EFFICIENT CALCULATION OF OPTIMAL CONFIGURATION PROCESSES

YASSER GONZALEZ FERNANDEZ

A THESIS SUBMITTED TO
THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF ARTS

GRADUATE PROGRAM IN INFORMATION SYSTEMS & TECHNOLOGY
YORK UNIVERSITY
TORONTO, ONTARIO

August 2015

Abstract

Customers are getting increasingly involved in the design of the products and services they choose by specifying their desired characteristics. As a result, configuration systems have become essential technologies to support the development of mass-customization business models. These technologies facilitate the configuration of complex products and services that otherwise could generate many incorrect configurations and overwhelm users with confusion. This thesis studies the problem of optimizing the user interaction in a configuration process—as in minimizing the number of questions asked to a user in order to obtain a fully-specified product or service configuration. The work carried out builds upon a previously existing framework to optimize the process of configuring a software system, and focuses on improving its efficiency and generalizing its application to a wider range of configuration domains. Two solution methods along with two alternative ways of specifying the configuration models are proposed and studied on different configuration scenarios. The experimental study evidences that the introduced solutions overcome the limitations of the existing framework, resulting in more suitable algorithms to work with models involving a large number of configuration variables.

Acknowledgements

First and foremost, I would like to thank my supervisors Sotirios Liaskos and Stephen Chen for their guidance and support through the research that has gone into this thesis. It has been an intense, but very rewarding learning experience. I feel privileged to have worked with them.

I would like to express my gratitude to the examining committee members, Suprakash Datta and Zhen Ming (Jack) Jiang, for their much appreciated time and constructive feedback. My sincere thanks also go to the faculty and staff in the School of Information Technology, especially to the graduate program director Zijiang Yang and administrative assistant Ellis Lau.

Finally, I want to thank my parents Mercedes and Noel, and my wife Susana for their essential encouragement and support during the difficult times.

Yasser Gonzalez Fernandez
Toronto, Canada
August 21, 2015

Contents

Abstract **ii**

Acknowledgements **iii**

Table of Contents **iv**

List of Tables **vi**

List of Figures **vii**

List of Abbreviations **viii**

1 Introduction **1**

1.1 Knowledge-Based Configuration 1

1.2 Precedent and Objectives 2

1.3 Outline of the Proposed Solutions 4

1.4 Thesis Structure 5

2 Background **6**

2.1 Configuration Knowledge Representation 6

2.1.1 Rule-Based Representation 6

2.1.2 Constraint-Based Representation 8

2.1.3 Feature Models and Decision Models 10

2.2 Sequential Decision Making Under Uncertainty 12

2.2.1 Markov Decision Processes 12

2.2.2 Dynamic Programming and Value Iteration 13

2.2.3 Value-Based Reinforcement Learning 15
2.2.4 Evolutionary Algorithms for Reinforcement Learning 19
2.3 Summary 21

3 Proposed Solutions 22
3.1 Configuration Models 22
3.2 Markov Decision Process Formulation 25
3.3 Solution Methods 28
3.3.1 Simulation of a Configuration Episode 29
3.3.2 Neural Fitted Q-iteration 29
3.3.3 Direct Policy Search Using a Genetic Algorithm 34
3.4 Summary 38

4 Evaluation 40
4.1 Goals and Experimental Setup 40
4.2 Rule-Based Configuration Models 43
4.2.1 Scalability Results 43
4.2.2 Case Study: Facebook Privacy Settings 46
4.3 Constraint-Based Configuration Models 52
4.3.1 Benchmark Problems 52
4.3.2 Performance Results 55
4.3.3 Scalability Results 58
4.4 Summary 59

5 Conclusions 61
5.1 Summary 61
5.2 Contributions 62
5.3 Related Work and Discussion 63
5.4 Future Work 65

Bibliography 66

A Algorithm Parameters 73

List of Tables

4.1 Facebook privacy settings 47

4.2 Results on the Facebook privacy settings problem 50

4.3 Characteristics of the constraint-based benchmark problems 52

4.4 Results on the constraint-based benchmark problems 56

4.5 Statistical analysis of the results on the constraint-based benchmark problems . . 57

List of Figures

2.1 Example of a feature model for a cellphone 11
2.2 The reinforcement learning scenario 15
2.3 Main loop of neural fitted Q-iteration 19
2.4 Pseudocode of a genetic algorithm 20

3.1 Configuration process with a rule-based configuration model 24
3.2 Configuration process with a constraint-based configuration model 24
3.3 Feedforward neural network with a single hidden layer 30
3.4 Plot of the hyperbolic tangent activation function 31
3.5 Illustration of the neural network used in neural fitted Q-iteration 33
3.6 Offspring generation using the precedence preservative crossover operator 37

4.1 Scalability results on the rule-based configuration models 44
4.2 Best attainable performance on the Facebook privacy settings problem 49
4.3 Individual cross-validation results on the Facebook privacy settings problem . . . 50
4.4 Combined cross-validation results on the Facebook privacy settings problem . . . 51
4.5 Best attainable performance on the constraint-based benchmark problems 54
4.6 Results on the constraint-based benchmark problems 56
4.7 Scalability results on the constraint-based benchmark problems 58

List of Abbreviations

CSP	Constraint satisfaction problem.
GA	Genetic algorithm.
MDP	Markov decision process.
NFQ	Neural fitted Q-iteration.

