as constrained combinatorial optimization programs [185]. The novelty of our approach consists in combining interaction, preference learning and combinatorial optimization to solve layout synthesis tasks.

An important advantage of our approach over the others found in the literature is the ability of a constructive layout synthesis system to generalize across different contexts and sets of constraints. This implies re-usability of the same model to different instances of the same problem or similar problems with different requirements. A use case for this feature is to employ a constructive layout synthesis system to learn to furnish apartments according to a user taste and reuse it to automatically furnish different apartments. For instance, a customer wishing to buy an apartment from a set of alternatives might want to evaluate each candidate by simulating the end result of filling in missing furniture through a layout synthesis system. While many existing tools may assist the user in this task [182, 184], none of them learn to generalize the user preferences across different apartments, forcing the user to repeat this tedious task for each of them separately. A constructive system would instead be able to learn from the first interaction with the user and then automatically furnish the rest of the apartments based on the estimated preferences and the new constraints.

In this work, first published in [82] and later extended in [93], we applied our constructive layout synthesis to two specific tasks: arranging furniture in a room and planning the layout of an apartment. We also tested two different ways to implement coactive feedback. In the furniture arrangement task we allow the user to adjust the values of features of interest (e.g. the average distance between tables in a cafè), while in the floor planning task the user is allowed to modify the layout directly by, for instance, moving or removing walls. In our experiments we test our system over increasingly more complex instances in terms of feedback quality and computational cost, showing that it is able to effectively learn and recommend better layouts over time in many different scenarios. Here we also explore the use of approximation techniques and we found that the algorithm can still reliably learn the user preferences over time, providing good trade-offs between recommendation quality and inference speed can be found. Finally, we also evaluate the ability of the algorithm to learn to generate proper layouts in very different scenarios and deal with users with very different preferences.

The rest of the chapter is organized as follows. In Section 7.1 we place our proposed technique with the prolific literature on design aiding systems, describing differences with other proposed techniques. In Section 7.2 we describe our approach and simulation strategy. In Section 7.3 we detail our two experimental settings and point out the results. Section 7.4 concludes the chapter.

7.1 Design aiding systems

The goal of this work is to investigate the feasibility of a *design aiding system* capable of creating and suggesting custom layouts, a task that requires solving two distinct problems: generating a layout consistent with the known requirements and preferences (*synthesis*), and biasing the synthesis process toward layouts preferred by the user (*customization*).

Broadly speaking, synthesis can be solved in two ways, namely *sampling* and *optimization*. The first consists in designing a parameterized distribution over layouts (e.g. a probabilistic graphical model [284] or a probabilistic grammar [167]), whose structure encodes the set of validity constraints on objects and arrangements. Synthesis equates to sampling from the distribution via Monte Carlo Markov Chain methods. A major downside of probabilistic approaches is that enforcing hard constraints (other than those implied by the structure of the distribution) may severely degrade the performance of the sampler, potentially compromising convergence, as discussed for instance in [259]. The alternative strategy, also adopted by our method, is to define a scoring function that ranks candidate layouts based on the arrangement of their constituents. In this case, synthesis amounts to finding a high-scoring layout subject to design and feasibility constraints. This optimization problem may be solved using stochastic local search [3, 285], mathematical optimization [184], or constraint programming [185]. Our constructive preference elicitation technique is based on the latter: constraint programming [228] allows to easily encode expressive local and global constraints, and is supported by many efficient off-the-shelf solvers. Further, in many cases it is easy to instruct the solver to look for (reasonably) suboptimal solutions, allowing to trade-off solution quality for runtime, for enhanced scalability. This synergizes with our learning method, which is robust against approximations, both theoretically [248] and experimentally (see Section 7.3). Many of the existing tools are concerned with synthesis only, and do not include a customization step: their main goal is to automate procedural generation of realistic-looking scenes or to produce concrete examples for simplifying requirement acquisition from customers [282]. Other approaches bias the underlying model (distribution or scoring function) toward "good" layouts by fitting it on sensibly furnished examples [182, 284, 285]. However, the generated configurations are not customized for each user[1]. More generally, offline model estimation may be used in conjunction with our method to accelerate layout fine-tuning for the end user.

Akase and colleagues proposed two interactive methods based on iterative evolutionary optimization [3, 4]. Upon seeing a candidate furniture arrangement, the user can tweak the fitness function either directly using sliders [3] or indirectly via conjoint analysis [4]. In both works the number of customizable parameters is small and not directly related to the scene composition (e.g. illumination, furniture crowdedness). Contrary to these methods, we enable the user to graphically or physically manipulate a proposed layout to produce an improved one. This kind of interaction was successfully employed by a number of systems [182, 184, 262] using ideas from direct manipulation interfaces [131, 251]. We stress that our method works even if user improvements are small, as shown by our empirical tests. The major difference to the work of Akase *et al.*, however, is that we leverage constraint programming rather than generic evolutionary optimization algorithms. This enables our method naturally handle arbitrary feasibility constraints on the synthesized layouts, extending its applicability to a variety of layout synthesis settings. Among interactive methods, the one of Merrell *et al.* [182] is the closest to ours. Both methods rely on a scoring function, and both require the user and system to interact by suggesting modifications to each other. In contrast to our approach, the method of Merrel *et al.* does not learn the scoring function in response to the user suggestions, i.e., it will always suggest configurations in line with fixed design guidelines. Since no user model is learned, this method does not allow transferring information across distinct design session.

[1]While the examples may be provided by the end user, it is unreasonable to expect the latter to manually select the large number of examples required for fine-grained model estimation. Through interaction, our system allows a more direct control over the end result.

7.2 Constructive layout synthesis

In this work we instantiate the preference perceptron algorithm to solve two different constructive layout synthesis tasks: [i] a furniture arrangement problem, in which a user has to decide which pieces of furniture to place in a room and how to arrange them; [ii] a floor planning problem, in which a floor (2D space) needs to be partitioned into rooms (subspaces).

As usual, we represent the user preferences with a utility function $u(x, y) = \langle \boldsymbol{w}, \boldsymbol{\phi}(x, y)\rangle$. In both types of layout synthesis problems, contexts $x \in \mathcal{X}$ will be the user provided floor area, while predicted objects will be the furniture arrangement and the floor partitioning, respectively. The structure of the layouts is then defined by a set of hard constraints $\mathcal{Y}(x)$, possibly also dependent on the context, while the adjustable components of the layouts $y \in \mathcal{Y}(x)$ are described by the features $\boldsymbol{\phi}(x, y)$. The layout y is encoded with a set of variables representing its properties, e.g. the position and size of the pieces of furniture in the furniture arrangement problem, or the assignment of the unit squares to different rooms in the floor planning one. Hard constraints include e.g. non-overlapping pieces of furniture and unique room type assignments. The feature vector $\boldsymbol{\phi}(x, y)$ contains, for example, the distance between the pieces of furniture, or the number of rooms of a certain type. We will give details about the full formulation of the two tasks in the following sections.

Inference at each iteration $t \in [T]$ is handled with the regular utility-maximizing structured prediction from coactive learning:

$$y_t = \operatorname*{argmax}_{y \in \mathcal{Y}(x_t)} \langle \boldsymbol{w}_t, \boldsymbol{\phi}(x_t, y)\rangle$$

We use a mix of Boolean, integer and floating point variables, with linear features and constraints, making inference a MILP problem, solved by external solvers.

For problems that are too complex to afford an exact solution, we used an approximation technique to speed-up the inference process. Reasonably suboptimal synthesized layouts do not significantly alter the performance of coactive learning, as proven theoretically in [216]. In our experiment we test this strategy and show empirically the advantage and disadvantage of this approach.

As mentioned, in this work we also tested two different feedback approaches, exemplified in Figure 7.1. In the first problem of furniture arrangement, we use a feature-based feedback, in which a user can produce improvements to the current object by tweaking the value of one or more features, using a UI composed of e.g. sliders for numeric features and switches for Boolean ones. In the top example of Figure 7.1, a user has improved the left configuration by increasing a feature encoding the distance between the tables, resulting in the top right configuration.

To simulate this type of behavior we follow the α-informative feedback model, and we assume a “minimal-effort” feedback, which translates into an improvement $\bar{y}_t$ that follows the α-informative rule, but has a minimal change in terms of features. To select a improvement

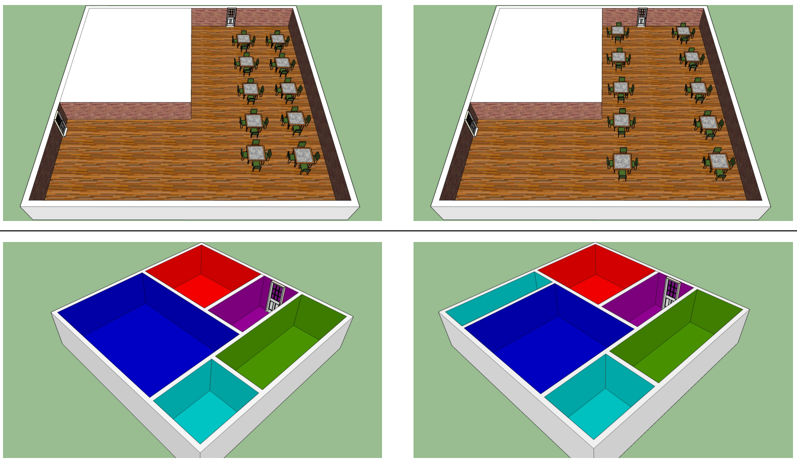

Figure 7.1: Examples of visual improvement that a user may perform. An image of a configurations y proposed by the system (left) is followed by an image of an improvement $\bar{y}$ made by the user (right). For the furniture arrangement setting (top), there is a feature-level improvement, e.g. the user sets the minimum distance between tables (a feature) to a higher value. The floor planning case (bottom) has, instead, an object-level improvement, e.g. the user manually modifies the object shape by adding a new wall. Best viewed in color.

following this approach we solve the following optimization problem:

$$\begin{aligned}\bar{y}_t &= \operatorname*{argmin}_{y \in \mathcal{Y}(x_t)} \|\boldsymbol{\phi}(x_t, y) - \boldsymbol{\phi}(x_t, y_t)\|_0 \\ \text{s.t.} \quad & u^*(x_t, y) - u^*(x_t, y_t) \geq \alpha(u^*(x_t, y_t^*) - u^*(x_t, y_t))\end{aligned}$$

That is we select the improvement by finding the object with minimum number of changed features that satisfies the α-informative feedback model.

The second type of feedback considered is an object-based feedback, in which a user can directly change the object itself, i.e. modifying the value of some of the object variables. The bottom part of Figure 7.1 shows this type of feedback applied to the floor planning task. On the left, the starting configuration, and on the right the user improvement, in which she directly modified the layout of the rooms adding a new room by splitting an existing one. To simulate this feedback we again select the minimal-effort α-informative improvement by solving the problem:

$$\begin{aligned}\bar{y}_t &= \operatorname*{argmin}_{y \in \mathcal{Y}(x_t)} \Delta(y, y_t) \\ \text{s.t.} \quad & u^*(x_t, y) - u^*(x_t, y_t) \geq \alpha(u^*(x_t, y_t^*) - u^*(x_t, y_t))\end{aligned}$$

where $\Delta(y, y_t)$ is a custom distance measure between two objects, dependent on the task at hand. We will give details on the one used for the floor planning task in Section 7.3.2. By minimizing the distance $\Delta(y, y_t)$ we simulate a minimal effort improvement.

7.3 Experiments

We evaluated our system on two different tasks. In the first experiment, the system recommends table arrangements in a room, for e.g. a bar or an office. The second experiment consists in a space partitioning task, in which the system suggests how to partition the surface of an apartment into rooms. In both settings we make a quantitative evaluation, i.e. we compare the regret of the system for increasing levels of problem complexity. As the problem complexity increases, approximate solutions become necessary to keep real-time interaction. As an approximate inference heuristic we set a time cut-off to the solver and return the best solution found in that time. Increasing the time cut-off will result in a higher utility solution but increases the inference time. We evaluate empirically the effect of using approximate inference on the quality of the recommendations in both settings, providing trade-offs between inference time and loss in recommendation quality.

All the quantitative experiments were run over 20 randomly generated users and averaged over them. The user responses were simulated following the α-informative feedback model with assumed minimal effort as described in the previous section. Varying the α parameter we can describe different levels of user expertise in providing good improvements over the system recommendation. We also assume no user expertise required to interact usefully with our system, and thus we set $\alpha = 0.1$ to simulate a non-expert user. For a larger α parameter the performance of the algorithm would increase proportionally.

In both experimental settings, we also report a qualitative evaluation showcasing the behavior of the system in interacting with some “prototypical” type of user (e.g. a cafè owner arranging the tables in her place). We show that the system achieves the goal of finding good configurations matching the user taste.

The system is implemented in Python[2] and uses MiniZinc to model the constrained optimization problems [188], and an external MILP solver[3] for inference and improvement problems. All the experiments were run on a 2.8 GHz Intel Xeon CPU with 8 cores and 32 GiB of RAM.

7.3.1 Furniture arrangement

In the first experimental setting, the goal of the system is to learn to arrange tables in a room according to the user preferences. Rooms are 2D spaces of different shapes. We model the rooms as squared bounding boxes, plus several inaccessible areas making up internal walls. The size of the bounding box and the inaccessible areas are given in the context x, together with the number of tables to place. The available space is discretized into unit squares of fixed size. Tables are rectangles of different shape occupying one or more unit squares. The output objects y consist in the table arrangements in the given room. More precisely, tables are represented by their bottom-left coordinates (h, v) in the bounding box and their sizes (dh, dv) in horizontal and vertical directions. The object y contains the coordinates (h_t, v_t)

[2]Code available at: `github.com/unitn-sml/constructive-layout-synthesis`
[3]Opturion CPX: `opturion.com`

	Furniture arrangement
Context x	- Size of bounding box - Inaccessible areas - Position of doors - Number of tables
Object y	- Position (h, v) of all tables - Sizes (dh, dv) of all tables
Features $\phi(x, y)$	- Max and min distance of tables from bounding box: $\max_{t \in \textit{Tables}} \textit{bbdist}(t)$ $\min_{t \in \textit{Tables}} \textit{bbdist}(t)$ - Max and min distance of tables from inaccessible areas: $\max_{t \in \textit{Tables}} \textit{wdist}(t)$ $\min_{t \in \textit{Tables}} \textit{wdist}(t)$ - Max and min distance between tables: $\max_{t_1, t_2 \in \textit{Tables}} \textit{dist}(t_1, t_2)$ $\min_{t_1, t_2 \in \textit{Tables}} \textit{dist}(t_1, t_2)$ - Number of tables per type (1×1 and 1×2): $\lvert\{t \in \textit{Tables} \mid dh_t + dv_t \leq 2\}\rvert$ $\lvert\{t \in \textit{Tables} \mid dh_t + dv_t \geq 3\}\rvert$

Table 7.1: Summary of the structure of the objects in the furniture arrangement settings. In this task, *Tables* is the set of tables, *bbdist*(t) is the distance of table t from the bounding box, *wdist*(t) is the distance of table t from the inaccessible areas (walls), *dist*(t_1, t_2) is distance between tables t_1 and t_2. All distances considered here are Manhattan distances.

and the sizes (dh_t, dv_t) of each table t. Several constraints are imposed to define the feasible configurations. Tables are constrained to fit all in the bounding box, to not overlap, and to not be positioned in unfeasible areas. Tables must keep a minimum "walking" distance between each other. Doors are also placed on the room walls (in the context) and tables are required to keep a minimum distance from the doors.

In our experiment the total size of the bounding box is 12×12. Tables are either 1×1 squares (occupying one unit square) or 1×2 rectangles (occupying two unit squares). Room shapes were selected randomly at each iteration from a pool of five candidates.

The feature vector $\phi(x, y)$ is composed of several numeric properties of the configuration, such as the maximum and minimum distance between tables, the maximum and minimum distance between tables and walls, and the number of tables per type (1×1 and 1×2). The first column of Table 7.1 contains a detailed summary of the structure of x, y and $\phi(x, y)$ in this setting.

As mentioned in the previous section, in this setting we employ a feature-based improvement scheme to simulate a minimal effort α-informative user behavior.

	Floor planning
Context x	- Size of bounding box - Inaccessible areas - Position of entrance door - Max and min rooms per type
Object y	- Position (h, v) of all rooms - Sizes (dh, dv) of all rooms - Type t_r of each room r
Features $\phi(x, y)$	- Ranges of occupied space (percent) per room type: $\forall t \in \textit{Types} \;\; \sum_{r \in R_t} A_r \leq 15\%$ $\forall t \in \textit{Types} \;\; 15\% < \sum_{r \in R_t} A_r \leq 30\%$ $\forall t \in \textit{Types} \;\; \sum_{r \in R_t} A_r > 30\%$ - Upper bound D_r of difference of sides for each room r: $\forall r \in \textit{Rooms} \;\; D_r$ s.t. $\lvert dh_r - dv_r \rvert \leq D_r$ - Number of rooms per type: $\forall t \in \textit{Types} \;\; \lvert R_t \rvert$ - Room with entrance door r_{door} is of type t: $\forall t \in \textit{Types} \;\; t == \textit{type}(r_{\text{door}})$ - Sum of pairwise difference of room areas per type: $\forall t \in \textit{Types} \;\; \sum_{i,j \in R_t} \lvert A_i - A_j \rvert$ - Number of rooms adjacent to corridors $\lvert \{ r \in \textit{Rooms} \mid \exists s \in R_{\text{corridor}} \; \textit{adj}(r, s) \} \rvert$ - Distance of each room from South (bottom edge) $\forall r \in \textit{Rooms} \;\; \textit{sdist}(r)$

Table 7.2: Summary of the structure of the objects in the floor planning experiment. In this setting, *Types* is the set of room types, *Rooms* is the set of rooms, R_t the set of rooms of type t, A_r the area of room r (number of unit squares), dh_r and dv_r the horizontal and vertical size of room r, *type*(r) the type of room r, *adj*(r, s) is a Boolean function denoting the adjacency between rooms r and s, *sdist*(r) is the distance between r and the south edge of the bounding box. All distances considered here are Manhattan distances.

In the quantitative evaluation we run the recommendation algorithm for an increasing number of tables to be placed. A high number of tables makes the inference problem more complex, as it involves more optimization variables and constraints. We test the algorithm on problems with 6, 8 and 10 tables. We compare the average regret and the running time of the system in each of these scenarios. Figure 7.2 shows the median results (over all users) on settings with different number of tables. The plots show the median average regret (top) and the median cumulative inference time (bottom). The first row of Figure 7.2 shows the results for the table arrangement task with exact inference on problems with different numbers of tables. Using exact inference, the difference in regret decay between different levels of complexity is minimal. This means that when the system is able to solve the inference problem to optimality, the complexity of the problem does not affect much the performance of the

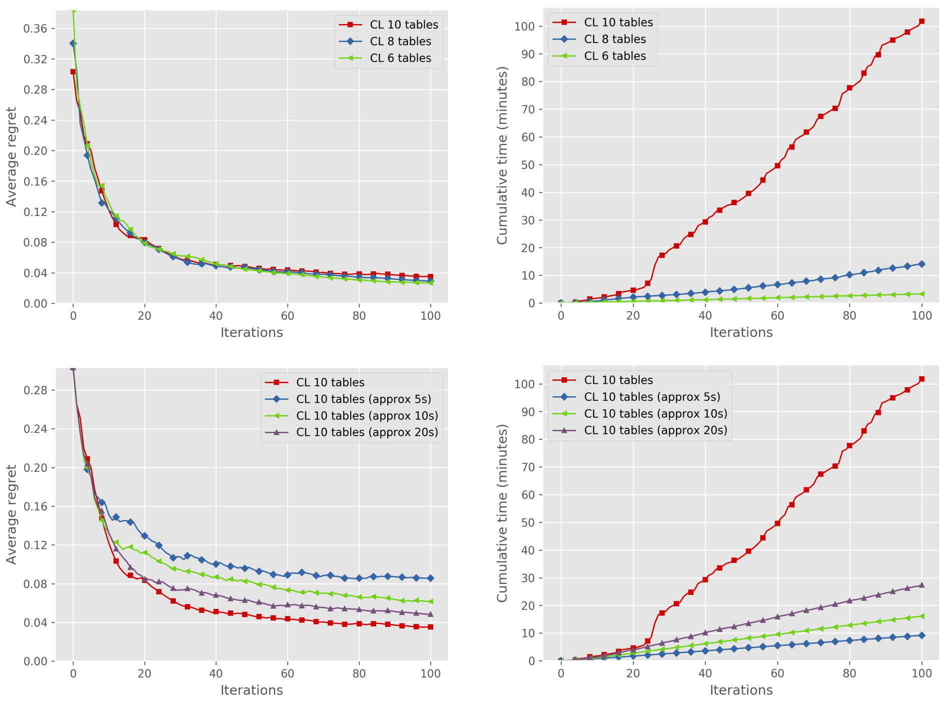

Figure 7.2: Median average regret (left) and median cumulative time (right) of the system in various settings. Top: furniture arrangement setting with exact inference on 6, 8 and 10 tables. Bottom: furniture arrangement settings comparison between exact and approximate inference in the 10 tables problem. Best viewed in color.

system. Inference time, however, increases drastically with the increasing complexity. Exact inference in the 10 tables setting is already largely impractical for an interactive system. The second row of Figure 7.2 shows a comparison of the results of exact and approximate inference on the furniture arrangement setting with 10 tables, for time cut-offs at 5, 10 and 20 seconds[4]. When using approximate inference, the running times drop to a much lower rate, while the regret suffers a slight increase but keeps decreasing at a similar pace as the exact variant. We can see that the time cut-off can be modulated to achieve the desired balance between recommendation quality and inference time. This is a promising behavior suggesting that the method can scale with the problem size with predictable running time without compromising performance.

To get a visual grasp of the quality of the recommendations, we also evaluated our system on two prototypical arrangement problems, featuring a user interested in furnishing a café and another one wishing to furnish an office. Cafés are usually furnished with small tables (1×1), positioned along the walls in a regular fashion. Offices, instead, contain mostly desks (1×2) positioned along the walls or in rows/columns across the room. We sampled two users

[4]The time cut-off is on the solver time, but the actual inference time has some more computational overhead, taking on average 2.21 seconds more.

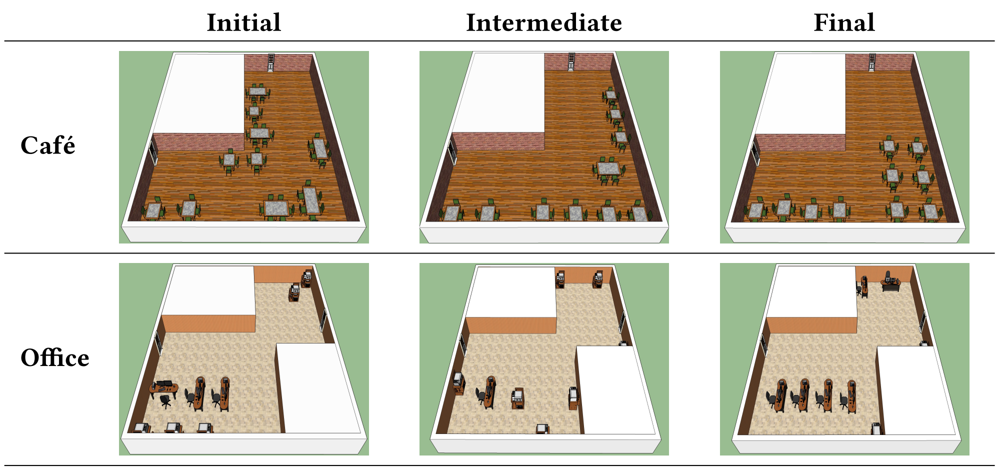

Figure 7.3: Two use cases of our system. The images are 3D renderings of configurations recommended by our system when interacting with users whose goal is to furnish a café (top) and an office (bottom). Horizontally, the figures/layout/layout-synthesis show different stages of the elicitation process. In the café, 1×1 and 1×2 tables are seen as dining tables of different sizes, whereas in the office 1×2 tables represent desks while 1×1 tables contain utilities such as printers. Best viewed in colors.

according to the above criteria. Figure 7.3 showcases the recommendations at different stages of the learning procedure. Initially, tables are randomly spread across the room. Gradually, the system learns to position tables in a more meaningful way. In the café, the intermediate image shows that the algorithm has learned that a café should mostly contain 1×1 tables and they should be placed along the walls. The intermediate figure in the office case shows that the algorithm has roughly figured out the position of tables, but not their correct type. At the end of the elicitation, the final configurations match the user desiderata.

7.3.2 Floor planning

Our second experimental setting is on floor planning, that is recommending partitions of apartments into separate rooms. The outer shape of the apartment is provided by the context, while the user and the system cooperate on the placement of the inner walls defining the room boundaries. As in the previous setting, the space is discretized into unit squares. Each room is a rectangle described by four variables: (h, v) indicate its position, (dh, dv) its size. Coordinates and sizes are measured in unit squares. Rooms must fit in the apartment and must not overlap. Rooms can be of one among five types, namely kitchen, living room, bedroom, bathroom and corridor. In the context, the user can also specify an upper and lower bound on the number of rooms of each type. For instance, a user may look for an apartment with exactly one kitchen, one living room, and between one and two bathrooms and bedrooms. After placing all the rooms, the spaces left in the apartment are considered corridors. The context also specifies the position of the entrance door to the apartment.

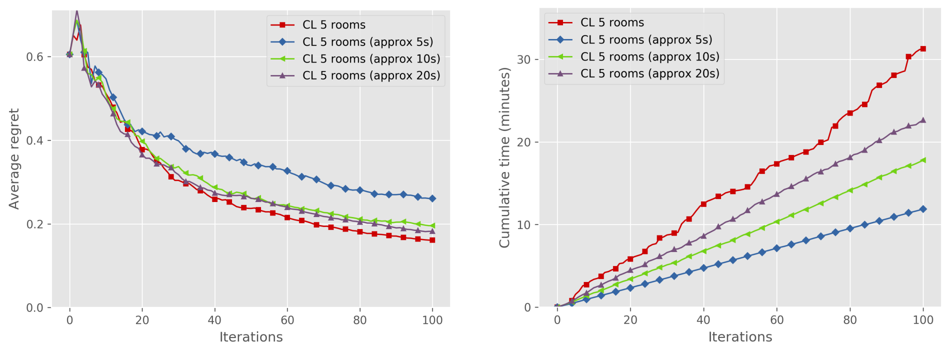

Figure 7.4: Median average regret (left) and median cumulative time (right) of the system in the floor planning setting, comparison between exact and approximate inference. Best viewed in color.

In this experiment we consider a 10×10 bounding box. We define five different apartment shapes and generate random contexts with any combination of room types, summing to a maximum of five rooms, with random lower bounds.

The feature vector $\phi(x, y)$ contains a variable number of features, depending on the maximum number of rooms (sum of upper bounds). Features are normalized in order to generalize different contexts and different numbers of rooms. The features include: [i] the percentage of space occupied by the rooms of each type, discretized in several ranges of values, each denoting a certain target size for each room type; [ii] an upper-bound on the difference between the sides dh_r and dv_r of each room r, which is used to modulate how "squared" the room should be; [iii] the actual number of rooms per type; [iv] a Boolean value for each room type indicating whether the entrance door is in a room of that type; [v] the sum of the pairwise difference between the areas of rooms of the same type, to modulate how similar in size rooms of a certain type should be; [vi] the number of rooms that are adjacent to corridors; [vii] the distance of each room from the South border of the apartment, as living rooms are usually made to look south and bedrooms look north for lighting purposes. A summary of all the features of this setting is listed in the second column of Table 7.2.

Differently from the previous setting, here we employ the object-based improvement schema, with minimal-effort α-informative feedback. The function $\Delta(y, y_t)$ used here is defined as:

$$\Delta(y, y_t) = \|U_y - U_{y^t}\|_0$$

where U_y is the matrix 10×10 containing the room types per unit square. To comply with the minimal-effort principle, we assume a user to perform an α-informative improvement involving the least possible number of rooms, which we simulate by minimizing the number of unit squares affected by the change.

In this case, the problem complexity is mainly given by the maximum number of rooms to be placed in the apartment. Notice that this problem is more difficult than the previous one, as it

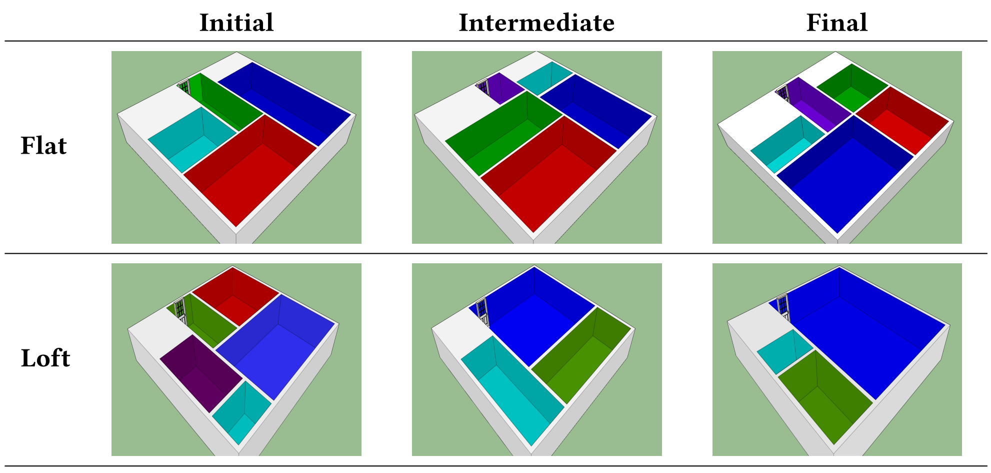

Figure 7.5: Two use cases of our system for the task of floor planning. The images are 3D renderings of configurations recommended by our system when interacting with users whose goal is to build a flat (top) and an loft (bottom). Horizontally, the figures/layout/layout-synthesis show different stages of the elicitation process. Room colors are associated to room types: the kitchen is in red, the living room is in blue, the bathroom is in turquoise, the bedroom in green, the corridor is in violet. Best viewed in colors.

has more optimization variables, more features and it has to learn from a more diverse set of possible contexts. We evaluate this setting only on a scenario with a maximum of five rooms. As in the previous experiment, we report a comparison of the results of exact inference and approximate inference. We again run approximate inference with time cut-offs at 5, 10, and 20 seconds. Figure 7.4 shows the median average regret and the median cumulative inference time in this setting. Both regret and times follow the same trend as the ones in the previous experiment. Approximate inference allows for substantial computational savings[5] at the cost of a small reduction in recommendation quality.

In the qualitative experiment we compare two users who are interested in different kinds of apartments. In the first case, the user is interested in a “traditional” apartment (here dubbed “flat” to avoid ambiguities), which contains a corridor from the entrance door to the rooms, two separate rooms for the kitchen and living room, with the former slightly smaller that the latter, a bedroom and a bathroom. The second user is interested in a loft, which is composed by fewer rooms, usually a big living room with a kitchenette, a bedroom and a bathroom. In Figure 7.5 we can see different stages of the learning process for both users. At the beginning the recommended configurations are random. The system then is able to learn that a flat should have a corridor as entrance and a smaller bathroom, and that a loft should have only a moderately large living room plus a bedroom and a bathroom of approximately equal size. Finally the system reaches good recommendations that meet the preferences of the users: an apartment with a kitchen smaller than the living room and a corridor connecting rooms, and a loft with a big living room, a bedroom and a small bathroom.

[5]Exact inference becomes impractical for more than five rooms.

7.4 Summary

In this chapter we presented a method to devise design aiding systems for layout synthesis, based on our constructive recommendation framework. The layout synthesis tasks, such as furniture arrangement and floor planning, are cast as constructive preference elicitation problems, in which inference over the components of the layouts (such as the position of the tables and the size of the rooms) and their relative structural and functional constraints is encoded as a MILP problem.

Differently from other approaches seen in Section 7.1, our constructive layout synthesis approach learns the user utility function and personalizes the created layouts, while also generalizing the learned preferences across different design sessions. Also, the use coactive learning [247] in our constructive layout synthesis approach seems a natural choice in such a design environment, where explicit manipulative feedback is abundant and easy to attain.

We tested our approach in two different layout synthesis tasks, namely furniture arrangement and floor planning, and evaluated it on instances of increasing complexity. The results show that coactive learning is an effective learning method for our constructive layout synthesis systems and that it was robust to suboptimal inference, allowing us to scale to larger synthesis problems at the cost of a minor degradation of recommendation quality. We also showcased the flexibility of our system by learning from users with radically different preferences, e.g., users that prefer lofts to highly partitioned apartments and vice-versa.

Chapter 8

Product and service bundling

In this chapter we apply constructive preference elicitation to another class of well known problems, namely product and service bundling, and in particular bundling of telephone connectivity and entertainment services. Service bundling has emerged as an important marketing tool for the telecommunication industry, as in many other sectors [111, 112, 136, 156]. Bundling in telco market consists in combining different product and services into one package sold at a single fixed price, usually paid in monthly installments. Nowadays, telco bundles may include mobile and home connectivity services, calls and messages, as well as subscription to third party services for streaming of music and movies, and other entertainment services. Offers may also include electronic devices and other expensive items, which can be purchased exploiting the recurrent structure of payments. The inclusion of many subscriptions and devices into a single plan is of great convenience for the customer and supports his or her retention.

The typical strategy telco operators use to market these bundles is to prepare a small amount of base packages, targeted to a certain group of consumers (young, family, business, etc.), and allow the customers to purchase some extra services on top of those. This approach, while widely spread and quite effective, might be relatively suboptimal, both for the customer and the company. Customization has been shown to be an important factor in customer satisfaction and often increases retention and brand loyalty [59, 88, 254, 260]. Creating a custom service plan from scratch is, however, not an easy task for a user, as it involves deciding within a large number of options for each component of the plan, and there might be some feasibility constraints from the company side that the user would not be aware of. Product configurators [138] and recommendation systems [221] can greatly help a customer in such endeavor.

In Section 2.2.2, we have highlighted the disadvantages of standard product configurators in the context of decision aiding for non-expert users. More practical in this setting are recommendation systems that free the user from the burden of choosing the value of each single component and let her choose by browsing generated alternatives. Many different approaches to product and service bundling recommendation have been proposed in the past, ranging from data-driven techniques [15, 127, 291] to constraint-based recommenders [287]. Data-driven approaches use a combination of rule mining of frequently purchased item-sets and similarity measures over collaborative data to generate new bundles [15, 127, 291]. These approaches, however, are heavily dependent on purchase data and cannot exploit the full combinatorial span of the services domain, nor can they express in a principled way the feasibility constraints over the generated plans that the company might want to impose. A more adequate technique from this point of view is constraint-based recommendation.

We have discussed the characteristics of constraint-based recommendation systems in Section 2.2.1. These systems can represent the full domain of feasible services implicitly through constraints over a collection of variables, and finding a new candidate amounts to solving a constraint satisfaction problem [97, 287]. Constraint-based systems can also handle *soft* constraints to represent the user preference criteria and their weights [79, 289]. Thanks to these properties, constraint-based recommenders have been extensively applied to the recommendation of configurable products like personalized service bundles [97, 99].

As discussed in Chapter 4, constructive preference elicitation can be used as a preference elicitation method over a constraint-based recommender. Applying constructive preference elicitation on top of a constraint-based system has numerous advantages. First, it provides a principled way to interact with the user and learn the weights of the soft constraints. When it comes to dealing with very large combinatorial spaces of choices, constructive methods have been proven superior to other preference elicitation techniques [86, 257]. Employing generic combinatorial optimization tools, constructive preference elicitation can not only be used to find a high utility solution, but one could also optimize a combination of different objectives, e.g. metrics correlating with long-term customer retention.

In this work we adopt a kind of coactive interaction, in which a user is allowed to reply not by changing the value of the component directly but instead the direction of improvement of some of the components. This lessens the cognitive burden for the user, letting the system take care of the underlying constraints between the components, and ensuring that no unfeasible configuration is provided as feedback to the learning algorithm. We also designed an ad-hoc coactive learning algorithm which adapts to this kind of feedback.

In the following sections we will describe the implementation of a constructive recommendation system for telco service bundling, dubbed *Smart Plan*. This system aids users in choosing the best monthly plan of telco services and products for their needs. We implemented the system as a web application that users could use in any standard web browser. We tested the system through a rigorous empirical study involving more than 130 participants. We tracked both quantitative measures of satisfaction, e.g. number of participants choosing a plan in the end, and qualitative ones through a questionnaire, e.g. whether the system was pleasant to interact with. Comparing constructive and non-constructive alternative systems, the results show a striking advantage of the constructive one in the number of users that concluded the interaction by choosing a satisfactory plan. Also, our results show that it is useless to let a user interact with a system whose interface is specifically designed for coactive interaction but that does not have the ability of synthesizing new plans from the full domain of feasible options.

Mobile Connectivity	
Gigabytes	`[0, 2, 4, 6, . . . ]`
Minutes	`[0, 500, 1000, 1500, 2000,` ∞`]`
Landline	
Internet landline	`[None, ADSL, Broadband]`
Phone landline	`[None, Pay-per-minute, Unlimited]`
Multimedia	
TV on demand	`[Netflix, Infinity TV, . . . ]`
Sky	`[Sky Tv, Sky Sport, . . . ]`
Music	`[Spotify, Apple Music, . . . ]`
Apps & Games	`[Audible, Playstation Plus, . . . ]`
Devices	
Smartphones	`[iPhone 8, Samsung Galaxy S8, . . . ]`
Tablet	`[iPad Pro 10, Galaxy Tab S2, . . . ]`
TV	`[Samsung TV 28", 4K 43", . . . ]`
Laptops	`[Macbook Air 13", . . . ]`

Table 8.1: Summary of the structure of the recommended telephone plans. Left column: component names. Right column: outline of the possible values for each component.

The rest of the chapter is organized as follows. In Section 8.1 we describe the Smart Plan system, detailing both its UI and the learning algorithm underneath. Section 8.2 reports the methodology and the findings of the user study. Finally, Section 8.3 draws the final conclusions.

8.1 The Smart Plan system

In this section we describe the Smart Plan system, a constructive recommender system for integrated telephone plans based on coactive learning. The system recommends configurations $y \in \mathcal{Y}$ formed of several components. The components are organized into four groups: [i] Mobile Connectivity services; [ii] Landline services; [iii] Multimedia (apps and services for entertainment and multimedia content provisioning such as TV streaming, music, etc.); [iv] Devices (such as smartphones, tablets, etc. paid in monthly installments). Table 8.1 summarizes the groups and their components, outlining the range of possible values for each component. In total, there are 12 basic components, each taking on average about 5 possible values, which combined form a number of possible configurations in the order of $5^{12} \approx 10^9$. The feasible space $\mathcal{Y}$ is a subset of the full set of combinations that is determined by hard constraints over the attributes. For example a landline internet connection is needed to get the pay TV service, or a mobile subscription (either voice minutes or GB of data traffic) is needed to include a smartphone in the plan. Besides the basic components, the plans also contain several derived attributes, such as the total price of the plan (computed as a function of the included components), or the amount of monthly payments due for the installments of the devices included in the plan. At the beginning of the interaction, the system asks the user to select one out of four *categories* of plans. The categories are: [i] “Young” (plans for users under 30); [ii] “Family” (plans for families with children); [iii] “Business” (high-end plans for business); [iv] “Flex” (for everybody else). The user choice of category becomes the context $x \in \mathcal{X}$ for the full elicitation process[1]. The category x may further alter the feasible space $\mathcal{Y}(x) \subseteq \mathcal{Y}$ of plans, e.g. prices of certain services are lower for the “Young” category.

The feature vector ϕ includes about 160 features describing the components of the plan and their interconnections, for instance several features describing different ranges of minutes amount (e.g. $\textit{minutes} \leq \{200, 500, 1000, \dots\}$), or a feature encoding the difference between the final price and the price paid after the installments due for the devices. A complete description of the components, constraints and features is given in the supplementary material of the original contribution [83].

The system is implemented as a web application composed of: [i] a web interface with which users interact; [ii] a web service connected to a learning back-end which is in charge of generating recommendations and collecting data.

8.1.1 The user interface

The user interface of the system is a web page alike to that shown in Figure 8.1. The page shows one recommended plan at the time, displayed as a grid containing all of its components. The grid separates the various groups of components and arranges the various components within the groups, showing placeholders for components that are not present in the given plan. For each group, the sub-total price is displayed. For the device group, the amount of monthly payments due for the devices included in the plan is shown as well. The total price of the plan is reported on the right side of the grid. A discount is sometimes applied to the total price depending on the number and type of services in the configuration. Additional information on the different components is displayed using tooltips and overlays on mouse over. The user can interact with the system via the buttons shown in the right part of the grid. At any given time the user can: [i] choose the currently displayed plan; [ii] suggest some changes to the current plan and request a new one; [iii] exit the session without choosing any plan. The system starts with an initial recommendation depending on the category chosen at the beginning and then computes new recommendations every time the user suggests changes. The plans in the history are numbered and the user can navigate the entire history of recommendations using the appropriate buttons (top right), and focus on any of them (and not only the last one) for the interaction. In any case, novel recommendations will be enqueued as the last plan in the history.

The bottom screenshot in Figure 8.1 shows the interface once the “Suggest changes” button has been pressed. In this state, several toggle buttons appear underneath the components that can be changed. In the case of numerical values and ordinal values, such as the amount of gigabytes for the mobile connection or the proposed monthly price of the plan, the user can suggest to increase or decrease the value, or ask it to be left unchanged. For categori-

[1] Henceforth, the terms “category” and “context” are used interchangeably.

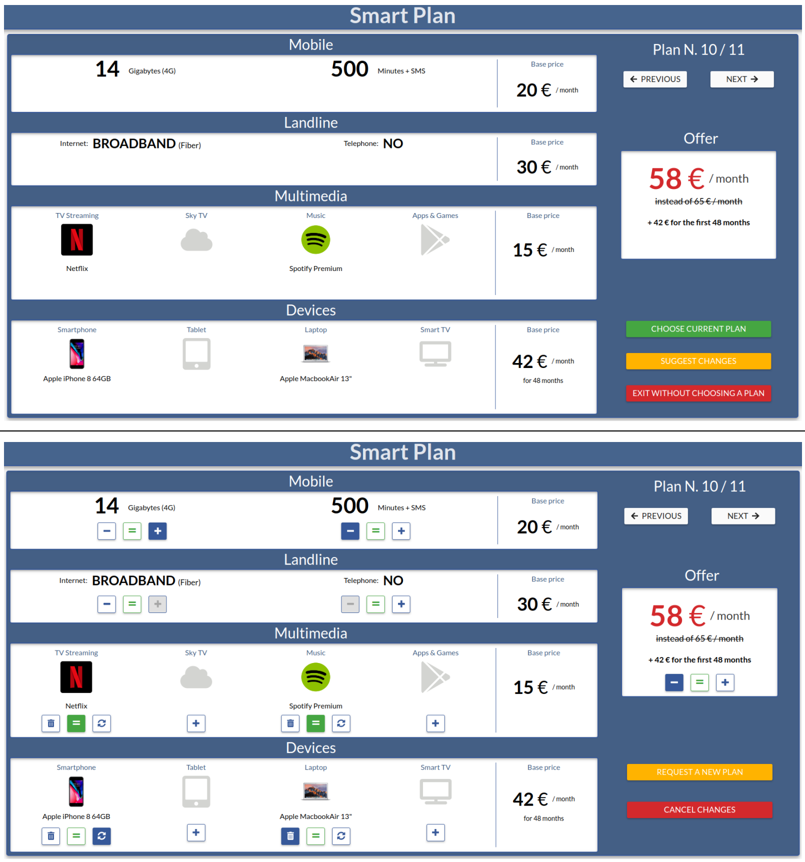

Figure 8.1: Two screenshots of the web interface of our system. On the top, a screenshot of the view of a recommended plan while navigating throughout the recommendation history. On the bottom, a screenshot showing the interface the user can interact with when selecting the changes to suggest to the system. Best viewed in color.

cal components, such as the multimedia services and devices, the user can suggest to add a service if it is not present, or, if it is present, the user can suggest to remove it, change it or keep it as it is. As the system needs to make trade-offs between the current configuration, the constraints of the domain, the current preference model and the user feedback, the user suggestion of changing one component may result in a change of other components as well. For instance, if the user asks to add more gigabytes to the plan and asks for a lower price, the system will need to remove some of the other services to accommodate this opposite feedback. This is the reason why we also included the “equal” button, with which the user can suggest which components should not change (e.g. because they are considered important) when computing these trade-offs. In general, however, we warn the user (via an overlay) to keep the number of suggestions per iteration as low as possible in order to increase the chance of having them satisfied, reminding her that there is no limit to the number of plans she can require from the system.

8.1.2 The learning subsystem

This component of the system is in charge of providing recommendations, keeping track of the user feedback and updating the user utility model accordingly. This subsystem is further divided into the actual learning algorithm, and a web service API, that intermediates between the constraint solver used to infer new recommendations, a database storing the collected data and the user interface. Here we outline the learning algorithm used in the Smart Plan system, while a more in depth description of the full system implementation is given in the supplementary material of the original publication [83].

The learning algorithm used in our system (Algorithm 13) is a slightly adapted version of the preference perceptron (see Section 3.3) to accommodate the type of feedback received by our user interface. Also, to make the learning algorithm more flexible, we used the "convex" version of the preference perceptron [247], based on the projected gradient method from [292]. In this version, the gradient steps are sized by a *learning rate* η_t and then the updated weights are projected back onto a convex set $\mathcal{B} \subseteq \mathbb{R}^d$ via a projection operator $\Pi_{\mathcal{B}}(\theta) = \operatorname{argmin}_{w \in \mathcal{B}} \|w - \theta\|$. The set $\mathcal{B}$ is typically a ℓ_2 d-dimensional ball of a given radius, or alternatively an ℓ_1 ball can be used to encourage sparsification of the weights [90].

At the beginning the algorithm receives the category $x \in \mathcal{X}$ of user choice. Based on the category x, the algorithm sets the initial weights $\boldsymbol{w}_0 = \boldsymbol{w}_x$ and the initial recommendation $y_0 = y_x$. Selecting a meaningful starting point for both the initial configuration and the initial weight vector can drastically speed-up the elicitation process. In principle, initial configurations may be estimated from generic data of previous users if available, using e.g. collaborative or content-based techniques. We did not have previous data on this task, so, with the help of a domain expert, we identified four realistic initial configurations $y_x \in \mathcal{Y}(x)$, one for each category x. The four initial configurations attempt to address the need of a prototypical user who chooses a certain category. For instance, a user choosing the category "young" would probably need more mobile data traffic than a user choosing the "family" one, probably more interested in more voice minutes and landline services (more details on the initial plans y_x can be found in the supplementary material [83]). The initial weights $\boldsymbol{w}_x$ are set to $\operatorname{argmax}_{\boldsymbol{w} \in \mathcal{B}} \langle \boldsymbol{w}, \boldsymbol{\phi}(x, y_x) \rangle$, i.e. the value for which y_x is the highest scoring configuration for the category x. The algorithm also initializes a list $\mathcal{H}$ where the plans recommended at each iteration will be stored.

At each iteration, the algorithm first presents to the user the plan $y_t = \bar{y}_{t-1}$ generated at the end of the previous iteration (which, except for the first iteration, is the one trying to meet user feedback on the previous recommendation) and stores it in the list $\mathcal{H}$. If the user is unsatisfied by the current recommendation (as well as all previous ones) and decides to quit the interaction, the algorithm stops without recommending any plan. Otherwise, the user chooses a plan from the list $\mathcal{H}$ and can either accept it (the algorithm stops and returns it as the final recommendation) or suggest some changes.

In the latter case, the algorithm updates its weight vector based on the plan selected and the current history. Indeed, the user selection of some y_k from the recommendation history as the plan on which to provide new suggestions is already an implicit source of feedback.

Algorithm 13 The learning algorithm of the Smart Plan system.

```
1: User selects category $x \in \mathcal{X}$
2: Initialize $\boldsymbol{w}_0$ and $y_0 = \bar{y}_0$ according to $x$
3: $\mathcal{H} \leftarrow$ empty list
4: for $t = 1, 2, \ldots$ do
5:     Recommend $y_t = \bar{y}_{t-1}$
6:     Append $y_t$ into $\mathcal{H}$
7:     if User quits the interaction then
8:         return $\varnothing$
9:     User selects plan $y_k \in \mathcal{H}, k \leq t$
10:    if User accepts $y_k$ then
11:        return $y_k$
12:    for $i = k-1, \ldots, t$ do
13:        $\boldsymbol{w}_t \leftarrow \Pi_{\mathcal{B}}\left(\boldsymbol{w}_{t-1} - \eta_t\left(\boldsymbol{\phi}(x, y_i) - \boldsymbol{\phi}(x, y_k)\right)\right)$
14:    Receive feedback $y_t^{=}$, $y_t^{\neq}$, $y_t^{\pm}$ on plan $y_k$
15:    $\bar{y}_t \leftarrow$ IMPROVE$(x, \boldsymbol{w}_t, y_k, y_t^{=}, y_t^{\neq}, y_t^{\pm})$
```

Intuitively, if the user chose to improve the last suggested plan (i.e. $k = t$), this implicitly tells us that the last performed improvement was most likely going in the right direction, and thus we can safely update the weights with the ranking pair (y_{t-1}, y_t) (same as in the standard preference perceptron but delayed of one iteration). If, instead, $k < t$, it means that the user preferred to "start over" from some previous recommendation y_k, probably because the recommendations y_i, for $k < i \leq t$, turned out to be farther out from the user desiderata than y_k. This implicit feedback provides the ranking pairs (y_i, y_k) for $k < i \leq t$, exploited to update the weights (line 13 of Algorithm 13). Preliminary experiments showed that this is a better choice than just adding the single pair (y_t, y_k).

Finally, the user provides feedback on the chosen plan y_k, and a refined plan accounting for such a feedback is generated. The user feedback is not a direct manipulation of the recommended plan but rather a set of "suggestions" on the components that need to be changed or should be kept equal, leaving the system the possibility of adjusting the remaining components to accommodate these suggestions. The improvement $\bar{y}_t$ is then obtained by finding a feasible plan satisfying the suggestions of the user as much as possible.

In order to formalize the search problem for an improvement $\bar{y}_t$, plans are encoded in a form that allows to test the satisfaction of the different types of suggestions the user can provide. A plan y_k is thus represented in terms of two component vectors, $\boldsymbol{\varphi}_n(y_k)$ and a $\boldsymbol{\varphi}_c(y_k)$ [2]. The first contains quantitative information for numerical or ordinal attributes (e.g. number of minutes or gigabytes) and presence/absence information for categorical ones (e.g. TV Streaming, Laptop). The second contains the one-hot encoding of the categorical attributes (e.g. which TV Streaming among the different options available) [3].

[2]Note that these vectors differ from the feature vector $\boldsymbol{\phi}$ on which the utility function is defined, although they largely overlap in practice. See the supplementary material for details [83].

[3]The vectors $\boldsymbol{\varphi}_n(y_k)$ and $\boldsymbol{\varphi}_c(y_k)$ also contain some more elaborate features, such as the price range and the smartphone category. More details are provided in the supplementary material of the original paper [83].

The user suggestions are gathered into three feedback vectors $\boldsymbol{y}_t^{\pm}, \boldsymbol{y}_t^{=}, \boldsymbol{y}_t^{\neq}$. The first feedback vector $\boldsymbol{y}_t^{\pm}$ has the same size as $\boldsymbol{\varphi}_n(y_k)$, and contains 1 and -1 for components the user has requested to respectively increase (by pressing the "plus" button) and decrease (by pressing the "minus" button) in $\bar{y}_t$ with respect to y_k, and 0 otherwise.

The second vector $\boldsymbol{y}_t^{=}$ has again the same size as $\boldsymbol{\varphi}_n(y_k)$, and contains for each component the value 1 if the user requested to keep the value unchanged in $\bar{y}_t$ with respect to y_k (by pressing the "equal" button), and 0 otherwise.

The third vector $\boldsymbol{y}_t^{\neq}$ has instead the same size as $\boldsymbol{\varphi}_c(y_k)$, and contains 1 on components for which the user has requested to be changed in value (by pressing the "change" button) in $\bar{y}_t$ with respect to their value assumed in y_k. In other words, if the $\boldsymbol{y}_t^{\neq}$ contains 1 for some component, then the new plan $\bar{y}_t$ should contain a different value (not specified which) for that component with respect to y_k.

The improvement $\bar{y}_t$ is obtained by solving an optimization problem that attempts to maximize a "feedback" score $G_t(\bar{y}_t)$, i.e. a function that encodes how well the user suggestion are satisfied. The new plan however should also be of high utility according to the current utility model u_t and should not be too distant from y_k, in order to make the transition from one plan to the next one as smooth as possible. The IMPROVE function (line 15 of Algorithm 13) seeks a plan $\bar{y}_t$ trading-off these aspects, by solving the following optimization problem:

$$\begin{aligned}
\bar{y}_t &= \operatorname*{argmax}_{y \in \mathcal{Y}(x)} \; G_t(y) + \lambda_1 \; u_t(x, y) - \lambda_2 \; \|\boldsymbol{\phi}(x, y) - \boldsymbol{\phi}(x, y_k)\|_1 \qquad (8.1) \\
& G_t(y) = \lambda_3 \langle \boldsymbol{y}_t^{\pm}, \boldsymbol{\delta}_t^n \rangle - \lambda_4 \langle \boldsymbol{y}_t^{=}, |\boldsymbol{\delta}_t^n| \rangle + \lambda_5 \|\boldsymbol{y}_t^{\neq} \odot \boldsymbol{\delta}_t^c\|_0 \\
& \boldsymbol{\delta}_t^n = \boldsymbol{\varphi}_n(y) - \boldsymbol{\varphi}_n(y_k) \qquad \boldsymbol{\delta}_t^c = \boldsymbol{\varphi}_c(y) - \boldsymbol{\varphi}_c(y_k)
\end{aligned}$$

The above optimization problem maximizes a combination of: [i] the utility of y according to the current model $u_t(x, y)$; [ii] the (negated) ℓ_1 distance between y and the selected plan y_k (in feature space); [iii] the feedback score function $G_t(y)$ computing how much y meets the user suggestions. The function $G_t(y)$ is itself a combination of: [i] the amount $\langle \boldsymbol{y}_t^{\pm}, \boldsymbol{\delta}_t^n \rangle$ of changes that go in the same direction with respect to the user suggestion (increase if user asked to increase and vice versa); [ii] the number $\langle \boldsymbol{y}_t^{=}, |\boldsymbol{\delta}_t^n| \rangle$ of components that should remain equal but they have not; [iii] the number $\|\boldsymbol{y}_t^{\neq} \odot \boldsymbol{\delta}_t^c\|_0$ of categorical components that changed according to the user request. In the above formula: $|y|$ denotes the element-wise absolute value; $\odot$ denotes the Hadamard product (element-wise product); and $\|\cdot\|_0$ denotes the ℓ_0 norm (number of non-zero elements). The parameters $\lambda_1, \ldots, \lambda_5$ are real coefficients defining the relative importance of each component of the function being maximized. These coefficients, as well as the learning rate η_t, were chosen manually among a set of predefined values, validating the quality of the recommendations on a pilot set of users (beta testers which are not included among the real users involved in the study described in Section 8.2). The final implementation used a learning rate $\eta_t = \frac{1}{t^{0.8}}$ with a projection onto a ℓ_2 ball. The learning rate decreases rather quickly because, in this particular setting, feedback given in the early iterations is more important than finer adjustments at later iterations when most of the users already had a rough idea in mind of what the final plan should look like. The parameters $\lambda_1, \ldots, \lambda_5$ were set so that the feedback score, primarily the $\boldsymbol{y}_t^{\pm}$ component, had the highest weight. Furthermore, a slightly higher weight was given to the distance $\|\boldsymbol{\phi}(x, \bar{y}_t) - \boldsymbol{\phi}(x, y_k)\|_1$ with respect to the utility $u_t(x, \bar{y}_t)$, so that not too many components would change between y_k and $\bar{y}_t$, thus ensuring a smooth enough transition between plans.

Regarding the implementation, we used the MiniZinc constraint programming language as modeling platform [188]. The underlying constraint solver used was Gurobi[4]. Concerning timing, preliminary experiments showed that inference and learning steps took on average 0.21 seconds per iteration using Gurobi on a 2.8 GHz Intel Xeon CPU with 8 cores and 32 GiB of RAM. This time magnitude makes the user interaction in the practical implementation comfortable and without delays.

8.2 Empirical validation

This section describes the empirical validation of the system through an experiment with real participants. The supplementary material of the original paper [83] also contains a batch of preliminary experiments made with simulated users in order to assess the convergence rate of the algorithm when interacting with users of different informativeness, as done in previous works on constructive preference elicitation [82, 84, 255].

In order to evaluate the usefulness of our constructive approach, we compared it with two alternative versions of the system. The first mimics the standard approach of telecommunication companies, which consists of hand-crafting a pool of integrated plans tailored for different categories of users, and let them choose their preferred plan in the pool. The second employs the very same interaction modality of our constructive approach, but when building the recommendation in response to the suggestions from the user, it is forced to pick one of the preset plans in the pool. The rationale behind the inclusion of this third version of the system is to evaluate the relative importance of the constructive interface *versus* the constructive plan generation in determining the success of the interaction and the user's satisfaction. In what follows, we refer to our constructive approach as CC (standing for "constructive-constructive"), while the two alternative approaches as PP ("pool-pool") and PC ("pool-constructive"), respectively.

In the CC version of the system, the recommendations were selected from the full configuration space as described in Sections 8.1. In the PP version, the recommendations were selected from a predefined pool. The pool was created by reviewing currently available telco companies offers and interacting with a domain expert. This process resulted in 65 different plans, divided into a set of sub-pools $\mathcal{P}_x \subset \mathcal{Y}$ of approximately equal size for each of the "Young", "Family", "Business", and "Flex" categories x (see Section 8.1). Note that, albeit very small if compared to the size of the configuration space, the pool is still substantially larger than those typically provided by telco companies. Finally, in the PC version, the algorithm learned a model of the user preferences as for the fully constructive version, but recommendations

[4]Gurobi: `gurobi.com`

were selected from the same predefined pool of the PP version. However, while in PC version the full pool was used for inference, in the PP version only the sub-pool associated to the chosen category was used, to make the choice not too overwhelming for the user.

Participants and procedure

A sample of 157 adults was recruited for the experiment. Participation was anonymous and on a voluntary basis. Those who abandoned the session or did not visualize at least two recommendations (15% of the total) were excluded, leaving us with 134 participants. Fifty-eight percent of them were male, the average age was 32.7 years (SD 10.42). The experiment was performed online, throughout a web page compatible with any widely used web browser. Participants were explicitly informed that the experiment was only for scientific purposes and that the plans were not associated to any real offer. However, they were asked to behave as if they really had to choose a plan to purchase.

A between-subjects design was employed: participants were randomly assigned to one of three groups, each interacting with a different version (CC, PP or PC) of the recommender. More specifically, 34% were presented with the PP, 33% with the PC, and 33% with the CC version of the system. Participants in all groups had first to select the category of plans they were most interested in. The system used the information about the category to suggest an appropriate initial recommendation y_0 (see Section 8.1.2). In the CC and PC versions of the system, this information was also used to initialize the weight vector $\boldsymbol{w}_0$, whereas in the PP version of the system the category was used to select the pool $\mathcal{P}_x$ to be presented (in a randomized order) to participants.

Participants could spend as much time as they needed to assess each recommended plan. They could then suggest modifications to the plan (for PC and CC versions only, as outlined in Section 8.1.1), choose the plan, or decide to conclude the experimental session without choosing any plan (or even to leave the test simply by closing the browser window). As described in Section 8.1.1, in all the versions of the system, participants were free to navigate the full history of recommendations by going back and forth between plans. Specific instructions on how to interact with the specific version of the system were provided by means of a video tutorial before the start of the experiment.

Finally, once participants had chosen a plan or decided to leave the experimental session without choosing any, they were presented with a small questionnaire. For all participants, this included questions about how pleasant and tiring the interaction with the system had been on a scale ranging from 0 (= “completely disagree”) to 5 (= “completely agree”). Other questions depended on the version of system and on whether the user chose a plan or left without choosing any. All participants in the PC and CC versions were also asked whether they had found the plans proposed by the system of growing interest, while all participants who chose a plan were asked how much they were satisfied with it (again from 0 to 5). At the very end, participants were invited to provide minor demographic information (i.e., their age and gender).

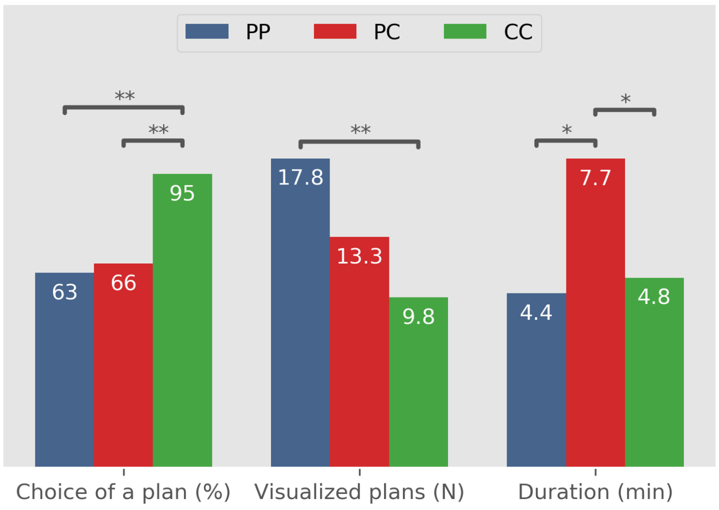

Figure 8.2: Statistics concerning the interaction with the three (PP, PC, CC) versions of the system. From left to right: [i] the percentage of participants who found a plan satisfying enough to choose it; [ii] the average number of plans visualized (only for participants who chose a plan); [iii] the average duration (in minutes) of the interaction with the system (only for participants who chose a plan). Top bars indicate significant comparisons (*= p<.05 and **= p<.01, adjusted standardized residuals post-hoc analysis for [i], Tukey post-hoc tests for [ii] and [iii]). Best viewed in color.

Results

Participants who interacted with the three (PP, PC, and CC) versions of the system were not significantly different by gender (χ^2(2)=4.334, p=.114), age (one-way ANOVA, F(2,120)=.617, p=.541), and possession of a phone plan (χ^2(2)=.628, p=.731). They did not significantly differ also in their choice of the category of the plan (χ^2(6)=3.763, p=.709). As a consequence, they can be considered fully suitable to be compared in the experiment. Overall, the categories “flex”, “young”, ‘family”, and “business” were chosen by 31%, 45%, 18%, and 6% of the participants, respectively.

As shown in Figure 8.2, the percentage of participants who ended their interaction with the system successfully by choosing a plan was significantly different in the three groups (χ^2(2)=15.106, p<.001). In particular, while almost all participants (95%) who interacted with the CC version of the system found a plan that they liked, the same held only for a bit more than half of those who interacted with the other two versions (63% and 66% for PP and PC, respectively). The difference between the former and the latter is statistically significant (adjusted standardized residuals post-hoc analysis, p<.01).

Before making their choice, participants in the three groups visualized a different number of plans (one-way ANOVA, F(2,97)=6.474, p<.01). In particular, participants who interacted with the PP version of the system visualized a greater number of plans than participants who

	Pleasant interaction	**Tiring interaction**	**Increasing interest**	**Satisfaction degree**
PP	3.79	1.07	–	3.54
PC	2.95	2.40	1.51	3.03
CC	3.44	1.79	3.02	3.34

Table 8.2: Participants' average answers to the final questions in the three (PP, PC, CC) system versions. Value range from 0 to 5. From left to right: [i] how pleasant the interaction with the system was; [ii] how tiring the interaction with the system was; [iii] (only for participants in the PCand CCversions) whether the plans proposed by the system were of growing interest; [iv] (only for participants who chose a plan) how much the chosen plan was considered satisfying.

interacted with the CC version (Tukey post-hoc test, p<.01). This is not entirely surprising since participants who interacted with the PP version of the system could not suggest themselves a new plan, but had only the possibility to explore the available plans.

Participants in the three groups who chose a plan also differed with regard to the duration of their overall interaction with the system (one-way ANOVA, F(2,97)=4.622, p<.05). This was greater for participants who interacted with the PC version of the system than participants who interacted with the PP and CC versions (Tukey post-hoc tests, p<.05), while participants in the latter two groups did not differ between each other (Tukey post-hoc test, p=.935). Such a result indicates that the PC version of the system was especially time-consuming, while the CC version was comparable to the PP one.

The average answers to the final questions are reported in Table 8.2. Unsurprisingly, the three versions of the system have not been rated as equally enjoyable or tiring (one-way ANOVAs, F(2,126)=5.928, p<.01, and F(2,126)=8.703, p<.01, respectively). More specifically, Tukey post-hoc tests revealed that interacting with the PC version of the system was considered as less pleasant (<.01) and more tiring (p<.01) than interacting with the PP version, while there were no significant differences between PC and CC versions (p=.33 and p=.06, respectively). This result indicates that combining a constructive interface with a standard search over a fixed pool of candidates is not a good strategy, since a restricted number of options cannot typically accommodate the modifications suggested by the users. Moreover, it shows that the interaction with the CC version of the system, although more engaging, is not perceived as less pleasant or more tiring than the interaction with the PP one.

Participants who interacted with the CC version of the system found the plans progressively suggested to them of growing interest more than participants who interacted with the PC version (Independent t-test, t(84)=5.972, p<.01). Yet again, this suggests that participants' appreciation of the interaction depends on having a constructive plan generator rather than a constructive interface.

Finally, there were not significant differences in the satisfaction with the chosen plan between the three groups (F(2,95)=1.911, p=.154).

8.3 Summary

In this chapter, we presented Smart Plan, a constructive recommender system for bundles of telecommunication services, entertainment subscriptions and electronic devices. The typical marketing strategy for this type of bundles used by telco companies consists in selecting a fixed set of carefully designed plans, yet with limited possibility of editing. In contrast, we allow the user to fully customize her own plan, within the allowed boundaries. This configuration process is eased by Smart Plan, which uses constructive preference elicitation and coactive feedback to learn the preferences of the user and suggests progressively more interesting plans.

A between-subjects study was carried out with real participants in order to validate and compare our system against two alternative non-constructive systems. The results showed that, when using Smart Plan, the participants almost always found a satisfactory plan at the end of the interaction, which was not the case for the non-constructive alternatives, regardless of the interfaced used. The results also show that using an interface designed for constructive interaction while using a non-constructive algorithm underneath leads to an unpleasant and tiring interaction. Satisfaction degree was not significantly different across the systems. This might imply that recommendation quality was, after all, similar across the various systems, either for the inability of the system to provide better recommendation, or to the general high quality of the plans in the handcrafted pool. Another explanation might be the stronger correlation of the satisfaction degree with the plan itself rather than the process for acquiring it. To assess this correlation, we will need to perform a more in depth analysis with a higher number of participants.

CHAPTER 9

CONCLUSION

In this chapter we will provide a summary of the whole thesis, highlighting the salient points and remarking how the methods proposed in the thesis addressed the research problems posed in the introduction. We will then conclude with some research directions that might be worth pursuing in the future.

9.1 Summary of contributions

In Chapter 1, we discussed how the adoption of mass customizable products has come to a halt due to the complexity of the decision problems associated with their configuration. To solve this problem and to tackle related preference-based combinatorial problems, we laid down twelve principles that a preference elicitation system should abide to. Of these, the first five were posited by Guo and Sanner [128] for any preference elicitation method. To be able to tackle complex combinatorial decision problems, a.k.a. constructive problems, we postulate other seven properties that a constructive preference elicitation system should exhibit. The following is a recapping list of all twelve of these properties:

1. *Real-time interaction*
2. *Multi-attribute domains*
3. *Low cognitive load*
4. *Robustness to noise*
5. *Scalability (feedback)*
6. *Constrained combinatorial spaces*
7. *Hybrid domains*
8. *Explicit and implicit feedback*
9. *Contextual information*
10. *Optimality guarantees*
11. *Scalability (inference time)*
12. *Expressive trade-offs*

As we argued in Chapter 1, and saw in detail in Chapter 2, existing state-of-the-art methods in preference elicitation or constraint-based recommendation do satisfy some of the above requirements, yet none of these techniques can cope with all of them simultaneously.

Preference elicitation for combinatorial problems

After stating our goal, we proposed a method for constructive preference elicitation and we claimed that it would be able to satisfy all of our desired properties. Based on the background knowledge built up in Chapter 3, we described our constructive preference elicitation framework in Chapter 4. Our technique is based on online structured prediction, a machine learning method for predicting structured objects, while learning a utility model in an online fashion. Online structured prediction, just like standard batch structured-output prediction, can be combined with constraints solvers, which can be used to make predictions over constrained combinatorial domains. Using a constraint solver as inference oracle, we can easily shape the domain as a hybrid multitude of categorical and numerical attributes subject to arbitrary constraints, hence satisfying properties 2, 6 and 7.

The particular online structured prediction framework that we employ in this thesis is coactive learning [247]. As we described it in Section 3.3, coactive learning satisfies properties 3, 4, 5, 8, 9, and 10 out-of-the-box. As we have proven in Chapter 7 and 8, coactive feedback is very flexible and can be easily acquired both explicitly and implicitly, thereby satisfying properties 3 and 8. Coactive learning also handles context directly in the utility model, satisfying property 9. Coactive learning comes with guarantees of vanishing average regret, ensuring its convergence to an optimal solution, as required by property 10. This method also learns from and provide guarantees for weak, noisy user feedback, ensuring the satisfaction of property 4. In our tests and validation in Chapter 7 and 8, we have demonstrated that this technique, when applied to constructive preference elicitation, is able to provide real time interaction and satisfying solutions with little feedback, hence meeting the requirements of properties 1 and 5.

In Chapter 1, and again in Chapter 4, we argued that coactive learning does not satisfy properties 11 and 12 on its own. In particular, the scalability to large domains is mainly limited by the capacity of the underlying constraint solver to make predictions fast enough to ensure real-time interaction. Also, the expressiveness of the utility function is capped by the amount of feature engineering required to define meaningful preference criteria for a large and diverse audience. In order to solve these research problems, we proposed two approaches that extend coactive learning so as to satisfy the requirements of properties 11 and 12.

Problem decomposition strategies

In Chapter 5, we introduced a partitioning strategy for constructive decision problems and proposed a coactive learning algorithm for eliciting the user utility relying solely on partwise interaction. This has the advantage that, at each iteration, the algorithm only needs to change a part of the object, i.e. a subset of the decision variables, rather than the full object at once. This may speedup the inference procedure exponentially, depending on the size of the parts with respect to the whole object. Feedback is only partial too, implying a lower cognitive effort per iteration for the user. These two aspects enable us to tackle much larger constructive problems by simply splitting them into smaller ones and then pulling the results together. This, however, comes at the price of optimality. As we have seen, for complex problems involving overlapping features between parts (which is equivalent to the case of GAI utilities), this method is not able to reach a global optimum with respect to the hidden user utility. However, we were able to show that our method converges to a "local optimum", for which the user cannot improve any part of the proposed object separately from the others. Also, we showed that this local optimum has bounded approximation error with respect to the global one. Crucially, the approximation error depends primarily on the amount of overlapping features within the parts, which essentially means that we are able to scale to very large problems by decomposing them and converge to good solutions as long as the features do not substantially overlap. We also evaluated the algorithm empirically and showed that in practice it often reaches or comes very close to a global optimum in a reasonable amount of iterations, while being vastly more computationally efficient than its counterpart on full objects. Using this technique we can scale to larger domains, and hence satisfy the above property 11.

Feature elicitation for coactive learning

In Chapter 6, we detailed our coactive critiquing approach, which combines coactive learning and example critiquing with the aim of eliciting relevant features and allowing the utility model to adapt to the user personal preference criteria. Our approach consists in eliciting a critique any time the algorithm is faced with a dubious improvement, i.e. when the set of features cannot correctly represent the improvements collected up to that point. This makes the algorithm ask for critiques only when strictly necessary. Also, the critiques asked by our technique are simpler than most other critiquing systems as they only require the user to state an "explanation" for why the object that she provided as feedback is indeed an improvement over the recommended one. We showed that our algorithm enjoys an average regret bound, converging to an optimal solution as soon as the feature space is expressive enough. This fact is reflected in the experiments, in which we see that a small percentage of the total features is needed for our algorithm to converge, compared to standard coactive learning. This approach allows the algorithm to express complex feature spaces and to adapt to each user's specific preferences, solving the issue stated in property 12.

Applications and extensive evaluation

Besides the algorithmic improvements, this thesis also contributes with two implemented application (besides the test beds used in Chapter 5 and 6), one of which was tested in a user study involving real participants. Chapter 7 showcases the application of coactive learning for constructive preference elicitation in the context of layout synthesis, a type of design task with the goal of arranging objects in a 2D or 3D space. Chapter 8, instead, describes our application of coactive learning to product and service bundling, a type of product configuration task. In particular, we implemented a constructive recommender for service bundles in the telecommunication industry, including connectivity and multimedia services, as well as leased devices. In our empirical validation, we have seen that, with our constructive recommendation system, the percentage of satisfied users increases dramatically with respect to alternative pool-based techniques. Also, using our system in combination with a specifically designed interface is not much more challenging than simply browsing a pool of alternatives, taking users about the same time as using the pool-based interface to arrive at a satisfying solution. These results highlight the practical benefit of using our constructive preference elicitation technique, aside from the methodological ones.

9.2 Future directions

In this section we describe several possible research direction not directly engaged in this thesis but that would constitute interesting developments for future work.

Multiple users and social choice

The simplest yet most generic way of extending our constructive preference elicitation framework is to scale it from single to multiple users [137]. Indeed, recommendation systems, including constructive ones, are meant to be used by many users and exploiting information about their collectively learned preferences is crucial for making the system effective. We already mentioned in Section 4.2.1 that, thanks to the contextual information directly handled by the utility model of coactive learning, we can learn offline a global utility model based on all the previously collected data, generalizing over user features encoded as contexts, and then start with this model as a new user comes along, in order to accelerate the elicitation process. This approach can be seen as form of *multitask* learning [11, 231], which has already been explored in the constructive case [258]. Specialized algorithms for learning global and local utility functions via coactive learning in a multitask setting have already been proposed [117], which are directly applicable to our framework, yet a practical implementation confirming the feasibility of this approach in a constructive scenario is still missing.

Another setting of interest for constructive recommenders considering multiple users is that of *social choice* [17, 174], in which a group of user has to collectively decide upon *one* instance maximizing the *social utility.* This has particular appeal in a constructive preference elicitation setting, which could be used to devise preference-based social extensions to classical combinatorial problems such as the travelling salesperson problem [195] and the nurse rostering problem [42]. These could be the basic formulations for e.g. a social trip planner and a social scheduler, based on social constructive preference elicitation. Coactive learning has been already extended to this context as well [214], yet it has mainly been applied to optimizing search engines responses to ambiguous terms involving conflicting results. Nevertheless, this technique may be straightforwardly applied to the constructive case as well, providing convergence guarantees out-of-the-box. Further research on the practical applicability of this technique to the constructive case is however needed.

More expressive utility functions

While the utility function used in coactive learning and constructive preference elicitation has the potential for representing any utility over the object attributes, we are still limited by two important factors, namely feature engineering and linearity. The former is partially addressed by coactive critiquing (see Chapter 6), yet we can not expect the user to state very complex critiques that may however be relevant or may speedup the elicitation process. The latter is mainly imposed by the inability of constraint solvers to handle non-linear objectives. One possible approach to overcome this limitation would be to employ solvers able to handle quadratic objectives and constraints, such as quadratically constrained quadratic programming or second-order cone programming solvers. Quadratic programs would already be much more expressive than linear, allowing to directly encode features and constraints over e.g. areas and Euclidean distances. However, quadratic optimization is much harder than linear, so its applicability in constructive preference elicitation still needs to be understood.

Non-linear aggregation of preference criteria has long been of interest in preference elicitation to make utility functions more expressive and capture subtle nuances in the preference criteria of the users. One recent approach uses the *Choquet integral*, a sophisticated non-linear aggregator, with an efficient minimax regret approach [18]. It would be interesting to investigate whether it is possible to leverage this approach in a constructive scenario.

Another approach would be to use a deep neural network [159] to represent a non-linear utility function. Recent advances in deep learning have produced novel architectures for *deep structured prediction* [16, 57]. These methods learn an *energy network* and make predictions by optimizing this network through gradient descent. The energy network is comparable to an (inverse) utility functions, which makes these methods appealing for preference learning. However, mixing deep neural networks and preference elicitation over constrained domains is still a challenge. One difficulty is the amount of data needed to train one such architecture, which is rarely available in preference elicitation. Another problem is the fact that deep neural networks still do not cope well with prior knowledge in the form of hard constraints.

One possibility to handle the above issues is to use a combination of deep energy networks and constraint solving in a multitask scenario. The energy network would provide a global utility, trained with data from all users, and would provide a rough, unconstrained prediction. Then, the constraint solver would be tasked to “refine” the network prediction by taking into account the hard constraints of the problem and the local utility of the specific user. We expect that this method would be a way of learning expressive utility models and would provide good recommendations for new users. We also expect that the inference process would be faster, since the constraint solver would only need to refine an existing prediction and not create one from scratch.

An alternative prospect relies on one particular types of networks, called *input convex neural networks* (ICNNs), which provide deep structured prediction with *convex* inference [9]. A way of injecting prior knowledge into an ICNN would be to represent constraints in a “soft logic” formalism. In particular, using a fragment of Łukasiewicz logic, the resulting objective is still convex [75]. In this setting, efficient optimization algorithms for non-smooth convex optimization can be used for inference, such as the *alternating direction method of multipliers* (ADMM) [32]. The same formalism and optimization method are used in *hinge-loss Markov random fields* (HL-MRFs), a class of statistical relational models for constrained structured prediction at the basis of the *probabilistic soft logic* (PSL) language [13]. ICNNs could function as non-linear potentials for HL-MRFs while keeping inference convex in the output variables. The application of PSL, with or without non-linear potentials through ICNNs, to constructive preference elicitation is an interesting research direction.

Other statistical relational learning frameworks may be integrated in constructive preference elicitation as well. *Type extension trees* [108], for instance, are tools for learning complex combinatorial features which may be used in substitution of or in combination with coactive critiquing to learn complex, personalized preference criteria. Their potential use in constructive preference elicitation is currently being investigated.

Bibliography

[1] Gediminas Adomavicius and Alexander Tuzhilin. “Context-aware Recommender Systems”. In: *Recommender Systems Handbook*. Springer, 2011, pp. 217–253.

[2] Charu C Aggarwal. *Recommender Systems: The Textbook*. Springer, 2016.

[3] Ryuya Akase and Yoshihiro Okada. “Automatic 3D Furniture Layout Based on Interactive Evolutionary Computation”. In: *Seventh International Conference on Complex, Intelligent, and Software Intensive Systems*. IEEE. 2013, pp. 726–731.

[4] Ryuya Akase and Yoshihiro Okada. “Web-based Multiuser 3D Room Layout System Using Interactive Evolutionary Computation with Conjoint Analysis”. In: *Proceedings of the 7th International Symposium on Visual Information Communication and Interaction*. ACM. 2014, p. 178.

[5] Michel Aldanondo and Elise Vareilles. “Configuration for mass customization: how to extend product configuration towards requirements and process configuration”. In: *Journal of Intelligent Manufacturing* 19.5 (2008), pp. 521–535.

[6] Christopher Alexander. *A Pattern Language: Towns, Buildings, Construction*. 1977.

[7] Xavier Amatriain, Alejandro Jaimes, Nuria Oliver, and Josep M Pujol. “Data Mining Methods for Recommender Systems”. In: *Recommender Systems Handbook*. Springer, 2011, pp. 39–71.

[8] Xavier Amatriain, Josep M Pujol, and Nuria Oliver. “I like it... I like it not: Evaluating User Ratings Noise in Recommender Systems”. In: *International Conference on User Modeling, Adaptation, and Personalization*. Springer. 2009, pp. 247–258.

[9] Brandon Amos, Lei Xu, and J Zico Kolter. “Input convex neural networks”. In: *Proceedings of the 34th International Conference on Machine Learning (ICML)*. 2017, pp. 146–155.

[10] Dana Angluin. “Queries and Concept Learning”. In: *Machine Learning* 2.4 (1988), pp. 319–342.

[11] Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. “Multi-Task Feature Learning”. In: *Advances in Neural Information Processing Systems*. 2007, pp. 41–48.

[12] Fahiem Bacchus and Adam Grove. “Graphical models for preference and utility”. In: *Proceedings of the Eleventh conference on Uncertainty in Artificial Intelligence (UAI)*. Morgan Kaufmann Publishers Inc. 1995, pp. 3–10.

[13] Stephen H Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. “Hinge-Loss Markov Random Fields and Probabilistic Soft Logic”. In: *arXiv preprint arXiv:1505.04406* (2015).

[14] Gükhan H. Bakir, Thomas Hofmann, Bernhard Schölkopf, Alexander J. Smola, Ben Taskar, and S. V. N. Vishwanathan. *Predicting Structured Data.* 2007. ISBN: 0262026171.

[15] Moran Beladev, Lior Rokach, and Bracha Shapira. “Recommender systems for product bundling”. In: *Knowledge-Based Systems* 111 (2016), pp. 193–206.

[16] David Belanger and Andrew McCallum. “Structured Prediction Energy Networks”. In: *Proceedings of the 33rd International Conference on Machine Learning (ICML).* 2016, pp. 983–992.

[17] Nawal Benabbou, Serena Di Sabatino Di Diodoro, Patrice Perny, and Paolo Viappiani. “Incremental Preference Elicitation in Multi-Attribute Domains for Choice and Ranking with the Borda Count”. In: *International Conference on Scalable Uncertainty Management.* Springer. 2016, pp. 81–95.

[18] Nawal Benabbou, Patrice Perny, and Paolo Viappiani. “Incremental Elicitation of Choquet Capacities for Multicriteria Decision Making”. In: *Proceedings of the Twenty-first European Conference on Artificial Intelligence (ECAI).* IOS Press. 2014, pp. 87–92.

[19] Hande Y Benson and Ümit Sağlam. “Mixed-Integer Second-Order Cone Programming: A Survey”. In: *Theory Driven by Influential Applications.* 2013, pp. 13–36.

[20] James R Bettman, Mary Frances Luce, and John W Payne. “Constructive Consumer Choice Processes”. In: *Journal of Consumer Research* 25.3 (1998), pp. 187–217.

[21] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh. *Handbook of Satisfiability.* Vol. 185. IOS press, 2009.

[22] Alessandro Biso, Francesca Rossi, and Alessandro Sperduti. “Experimental Results on Learning Soft Constraints”. In: vol. 2. 2000, pp. 435–444.

[23] Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. “Semiring-Based Constraint Satisfaction and Optimization”. In: *Journal of the ACM (JACM)* 44.2 (1997), pp. 201–236.

[24] Stefano Bistarelli and Barry O’Sullivan. “A Theoretical Framework for Tradeoff Generation using Soft Constraints”. In: *International Conference on Innovative Techniques and Applications of Artificial Intelligence.* Springer, 2004, pp. 69–82.

[25] Dirk Bollen, Bart P Knijnenburg, Martijn C Willemsen, and Mark Graus. “Understanding Choice Overload in Recommender Systems”. In: *Proceedings of the 4th ACM Conference on Recommender Systems (RecSys).* ACM. 2010, pp. 63–70.

[26] Craig Boutilier. “A POMDP Formulation of Preference Elicitation Problems”. In: *AAAI/IAAI.* American Association for Artificial Intelligence. 2002, pp. 239–246.

[27] Craig Boutilier, Fahiem Bacchus, and Ronen I Brafman. “UCP-Networks: A Directed Graphical Representation of Conditional Utilities”. In: *Proceedings of the Seventeenth conference on Uncertainty in Artificial Intelligence (UAI).* Morgan Kaufmann Publishers Inc. 2001, pp. 56–64.

[28] Craig Boutilier, Ronen I Brafman, Holger H Hoos, and David Poole. “Reasoning With Conditional Ceteris Paribus Preference Statements”. In: *Proceedings of the Fifteenth conference on Uncertainty in Artificial Intelligence (UAI).* 1999, pp. 71–80.

[29] Craig Boutilier, Relu Patrascu, Pascal Poupart, and Dale Schuurmans. “Constraint-based optimization and utility elicitation using the minimax decision criterion”. In: *Artificial Intelligence* 170.8-9 (2006), pp. 686–713.

[30] Craig Boutilier, Kevin Regan, and Paolo Viappiani. “Preference Elicitation with Subjective Features”. In: *Proceedings of the third ACM Conference on Recommender Systems (RecSys).* ACM. 2009, pp. 341–344.

[31] Craig Boutilier, Kevin Regan, and Paolo Viappiani. “Simultaneous Elicitation of Preference Features and Utility”. In: *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence.* 2010.

[32] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. “Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers”. In: *Foundations and Trends® in Machine learning* 3.1 (2011), pp. 1–122.

[33] Ralph Allan Bradley and Milton E. Terry. “Rank Analysis of Incomplete Block Designs: I. The Method of Paired Comparisons”. In: *Biometrika* 39.3/4 (1952), pp. 324–345.

[34] R Brafman, C Domshlak, and T Kogan. “On Generalized Additive Value-Function Decomposition”. In: *Proceedings of the conference on Uncertainty in Artificial Intelligence (UAI)* (2004).

[35] Ronen Brafman and Carmel Domshlak. “Preference Handling – An Introductory Tutorial”. In: *AI magazine* 30.1 (2009), p. 58.

[36] Jacinto Braga and Chris Starmer. “Preference Anomalies, Preference Elicitation and the Discovered Preference Hypothesis”. In: *Environmental and Resource Economics* 32.1 (2005), pp. 55–89.

[37] Darius Braziunas. *Computational Approaches to Preference Elicitation.* Tech. rep. Department of Computer Science, University of Toronto, 2006.

[38] Darius Braziunas and Craig Boutilier. “Elicitation of Factored Utilities”. In: *AI Magazine* 29.4 (2008), p. 79.

[39] Darius Braziunas and Craig Boutilier. “Local Utility Elicitation in GAI Models”. In: *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence (UAI).* 2005, pp. 42–49.

[40] Darius Braziunas and Craig Boutilier. “Minimax regret based elicitation of generalized additive utilities”. In: *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence (UAI).* 2007, pp. 25–32.

[41] Sébastien Bubeck and Nicolò Cesa-Bianchi. “Regret Analysis of Stochastic and Nonstochastic Multi-armed Bandit Problems”. In: *Foundations and Trends® in Machine Learning* 5.1 (2012), pp. 1–122.

[42] Edmund K Burke, Patrick De Causmaecker, Greet Vanden Berghe, and Hendrik Van Landeghem. “The state of the art of nurse rostering”. In: *Journal of Scheduling* 7.6 (2004), pp. 441–499.

[43] Robin Burke. “Hybrid Recommender Systems: Survey and Experiments”. In: *User Modeling and User-adapted Interaction* 12.4 (2002), pp. 331–370.

[44] Robin Burke. “Interactive Critiquing for Catalog Navigation in E-Commerce”. In: *Artificial Intelligence Review* 18.3-4 (2002), pp. 245–267.

[45] Robin Burke. “Knowledge-based recommender systems”. In: *Encyclopedia of Library and Information Science* 69.Supplement 32 (2000), p. 180.

[46] Robin Burke. “The Wasabi Personal Shopper: A Case-Based Recommender System”. In: *AAAI/IAAI*. 1999, pp. 844–849.

[47] Robin D Burke, Kristian J Hammond, and BC Yound. “The FindMe Approach to Assisted Browsing”. In: *IEEE Expert* 12.4 (1997), pp. 32–40.

[48] Paolo Campigotto, Roberto Battiti, and Andrea Passerini. “Learning Modulo Theories for preference elicitation in hybrid domains”. In: *CoRR* (2015).

[49] Nicolo Cesa-Bianchi, Alex Conconi, and Claudio Gentile. “On the Generalization Ability of On-Line Learning Algorithms”. In: *IEEE Transactions on Information Theory* 50.9 (2004), pp. 2050–2057.

[50] Nicolo Cesa-Bianchi and Gábor Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2006.

[51] Urszula Chajewska. “Acting rationally with incomplete utility information”. PhD thesis. Stanford University, 2002.

[52] Urszula Chajewska, Daphne Koller, and Ronald Parr. “Making Rational Decisions using Adaptive Utility Elicitation”. In: *AAAI/IAAI*. 2000, pp. 363–369.

[53] Olivier Chapelle, Quoc Le, and Alex Smola. “Large margin optimization of ranking measures”. In: *NIPS workshop: Machine learning for Web search*. 2007.

[54] Li Chen and Pearl Pu. “Critiquing-based recommenders: survey and emerging trends”. In: *User Modeling and User-Adapted Interaction* 22.1-2 (2012), pp. 125–150.

[55] Li Chen and Pearl Pu. “Preference-based Organization Interfaces: Aiding User Critiques in Recommender Systems”. In: *International Conference on User Modeling*. Springer. 2007, pp. 77–86.

[56] Li Chen and Pearl Pu. *Survey of Preference Elicitation Methods*. Tech. rep. École Politechnique Fédérale de Lausanne, 2004.

[57] Liang-Chieh Chen, Alexander Schwing, Alan Yuille, and Raquel Urtasun. “Learning Deep Structured Models”. In: *Proceedings of the 32nd International Conference on Machine Learning (ICML)*. 2015, pp. 1785–1794.

[58] Wei Chen, Yajun Wang, and Yang Yuan. “Combinatorial Multi-Armed Bandit: General Framework, Results and Applications”. In: *Proceedings of the 30th International Conference on Machine Learning*. 2013, pp. 151–159.

[59] Pedro S Coelho and Jörg Henseler. “Creating Customer Loyalty through Service Customization”. In: *European Journal of Marketing* 46.3/4 (2012), pp. 331–356.

[60] Michael Collins. “Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms”. In: *Proceedings of the ACL-02 Conference on Empirical methods in Natural Language Processing-Volume 10*. Association for Computational Linguistics. 2002, pp. 1–8.

[61] Michael Collins, Amir Globerson, Terry Koo, Xavier Carreras, and Peter L Bartlett. “Exponentiated Gradient Algorithms for Conditional Random Fields and Max-Margin Markov Networks”. In: *Journal of Machine Learning Research* 9.Aug (2008), pp. 1775–1822.

[62] Michael Collins and Brian Roark. “Incremental Parsing with the Perceptron Algorithm”. In: *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics.* Association for Computational Linguistics. 2004, p. 111.

[63] Simon Colton and Geraint A Wiggins. “Computational Creativity: The Final Frontier?” In: *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI).* Vol. 2012. Montpelier. 2012, pp. 21–16.

[64] Vincent Conitzer. “Eliciting Single-Peaked Preferences Using Comparison Queries”. In: *Journal of Artificial Intelligence Research* 35 (2009), pp. 161–191.

[65] Corinna Cortes and Vladimir Vapnik. “Support-Vector Networks”. In: *Machine Learning* 20.3 (1995), pp. 273–297.

[66] Fabrizio Costa. “Learning an efficient constructive sampler for graphs”. In: *Artificial Intelligence* 244 (2017), pp. 217–238.

[67] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. “Online Passive-Aggressive Algorithms”. In: *Journal of Machine Learning Research* 7.Mar (2006), pp. 551–585.

[68] Koby Crammer and Yoram Singer. “On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines”. In: *Journal of Machine Learning Research* 2.Dec (2001), pp. 265–292.

[69] Marlies De Clercq, Michiel Stock, Bernard De Baets, and Willem Waegeman. “Data-driven recipe completion using machine learning methods”. In: *Trends in Food Science & Technology* 49 (2016), pp. 1–13.

[70] Leonardo De Moura and Nikolaj Bjørner. “Satisfiability Modulo Theories: Introduction and Applications”. In: *Communications of the ACM* 54.9 (2011), pp. 69–77.

[71] Luc De Raedt. “Inductive Logic Programming”. In: *Encyclopedia of Machine Learning.* Springer, 2011, pp. 529–537.

[72] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. “ProbLog: A probabilistic Prolog and its application in link discovery”. In: *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI).* 2007.

[73] Gerard Debreu. “Representation of a preference ordering by a numerical function”. In: *Decision Processes* 3 (1954), pp. 159–165.

[74] Rémi Desmeules. “The Impact of Variety on Consumer Happiness: Marketing and the Tyranny of Freedom”. In: *Academy of Marketing Science Review* 12.1 (2002), pp. 1–18.

[75] Michelangelo Diligenti, Marco Gori, and Claudio Sacca. “Semantic-based regularization for learning and inference”. In: *Artificial Intelligence* 244 (2017), pp. 143–165.

[76] Carmel Domshlak, Eyke Hüllermeier, Souhila Kaci, and Henri Prade. “Preferences in AI: An overview”. In: *Artificial Intelligence* 175.7-8 (2011), pp. 1037–1052.

[77] Carmel Domshlak and Thorsten Joachims. “Efficient and non-parametric reasoning over user preferences”. In: *User Modeling and User-Adapted Interaction* 17.1-2 (2007), pp. 41–69.

[78] Carmel Domshlak and Thorsten Joachims. “Unstructuring User Preferences: Efficient Non-Parametric Utility Revelation”. In: *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence (UAI).* AUAI Press. 2005, pp. 169–177.

[79] Carmel Domshlak, Francesca Rossi, Kristen Brent Venable, and Toby Walsh. “Reasoning about soft constraints and conditional preferences: complexity results and approximation techniques”. In: *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*. 2003.

[80] Jon Doyle and Richmond H Thomason. “Background to Qualitative Decision Theory”. In: *AI magazine* 20.2 (1999), p. 55.

[81] Paolo Dragone. “Constructive Recommendation”. In: *Proceedings of the Eleventh ACM Conference on Recommender Systems*. 2017, pp. 441–445.

[82] Paolo Dragone, Luca Erculiani, Maria Teresa Chietera, Stefano Teso, and Andrea Passerini. “Constructive Layout Synthesis via Coactive Learning”. In: *Constructive Machine Learning Workshop at NIPS*. 2016.

[83] Paolo Dragone, Pellegrini Giovanni, Michele Vescovi, Katya Tentori, and Andrea Passerini. “No More Ready-made Deals: Constructive Recommendation for Telco Service Bundling”. In: *Proceedings of the 12th ACM Conference on Recommender Systems (RecSys)*. 2018.

[84] Paolo Dragone, Stefano Teso, Mohit Kumar, and Andrea Passerini. “Decomposition Strategies for Constructive Preference Elicitation”. In: *Proceedings of the 32st AAAI Conference on Artificial Intelligence*. 2018.

[85] Paolo Dragone, Stefano Teso, and Andrea Passerini. “Constructive Preference Elicitation”. In: *Frontiers in Robotics and AI* 4 (2018).

[86] Paolo Dragone, Stefano Teso, and Andrea Passerini. “Constructive Preference Elicitation over Hybrid Combinatorial Spaces”. In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI)*. 2018.

[87] Paolo Dragone, Stefano Teso, and Andrea Passerini. “Pyconstruct: Constraint Programming meets Structrued Prediction”. In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI)*. 2018.

[88] Xuehong Du, Jianxin Jiao, and Mitchell M Tseng. “Understanding customer satisfaction in product customization”. In: *The International Journal of Advanced Manufacturing Technology* 31.3-4 (2006), pp. 396–406.

[89] Didier Dubois, Hélene Fargier, and Henri Prade. “The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction”. In: *Second IEEE International Conference on Fuzzy Systems*. IEEE. 1993, pp. 1131–1136.

[90] John Duchi, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra. “Efficient Projections onto the ℓ_1-Ball for Learning in High Dimensions”. In: *Proceedings of the 25th International Conference on Machine Learning (ICML)*. ACM. 2008, pp. 272–279.

[91] Miroslav Dudik, Katja Hofmann, Robert E Schapire, Aleksandrs Slivkins, and Masrour Zoghi. “Contextual Dueling Bandits”. In: *Conference on Learning Theory*. 2015, pp. 563–587.

[92] Michael D Ekstrand, John T Riedl, and Joseph A Konstan. “Collaborative Filtering Recommender Systems”. In: *Foundations and Trends® in Human–Computer Interaction* 4.2 (2011), pp. 81–173.

[93] Luca Erculiani, Paolo Dragone, Stefano Teso, and Andrea Passerini. “Automating Layout Synthesis with Constructive Preference Elicitation”. In: *Joint European Conference*

on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD). 2018.

[94] Andreas Falkner, Alexander Felfernig, and Albert Haag. "Recommendation Technologies for Configurable Products". In: *AI magazine* 32.3 (2011), pp. 99–108.

[95] Hélène Fargier, Pierre-François Gimenez, and Jérôme Mengin. "Recommendation for product configuration: an experimental evaluation". In: *18th CP International Configuration Workshop.* 2016, pp–9.

[96] Alexander Felfernig, Ludovico Boratto, Martin Stettinger, and Marko Tkalčič. "Decision Tasks and Basic Algorithms". In: *Group Recommender Systems.* Springer, 2018, pp. 3–26.

[97] Alexander Felfernig and Robin Burke. "Constraint-based Recommender Systems: Technologies and Research Issues". In: *Proceedings of the 10th International Conference on Electronic Commerce.* ACM. 2008, p. 3.

[98] Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, and Markus Zanker. "An Integrated Environment for the Development of Knowledge-Based Recommender Applications". In: *International Journal of Electronic Commerce* 11.2 (2006), pp. 11–34.

[99] Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, and Markus Zanker. "Constraint-Based Recommender Systems". In: *Recommender Systems Handbook.* Springer, 2015, pp. 161–190.

[100] Alexander Felfernig, Lothar Hotz, Claire Bagley, and Juha Tiihonen. *Knowledge-Based Configuration: From Research to Business Cases.* Newnes, 2014.

[101] Alexander Felfernig, Klaus Isak, Kalman Szabo, and Peter Zachar. "The VITA Financial Services Sales Support Environment". In: *Proceedings of the National Conference on Artificial Intelligence.* Vol. 22. 2. 2007, p. 1692.

[102] Alexander Felfernig, Stefan Schippel, Gerhard Leitner, Florian Reinfrank, Klaus Isak, Monika Mandl, Paul Blazek, and Gerald Ninaus. "Automated Repair of Scoring Rules in Constraint-Based Recommender Systems". In: *AI communications* 26.1 (2013), pp. 15–27.

[103] Peter C Fishburn. "Interdependence and additivity in multivariate, unidimensional expected utility theory". In: *International Economic Review* 8.3 (1967), pp. 335–342.

[104] Peter C Fishburn. *Utility Theory for Decision Making.* Tech. rep. Research analysis corp McLean VA, 1970.

[105] Iztok Fister, Samo Rauter, Xin-She Yang, and Karin Ljubič. "Planning the sports training sessions with the bat algorithm". In: *Neurocomputing* 149 (2015), pp. 993–1002.

[106] Abraham D Flaxman, Adam Tauman Kalai, and H Brendan McMahan. "Online convex optimization in the bandit setting: gradient descent without a gradient". In: *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete Algorithms.* Society for Industrial and Applied Mathematics. 2005, pp. 385–394.

[107] Robert H Frank. *Microeconomics and Behavior.* Boston: McGraw-Hill Irwin, 2008.

[108] Paolo Frasconi, Manfred Jaeger, and Andrea Passerini. "Feature Discovery with Type Extension Trees". In: *International Conference on Inductive Logic Programming.* Springer. 2008, pp. 122–139.

[109] Yoav Freund and Robert E Schapire. “Large Margin Classification Using the Perceptron Algorithm”. In: *Machine Learning* 37.3 (1999), pp. 277–296.

[110] Krzysztof Gajos and Daniel S Weld. “Preference Elicitation for Interface Optimization”. In: *Proceedings of the 18th annual ACM symposium on User Interface Software and Technology.* ACM. 2005, pp. 173–182.

[111] Joshua S Gans and Stephen P King. “Paying for Loyalty: Product Bundling in Oligopoly”. In: *The Journal of Industrial Economics* 54.1 (2006), pp. 43–62.

[112] Begoña García-Mariñoso and Xavier Martínez Giralt. “Bundling in telecommunications”. In: (2008).

[113] Mirco Gelain, Maria Silvia Pini, Francesca Rossi, K Brent Venable, and Toby Walsh. “Elicitation strategies for soft constraint problems with missing preferences: Properties, algorithms and experimental studies”. In: *Artificial Intelligence* 174.3 (2010), p. 270.

[114] Lise Getoor and Ben Taskar. *Introduction to Statistical Relational Learning.* Vol. 1. MIT press Cambridge, 2007.

[115] Jon Gillick, Kevin Tang, and Robert M Keller. “Machine Learning of Jazz Grammars”. In: *Computer Music Journal* 34.3 (2010), pp. 56–66.

[116] Robby Goetschalckx, Alan Fern, and Prasad Tadepalli. “Coactive Learning for Locally Optimal Problem Solving”. In: *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence.* 2014, pp. 1824–1830.

[117] Robby Goetschalckx, Alan Fern, and Prasad Tadepalli. “Multitask Coactive Learning”. In: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI).* Vol. 14. 2015, pp. 3518–3524.

[118] Mehmet H Göker and Cynthia A Thompson. “Personalized Conversational Case-Based Recommendation”. In: *Proceedings of the European Conference on Case-Based Reasoning.* Springer. 2000, pp. 99–111.

[119] Judy Goldsmith and Ulrich Junker. “Preference Handling for Artificial Intelligence”. In: *AI Magazine* 29.4 (2008), p. 9.

[120] Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. “Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules”. In: *ACS Central Science* 4.2 (2018), pp. 268–276.

[121] Christophe Gonzales and Patrice Perny. “GAI Networks for Utility Elicitation”. In: *Proceedings of the Ninth International Conference on Principles of Knowledge Representation and Reasoning.* AAAI Press. 2004, pp. 224–233.

[122] P Grasch, A Felfernig, and F Reinfrank. “ReComment: Towards Critiquing-based Recommendation with Speech Interaction”. In: *Proceedings of the ACM conference on Recommendation Systems (RecSys).* 2013, pp. 157–164.

[123] Chiara Grosso, Cipriano Forza, and Alessio Trentin. “Assessing the configurators user need for social interaction during product configuration process”. In: *18th CP International Configuration Workshop.* 2016, p. 29.

[124] Adam J Grove, Nick Littlestone, and Dale Schuurmans. “General Convergence Results for Linear Discriminant Updates”. In: *Machine Learning* 43.3 (2001), pp. 173–210.

[125] Tias Guns, Anton Dries, Guido Tack, Siegfried Nijssen, and Luc De Raedt. “MiningZinc: A Modeling Language for Constraint-Based Mining”. In: *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence.* Vol. 13. 2013, pp. 1365–1372.

[126] Andreas Günter and Christian Kühn. “Knowledge-Based Configuration: Survey and Future Directions”. In: *German Conference on Knowledge-Based Systems.* Springer. 1999, pp. 47–66.

[127] Liu Guo-rong and Zhang Xi-zheng. “Collaborative Filtering based Recommendation System for Product Bundling”. In: *International Conference on Management Science and Engineering.* IEEE. 2006, pp. 251–254.

[128] Shengbo Guo and Scott Sanner. “Real-time Multiattribute Bayesian Preference Elicitation with Pairwise Comparison Queries”. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS).* 2010, pp. 289–296.

[129] Shengbo Guo, Scott Sanner, and Edwin V Bonilla. “Gaussian Process Preference Elicitation”. In: *Advances in Neural Information Processing Systems.* 2010, pp. 262–270.

[130] Bernd Gutmann, Manfred Jaeger, and Luc De Raedt. “Extending ProbLog with Continuous Distributions”. In: *International Conference on Inductive Logic Programming.* Springer. 2010, pp. 76–91.

[131] Mikako Harada, Andrew Witkin, and David Baraff. “Interactive Physically-Based Manipulation of Discrete/Continuous Models”. In: *Proceedings of the 22nd annual conference on Computer Graphics and Interactive Techniques.* ACM. 1995, pp. 199–208.

[132] Elad Hazan. “Introduction to Online Convex Optimization”. In: *Foundations and Trends® in Optimization* 2.3-4 (2016), pp. 157–325.

[133] Elad Hazan, Adam Kalai, Satyen Kale, and Amit Agarwal. “Logarithmic regret algorithms for online convex optimization”. In: *Machine Learning* 69 ().

[134] Elad Hazan, Adam Kalai, Satyen Kale, and Amit Agarwal. “Logarithmic regret algorithms for online convex optimization”. In: *International Conference on Computational Learning Theory.* Springer. 2006, pp. 499–513.

[135] Xuming He, Richard S Zemel, and Miguel Á Carreira-Perpiñán. “Multiscale Conditional Random Fields for Image Labeling”. In: *Proceedings of the 2004 IEEE computer society conference on Computer Vision and Pattern Recognition.* Vol. 2. IEEE. 2004, pp. II–II.

[136] Andreas Herrmann, Frank Huber, and Robin Higie Coulter. “Product and Service Bundling Decisions and their Effects on Purchase Intention”. In: *Pricing Strategy and Practice* 5.3 (1997), pp. 99–107.

[137] Greg Hines and Kate Larson. “Efficiently Eliciting Preferences from a Group of Users”. In: *International Conference on Algorithmic Decision Theory.* Springer. 2011, pp. 96–107.

[138] Lothar Hotz, Alexander Felfernig, Markus Stumptner, Anna Ryabokon, Claire Bagley, and Katharina Wolter. “Configuration Knowledge Representation and Reasoning”. In: *Knowledge-Based Configuration: From Research to Business Cases.* 2014, pp. 41–72.

[139] Cynthia Huffman and Barbara E Kahn. “Variety for Sale: Mass Customization or Mass Confusion?” In: *Journal of Retailing* 74.4 (1998), pp. 491–513.

[140] Ashesh Jain, Shikhar Sharma, Thorsten Joachims, and Ashutosh Saxena. “Learning preferences for manipulation tasks from online coactive feedback”. In: *The International Journal of Robotics Research* 34.10 (2015), pp. 1296–1313.

[141] Ashesh Jain, Brian Wojcik, Thorsten Joachims, and Ashutosh Saxena. “Learning Trajectory Preferences for Manipulators via Iterative Improvement”. In: *Advances in Neural Information Processing Systems*. 2013, pp. 575–583.

[142] Dietmar Jannach, Markus Zanker, and Matthias Fuchs. “Constraint-based recommendation in tourism: A multiperspective case study”. In: *Information Technology & Tourism* 11.2 (2009), pp. 139–155.

[143] Yangbangyan Jiang, XU Qianqian, and Xiaochun Cao. “Outfit Recommendation with Deep Sequence Learning”. In: *2018 IEEE Fourth International Conference on Multimedia Big Data (BigMM)*. IEEE. 2018, pp. 1–5.

[144] Thorsten Joachims. “Optimizing Search Engines using Clickthrough Data”. In: *Proceedings of the eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2002, pp. 133–142.

[145] Thorsten Joachims, Thomas Finley, and Chun-Nam John Yu. “Cutting-plane training of structural SVMs”. In: *Machine Learning* 77.1 (2009), pp. 27–59.

[146] Thorsten Joachims, Thomas Hofmann, Yisong Yue, and Chun-Nam Yu. “Predicting Structured Objects with Support Vector Machines”. In: *Communications of the ACM* 52.11 (2009), pp. 97–104.

[147] Daniel Kahneman and Amos Tversky. “The Psychology of Preferences”. In: *Scientific American* 246.1 (1982), pp. 160–173.

[148] Alexandros Karatzoglou, Linas Baltrunas, and Yue Shi. “Learning to Rank for Recommender Systems”. In: *Proceedings of the 7th ACM conference on Recommender Systems (RecSys)*. ACM. 2013, pp. 493–494.

[149] Narendra Karmarkar. “A New Polynomial-Time Algorithm for Linear Programming”. In: *Proceedings of the sixteenth annual ACM Symposium on Theory of Computing*. ACM. 1984, pp. 302–311.

[150] Ralph L Keeney and Howard Raiffa. *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. 1976.

[151] Robert M Keller and David R Morrison. “A Grammatical Approach to Automatic Improvisation”. In: *Proceedings of the 4th Sound and Music Computing Conference*. 2007.

[152] Jyrki Kivinen and Manfred K Warmuth. “Exponentiated Gradient versus Gradient Descent for Linear Predictors”. In: *Information and Computation* 132.1 (1997), pp. 1–63.

[153] Yehuda Koren and Robert Bell. “Advances in Collaborative Filtering”. In: *Recommender Systems Handbook*. Springer, 2015, pp. 77–118.

[154] Frédéric Koriche and Bruno Zanuttini. “Learning conditional preference networks”. In: *Artificial Intelligence* 174.11 (2010), pp. 685–703.

[155] Philip Kotler. “From Mass Marketing to Mass Customization”. In: *Planning review* 17.5 (1989), pp. 10–47.

[156] Jan Krämer. *Bundling Telecommunications Services: Competitive Strategies for Converging Markets.* Vol. 10. KIT Scientific Publishing, 2009.

[157] Simon Lacoste-Julien, Martin Jaggi, Mark Schmidt, and Patrick Pletscher. "Block-Coordinate Frank-Wolfe Optimization for Structural SVMs". In: *Proceedings of the 30th International Conference on Machine Learning (ICML).* 2013, pp. 53–61.

[158] John Lafferty, Andrew McCallum, and Fernando CN Pereira. "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data". In: *roceedings of the Eighteenth International Conference on Machine Learning (ICML).* 2001.

[159] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep Learning". In: *Nature* 521.7553 (2015), p. 436.

[160] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. "A Contextual-Bandit Approach to Personalized News Article Recommendation". In: *Proceedings of the 19th International Conference on World Wide Web.* 2010, pp. 661–670.

[161] Lihong Li, Wei Chu, John Langford, and Xuanhui Wang. "Unbiased Offline Evaluation of Contextual-bandit-based News Article Recommendation Algorithms". In: *Proceedings of the 4th ACM International Conference on Web Search and Data Mining.* 2011, pp. 297–306.

[162] Percy Liang and Christopher Potts. "Bringing machine learning and compositional semantics together". In: *Annual Review of Linguistics* 1.1 (2015), pp. 355–376.

[163] Sarah Lichtenstein and Paul Slovic. *The Construction of Preference.* 2006.

[164] Blerina Lika, Kostas Kolomvatsos, and Stathes Hadjiefthymiades. "Facing the cold start problem in recommender systems". In: *Expert Systems with Applications* 41.4 (2014), pp. 2065–2073.

[165] Greg Linden, Steve Hanks, and Neal Lesh. "Interactive Assessment of User Preference Models: The Automated Travel Assistant". In: *User Modeling.* Springer. 1997, pp. 67–78.

[166] Nick Littlestone. "Learning Quickly When Irrelevant Attributes Abound: A New Linear-threshold Algorithm". In: *Machine learning* 2.4 (1988), pp. 285–318.

[167] Tianqiang Liu, Siddhartha Chaudhuri, Vladimir G Kim, Qixing Huang, Niloy J Mitra, and Thomas Funkhouser. "Creating Consistent Scene Graphs Using a Probabilistic Grammar". In: *ACM Transactions on Graphics (TOG)* 33.6 (2014), p. 211.

[168] Tie-Yan Liu. "Learning to Rank for Information Retrieval". In: *Foundations and Trends® in Information Retrieval* 3.3 (2009), pp. 225–331.

[169] Andrew J Lloyd. "Threats to the estimation of benefit: are preference elicitation methods accurate?" In: *Health Economics* 12.5 (2003), pp. 393–402.

[170] George Loewenstein. "Emotions in Economic Theory and Economic Behavior". In: *American Economic Review* 90.2 (2000), pp. 426–432.

[171] Santiago Londoño and Olana Missura. "Graph Grammars for Super Mario Bros Levels". In: *Procedural Content Generation workshop, Foundation of Digital Games.* 2015.

[172] Pasquale Lops, Marco De Gemmis, and Giovanni Semeraro. "Content-based Recommender Systems: State of the Art and Trends". In: *Recommender Systems Handbook.* Springer, 2011, pp. 73–105.

[173] Jordan J Louviere, David A Hensher, and Joffre D Swait. *Stated Choice Methods: Analysis and Applications.* Cambridge university press, 2000.

[174] Tyler Lu and Craig Boutilier. “Robust Approximation and Incremental Elicitation in Voting Protocols”. In: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI).* Vol. 1. 2011, pp. 287–293.

[175] R Duncan Luce. *Individual choice behavior: A theoretical analysis.* 1959.

[176] Matteo Luperto and Francesco Amigoni. “Predicting the global structure of indoor environments: A constructive machine learning approach”. In: *Autonomous Robots* (2018), pp. 1–23.

[177] Christopher D Manning, Christopher D Manning, and Hinrich Schütze. *Foundations of Statistical Natural Language Processing.* MIT press, 1999.

[178] Raúl Mazo, Cosmin Dumitrescu, Camille Salinesi, and Daniel Diaz. “Recommendation Heuristics for Improving Product Line Configuration Processes”. In: *Recommendation Systems in Software Engineering.* Springer, 2014, pp. 511–537.

[179] Peter McCullagh and John A Nelder. *Generalized Linear Models.* Vol. 37. CRC press, 1989.

[180] Daniel McFadden. “Economic Choices”. In: *American Economic Review* 91.3 (2001), pp. 351–378.

[181] Lorraine McGinty and James Reilly. “On the Evolution of Critiquing Recommenders”. In: *Recommender Systems Handbook.* Springer, 2011, pp. 419–453.

[182] Paul Merrell, Eric Schkufza, Zeyang Li, Maneesh Agrawala, and Vladlen Koltun. “Interactive Furniture Layout Using Interior Design Guidelines”. In: *ACM Transactions on Graphics (TOG).* Vol. 30. 4. ACM. 2011, p. 87.

[183] Pedro Meseguer, Francesca Rossi, and Thomas Schiex. “Soft Constraints”. In: *Foundations of Artificial Intelligence.* Vol. 2. Elsevier, 2006, pp. 281–328.

[184] Jeremy Michalek and Panos Papalambros. “Interactive design optimization of architectural layouts”. In: *Engineering Optimization* 34.5 (2002), pp. 485–501.

[185] William J Mitchell, J Philip Steadman, and Robin S Liggett. “Synthesis and Optimization of Small Rectangular Floor Plans”. In: *Environment and Planning B: Planning and Design* 3.1 (1976), pp. 37–70.

[186] Oskar Morgenstern and John Von Neumann. *Theory of Games and Economic Behavior.* Princeton university press, 1953.

[187] Roger B Myerson. *Game Theory.* Harvard university press, 2013.

[188] Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck, and Guido Tack. “MiniZinc: Towards a Standard CP Modelling Language”. In: *International Conference on Principles and Practice of Constraint Programming.* Springer. 2007, pp. 529–543.

[189] Albert B Novikoff. *On Convergence Proofs for Perceptrons.* Tech. rep. 1963.

[190] Sebastian Nowozin and Christoph H Lampert. “Structured Learning and Prediction in Computer Vision”. In: *Foundations and Trends® in Computer Graphics and Vision* 6.3–4 (2011), pp. 185–365.

[191] Dino Oglic, Roman Garnett, and Thomas Gärtner. “Active Search in Intensionally Specified Structured Spaces”. In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI).* 2017, pp. 2443–2449.

[192] Dino Oglic, Roman Garnett, and Thomas Gärtner. “Learning to Construct Novel Structures”. In: (2014).

[193] Dino Oglic, Steven A Oatley, Simon JF Macdonald, Thomas Mcinally, Roman Garnett, Jonathan D Hirst, and Thomas Gärtner. “Active Search for Computer-aided Drug Design”. In: *Molecular Informatics* 37.1-2 (2018), p. 1700130.

[194] Ivan Olier, Noureddin Sadawi, G Richard Bickerton, Joaquin Vanschoren, Crina Grosan, Larisa Soldatova, and Ross D King. “Meta-QSAR: a large-scale application of meta-learning to drug design and discovery”. In: *Machine Learning* 107.1 (2018), pp. 285–311.

[195] AJ Orman and H Paul Williams. “A Survey of Different Integer Programming Formulations of the Travelling Salesman Problem”. In: *Optimisation, Econometric and Financial Analysis* 9 (2006), pp. 93–108.

[196] Pedro A Ortega and Alan A Stocker. “Human Decision-Making under Limited Time”. In: *Advances in Neural Information Processing Systems.* 2016, pp. 100–108.

[197] Anton Osokin, Jean-Baptiste Alayrac, Isabella Lukasewitz, Puneet K Dokania, and Simon Lacoste-Julien. “Minding the Gaps for Block Frank-Wolfe Optimization of Structured SVMs”. In: *Proceedings of the 33rd International Conference on Machine Learning (ICML).* 2016.

[198] François Pachet, Alexandre Papadopoulos, and Pierre Roy. “Sampling Variations of Sequences for Structured Music Generation”. In: *Proceedings of the 18th International Society for Music Information Retrieval Conference.* 2017, pp. 167–173.

[199] Julius Panero and Martin Zelnik. *Human dimension and interior space: a source book of design reference standards.* 1979.

[200] Christos H Papadimitriou. “On the Complexity of Integer Programming”. In: *Journal of the ACM (JACM)* 28.4 (1981), pp. 765–768.

[201] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Courier Corporation, 1998.

[202] John W Payne, James R Bettman, and Eric J Johnson. “Behavioral decision research: A constructive processing perspective”. In: *Annual Review of Psychology* 43.1 (1992), pp. 87–131.

[203] John W Payne, James R Bettman, and Eric J Johnson. *The adaptive decision maker.* Cambridge University Press, 1993.

[204] Michael J Pazzani and Daniel Billsus. “Content-Based Recommendation Systems”. In: *The Adaptive Web.* Springer, 2007, pp. 325–341.

[205] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, 1988.

[206] Gabriella Pigozzi, Alexis Tsoukias, and Paolo Viappiani. “Preferences in artificial intelligence”. In: *Annals of Mathematics and Artificial Intelligence* 77.3-4 (2016), pp. 361–401.

[207] B Joseph Pine and Richard Hull. *Mass customization: the new frontier in business competition.* Vol. 288. Harvard Business School Press, 1995.

[208] Robin L Plackett. “The Analysis of Permutations”. In: *Applied Statistics* (1975), pp. 193–202.

[209] BT Poljak and Ya Z Tsypkin. “Pseudogradient adaptation and training algorithms”. In: *Automation and Remote Control* 34 (1973), pp. 45–67.

[210] Pearl Pu and Li Chen. “Integrating Tradeoff Support in Product Search Tools for E-Commerce Sites”. In: *Proceedings of the 6th ACM conference on Electronic Commerce.* ACM. 2005, pp. 269–278.

[211] Pearl Pu and Boi Faltings. “Decision Tradeoff Using Example-Critiquing and Constraint Programming”. In: *Constraints* 9.4 (2004), pp. 289–310.

[212] Lijing Qin, Shouyuan Chen, and Xiaoyan Zhu. “Contextual Combinatorial Bandit and its Application on Diversified Online Recommendation”. In: *Proceedings of the 2014 SIAM International Conference on Data Mining.* SIAM. 2014, pp. 461–469.

[213] Luc De Raedt, Andrea Passerini, and Stefano Teso. “Learning Constraints from Examples”. In: *Proceedings of the 32nd Conference on Artificial Intelligence (AAAI).* 2018.

[214] Karthik Raman and Thorsten Joachims. “Learning Socially Optimal Information Systems from Egoistic Users”. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PDKK).* Springer. 2013, pp. 128–144.

[215] Karthik Raman, Thorsten Joachims, Pannaga Shivaswamy, and Tobias Schnabel. “Stable Coactive Learning via Perturbation”. In: *Proceedings of the 30th International Conference on Machine Learning.* 2013, pp. 837–845.

[216] Karthik Raman, Pannaga Shivaswamy, and Thorsten Joachims. “Online Learning to Diversify from Implicit Feedback”. In: *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM. 2012, pp. 705–713.

[217] Nathan D Ratliff, J Andrew Bagnell, and Martin A Zinkevich. “(Online) Subgradient Methods for Structured Prediction”. In: *Artificial Intelligence and Statistics.* 2007, pp. 380–387.

[218] James Reilly, Kevin McCarthy, Lorraine McGinty, and Barry Smyth. “Dynamic Critiquing”. In: *Proceedings of the European Conference on Case-Based Reasoning.* Springer. 2004, pp. 763–777.

[219] James Reilly, Kevin McCarthy, Lorraine McGinty, and Barry Smyth. “Incremental Critiquing”. In: *Knowledge-Based System* (2005), pp. 101–114.

[220] Francesco Ricci, Lior Rokach, and Bracha Shapira. *Recommender Systems Handbook.* 2011.

[221] Francesco Ricci, Lior Rokach, and Bracha Shapira. “Recommender Systems: Introduction and Challenges”. In: *Recommender Systems Handbook.* Springer, 2011, pp. 1–34.

[222] Matthew Richardson and Pedro Domingos. “Markov logic networks”. In: *Machine Learning* 62.1-2 (2006), pp. 107–136.

[223] Frank Rosenblatt. “The Perceptron: A probabilistic model for information storage and organization in the brain”. In: *Psychological Review* 65.6 (1958), p. 386.

[224] Azriel Rosenfeld, Robert A Hummel, and Steven W Zucker. "Scene Labeling by Relaxation Operations". In: *IEEE Transactions on Systems, Man, and Cybernetics* 6 (1976), pp. 420–433.

[225] Francesca Rossi and Alessandro Sperduti. "Learning Solution Preferences in Constraint Problems". In: *Journal of Experimental & Theoretical Artificial Intelligence* 10.1 (1998), pp. 103–116.

[226] Francesca Rossi, Alessandro Sperduti, Kristen B Venable, Lina Khatib, Paul Morris, and Robert Morris. "Learning and Solving Soft Temporal Constraints: An Experimental Study". In: *International Conference on Principles and Practice of Constraint Programming (CP).* Springer. 2002, pp. 249–264.

[227] Francesca Rossi and Allesandro Sperduti. "Acquiring Both Constraint and Solution Preferences in Interactive Constraint Systems". In: *Constraints* 9.4 (2004), pp. 311–332.

[228] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of Constraint Programming.* Elsevier, 2006.

[229] Ariel Rubinstein. *Modeling Bounded Rationality.* 1998.

[230] Zsofi Ruttkay. "Fuzzy Constraint Satisfaction". In: *Proceedings of the Third IEEE Conference on Fuzzy Systems.* IEEE. 1994, pp. 1263–1268.

[231] Avishek Saha, Piyush Rai, Hal DaumÃ, and Suresh Venkatasubramanian. "Online Learning of Multiple Tasks and Their Relationships". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS).* 2011, pp. 643–651.

[232] Camille Salinesi, Raouia Triki, and Raul Mazo. "Combining configuration and recommendation to define an interactive product line configuration approach". In: *Journée d'étude sur les moteurs de recommandation.* 2012.

[233] Leonard J Savage. *The foundations of statistics.* Courier Corporation, 1972.

[234] Leonard J Savage. "The Theory of Statistical Decision". In: *Journal of the American Statistical Association* 46.253 (1951), pp. 55–67.

[235] Amihai Savir, Ronen Brafman, and Guy Shani. "Recommending improved configurations for complex objects with an application in travel planning". In: *Proceedings of the 7th ACM conference on Recommender systems.* ACM. 2013, pp. 391–394.

[236] Andrew I Schein, Alexandrin Popescul, Lyle H Ungar, and David M Pennock. "Methods and Metrics for Cold-Start Recommendations". In: *Proceedings of the 25th annual International ACM SIGIR Conference on Research and Development in Information Retrieval.* ACM. 2002, pp. 253–260.

[237] Bernhard Scholkopf and Alexander J Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* MIT press, 2001.

[238] Alexander Schrijver. *Theory of Linear and Integer Programming.* John Wiley & Sons, 1998.

[239] Barry Schwartz. *The Paradox of Choice: Why More Is Less.* 2004.

[240] Roberto Sebastiani and Silvia Tomasi. "Optimization Modulo Theories with Linear Rational Costs". In: *ACM Transactions on Computational Logic (TOCL)* 16.2 (2015), p. 12.

[241] Roberto Sebastiani and Patrick Trentin. “OptiMathSAT: A Tool for Optimization Modulo Theories”. In: *International Conference on Computer Aided Verification*. Springer. 2015, pp. 447–454.

[242] Shai Shalev-Shwartz. “Online Learning and Online Convex Optimization”. In: *Foundations and Trends® in Machine Learning* 4.2 (2012), pp. 107–194.

[243] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge university press, 2014.

[244] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. “Pegasos: primal estimated sub-gradient solver for SVM”. In: *Mathematical Programming* 127.1 (2011), pp. 3–30.

[245] Guy Shani and Asela Gunawardana. “Evaluating Recommendation Systems”. In: *Recommender Systems Handbook*. Springer, 2011, pp. 257–297.

[246] Pannaga K Shivaswamy and Thorsten Joachims. “Online Learning with Preference Feedback”. In: *NIPS Workshop on Choice Models and Preference Learning*. 2011.

[247] Pannaga Shivaswamy and Thorsten Joachims. “Coactive Learning”. In: *Journal of Artificial Intelligence Research* 53 (2015), pp. 1–40.

[248] Pannaga Shivaswamy and Thorsten Joachims. “Online Structured Prediction via Coactive Learning”. In: *Proceedings of the 29th International Conference on Machine Learning*. 2012, pp. 59–66.

[249] Paul Slovic. “The Construction of Preference”. In: *American Psychologist* 50.5 (1995), p. 364.

[250] Artem Sokolov, Stefan Riezler, and Shay B Cohen. “Coactive Learning for Interactive Machine Translation”. In: *Proceedings of the 4th Workshop on Machine Learning for Interactive Systems*. 2015, pp. 41–45.

[251] Ivan E Sutherland. “Sketchpad: A man-machine graphical communication system”. In: *Transactions of the Society for Computer Simulation* 2.5 (1964), R–3.

[252] Ben Taskar, Carlos Guestrin, and Daphne Koller. “Max-Margin Markov Networks”. In: *Advances in Neural Information Processing Systems*. 2004, pp. 25–32.

[253] Ben Taskar, Dan Klein, Mike Collins, Daphne Koller, and Christopher Manning. “Max-Margin Parsing”. In: *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*. 2004.

[254] Ching-I Teng. “Customization, immersion satisfaction, and online gamer loyalty”. In: *Computers in Human Behavior* 26.6 (2010), pp. 1547–1554.

[255] Stefano Teso, Paolo Dragone, and Andrea Passerini. “Coactive Critiquing: Elicitation of Preferences and Features”. In: *Proceedings of the 31st AAAI Conference on Artificial Intelligence*. 2017, pp. 2639–2645.

[256] Stefano Teso, Paolo Dragone, and Andrea Passerini. “Structured Feedback for Preference Elicitation in Complex Domains”. In: *BeyondLabeler Workshop at IJCAI 2016*. 2016.

[257] Stefano Teso, Andrea Passerini, and Paolo Viappiani. “Constructive Preference Elicitation by Setwise Max-Margin Learning”. In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI)*. 2016.

[258] Stefano Teso, Andrea Passerini, and Paolo Viappiani. “Constructive Preference Elicitation for Multiple Users with Setwise Max-margin”. In: *International Conference on Algorithmic Decision Theory.* Springer. 2017, pp. 3–17.

[259] Stefano Teso, Roberto Sebastiani, and Andrea Passerini. “Structured learning modulo theories”. In: *Artificial Intelligence* 244 (2015), pp. 166–187.

[260] Sriram Thirumalai and Kingshuk K Sinha. “Customization of the online purchase process in electronic retailing and customer satisfaction: An online field study”. In: *Journal of Operations Management* 29.5 (2011), pp. 477–487.

[261] Robert Tibshirani. “Regression Shrinkage and Selection via the Lasso”. In: *Journal of the Royal Statistical Society. Series B (Methodological)* (1996), pp. 267–288.

[262] William F Tidd, James R Rinderle, and Andrew Witkin. “Design refinement via interactive manipulation of design parameters and behaviors”. In: *Proceedings of the ASME Design Theory and Methodology Conference.* 1992.

[263] Juha Tiihonen. “Characterization of configuration knowledge bases”. In: *19th European Conferenceon Artificial Intelligence.* 2010, p. 13.

[264] Juha Tiihonen and Alexander Felfernig. “Towards recommending configurable offerings”. In: *International Journal of Mass Customisation* 3.4 (2010), pp. 389–406.

[265] Juha Tiihonen, Alexander Felfernig, and Monika Mandl. “Personalized Configuration”. In: *Knowledge-Based Configuration: From Research to Business Cases.* 2014, pp. 167–179.

[266] Marc Torrens, Boi Faltings, and Pearl Pu. “SmartClients: Constraint Satisfaction as a Paradigm for Scaleable Intelligent Information Systems”. In: *Constraints* 7.1 (2002), pp. 49–69.

[267] Alessio Trentin, Elisa Perin, and Cipriano Forza. “Sales configurator capabilities to avoid the product variety paradox: Construct development and validation”. In: *Computers in Industry* 64.4 (2013), pp. 436–447.

[268] Edward Tsang. *Foundations of Constraint Satisfaction.* 1995.

[269] Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. “Support Vector Machine Learning for Interdependent and Structured Output Spaces”. In: *Proceedings of the twenty-first International Conference on Machine Learning.* ACM. 2004, p. 104.

[270] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. “Large Margin Methods for Structured and Interdependent Output Variables”. In: *Journal of Machine Learning Research* 6 (2005), pp. 1453–1484.

[271] Aaron Van den Oord, Sander Dieleman, and Benjamin Schrauwen. “Deep Content-based Music Recommendation”. In: *Advances in Neural Information Processing Systems.* 2013, pp. 2643–2651.

[272] Paolo Viappiani and Craig Boutilier. “Optimal Bayesian Recommendation Sets and Myopically Optimal Choice Query Sets”. In: *Advances in Neural Information Processing Systems.* 2010, pp. 2352–2360.

[273] Paolo Viappiani and Craig Boutilier. “Recommendation Sets and Choice Queries: There Is No Exploration/Exploitation Tradeoff!” In: *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence.* 2011.

[274] Paolo Viappiani and Craig Boutilier. "Regret-based Optimal Recommendation Sets in Conversational Recommender Systems". In: *Proceedings of the third ACM conference on Recommender Systems (RecSys)*. ACM. 2009, pp. 101–108.

[275] Paolo Viappiani and Boi Faltings. "Preference-based Search for Configurable Catalogs". In: *AAAI Workshop on Configutation*. 2007.

[276] Paolo Viappiani, Boi Faltings, and Pearl Pu. "Evaluating Preference-based Search Tools: a Tale of Two Approaches". In: *Proceedings of the Twenty-first National Conference on Artificial Intelligence (AAAI)*. 2006.

[277] Paolo Viappiani, Boi Faltings, and Pearl Pu. "Preference-based Search using Example-Critiquing with Suggestions". In: *Journal of Artificial Intelligence Research* 27 (2006), pp. 465–503.

[278] Paolo Viappiani and Christian Kroer. "Robust Optimization of Recommendation Sets with the Maximin Utility Criterion". In: *International Conference on Algorithmic Decision Theory*. Springer. 2013, pp. 411–424.

[279] Paolo Viappiani, Pearl Pu, and Boi Faltings. "Conversational Recommenders with Adaptive Suggestions". In: *Proceedings of the 1st ACM conference on Recommender Systems (RecSys)*. ACM. 2007, pp. 89–96.

[280] Jue Wang and Pedro M Domingos. "Hybrid Markov Logic Networks". In: *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*. Vol. 8. 2008, pp. 1106–1111.

[281] Wayne L Winston and Jeffrey B Goldberg. *Operations research: applications and algorithms*. Vol. 3. Thomson Brooks/Cole Belmont, 2004.

[282] Wenzhuo Xu, Bin Wang, and Dong-Ming Yan. "Wall grid structure for interior scene synthesis". In: *Computers & Graphics* 46 (2015), pp. 231–243.

[283] Zhao Yan, Nan Duan, Peng Chen, Ming Zhou, Jianshe Zhou, and Zhoujun Li. "Building Task-Oriented Dialogue Systems for Online Shopping". In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI)*. 2017, pp. 4618–4626.

[284] Yi-Ting Yeh, Lingfeng Yang, Matthew Watson, Noah D Goodman, and Pat Hanrahan. "Synthesizing Open Worlds with Constraints using Locally Annealed Reversible Jump MCMC". In: *ACM Transactions on Graphics (TOG)* 31.4 (2012), p. 56.

[285] Lap-Fai Yu, Sai Kit Yeung, Chi-Keung Tang, Demetri Terzopoulos, Tony F Chan, and Stanley Osher. "Make it Home: Automatic Optimization of Furniture Arrangement". In: *ACM Transactions on Graphics (TOG)* 30.4 (2011), pp. 86–1.

[286] Yisong Yue and Thorsten Joachims. "Interactively Optimizing Information Retrieval Systems as a Dueling Bandits Problem". In: *Proceedings of the 26th International Conference on Machine Learning*. 2009, pp. 1201–1208.

[287] Markus Zanker, Markus Aschinger, and Markus Jessenitschnig. "Constraint-based personalised configuring of product and service bundles". In: *International Journal of Mass Customisation* 3.4 (2010), pp. 407–425.

[288] Markus Zanker and Markus Jessenitschnig. "Case-studies on exploiting explicit customer requirements in recommender systems". In: *User Modeling and User-Adapted Interaction* 19.1-2 (2009), pp. 133–166.

[289] Markus Zanker, Markus Jessenitschnig, and Wolfgang Schmid. “Preference reasoning with soft constraints in constraint-based recommender systems”. In: *Constraints* 15.4 (2010), pp. 574–595.

[290] Yingfeng Zhao and Sanyang Liu. “Global optimization algorithm for mixed integer quadratically constrained quadratic program”. In: *Journal of Computational and Applied Mathematics* 319 (2017), pp. 159–169.

[291] Tao Zhu, Patrick Harrington, Junjun Li, and Lei Tang. “Bundle Recommendation in eCommerce”. In: *Proceedings of the 37th international ACM SIGIR conference on Research & Development in Information Retrieval.* ACM. 2014, pp. 657–666.

[292] Martin Zinkevich. “Online Convex Programming and Generalized Infinitesimal Gradient Ascent”. In: *Proceedings of the 20th International Conference on Machine Learning (ICML).* 2003, pp. 928–936.