1 Introduction

More than fifty percent of global consumers are more likely to buy from companies that allow them to shape products and services to their particular needs[1]. The world has been increasingly moving away form the mass-production business model introduced in the first half of the 20th century, and has entered the new frontier of mass customization [1]. This paradigm's goal is to combine the low costs of mass-production processes with the flexibility of individual customization [2]. Instead of choosing pre-built products from a catalog, customers are allowed to participate in the design process of the products they choose by specifying their desired characteristics. In this context, configuration systems have become leading technologies supporting the development of mass-customization business models.

1.1 Knowledge-Based Configuration

The literature has not shown much consensus on an academic definition of the process of configuration [3]. A widely-accepted informal definition given by Sabin and Weigel [4] describes configuration as "a special case of design activity, where the artifact being configured is assembled from instances of a fixed set of well-defined *component types* which can be composed conforming to a set of *constraints*". Two important elements can be identified from this definition:

i. the component types, which determine the aspects that can be configured (for example, a cellphone display may be in color or monochrome); and

ii. the constraints, which regulate the way in which the components interact with each other (for example, a color display may not work with Ni-Cd batteries).

[1] See Microsoft's white paper *Digital Trends 2015* at http://advertising.microsoft.com/en/digital-trends.

Both elements together constitute a knowledge-based configuration model that serves as the input to a configuration process. The outcome of this configuration process is an artifact instantiation or configuration that satisfies all the constraints (for example, a working cellphone with a color display and a Li-Ion battery, among other components).

This implicit modelling approach contrasts with an explicit enumeration of all the valid artifacts that can be assembled, and it makes knowledge-based configuration a major artificial intelligence application area. ‘Intelligent’ configuration systems, commonly known as configurators or mass-customization toolkits, use some representation of the configuration model and diverse reasoning techniques to provide assistance at both ends of the configuration activity—i.e., when the configuration system is being developed, and during the course of the configuration process. The assistance may involve, for example, the efficient maintenance of the configuration models, the development of well-informed interfaces to guide the users in the search for a configuration that meets their needs, and checking the consistency of the user requirements with the constraints of the configuration model. These technologies facilitate the configuration of complex products and services that otherwise could generate many erroneous or suboptimal configurations, and overwhelm users with confusion.

1.2 Precedent and Objectives

This thesis has an important precedent in the work of Hamidi et al. [5]. These authors introduced a framework effectively implementing a configurator for calculating the shortest sequence of questions to be presented to the users of a software system in order to customize it to their needs. Most modern software systems can be adapted to best fit the user’s individual skills and preferences through a number of configuration options. This characteristic offers great flexibility to the users, but as the amount and complexity of the options offered by the system grow, so does the space of configuration possibilities. As a result, the users have to deal with an increasingly overwhelming task when they attempt to configure the software. Some studies have shown that most people rarely change the default configuration options [6], which is understandable considering that they not only need to be familiar with the technical options, but also aware of the frequently unstated implications of each alternative [7].

The approach presented by Hamidi et al. [5] relies on information obtained from user preferences frequently used together by other (possibly expert) users to recommend default configuration options and avoid asking some of the questions explicitly. The proposed framework is based on two techniques: association rule mining [8] and Markov decision processes (MDPs) [9]. Association rule mining is used to analyze frequently co-occurring configuration options in order to predict a group of options based on the occurrence of another group—by means of an association rule discovered between them. The sequential process of asking the user a value for every configuration option is represented by an MDP, which provides a mathematical model for the uncertainty in the user's response. The MDP represents transitions from an initial configuration state in which all options are unknown to a terminal state where all the options are known, possibly skipping some questions in between because their answers can be predicted using the discovered association rules. By using dynamic programming methods to solve the MDP rewarding shorter paths towards the terminal state, the authors are able to calculate the shortest sequence of questions to be presented to the users. The solution of the MDP is expressed in terms of an optimal policy that dictates which question should be asked at each state, according to the knowledge acquired so far about the user's preferred configuration. This policy is optimal in the sense that it minimizes the expected number of questions to be asked to the users.

The applicability of this framework was evaluated in a two-part experimental study whose results were reported in detail by Hamidi [10]. The first part evaluated its use in a small case study according to the utility that it brings to the problem of configuring a software system. The case study employed configuration records from the privacy settings of the Facebook social network platform and showed that the technique reduced the number of questions asked to the users in this scenario. In the second part of the study, an artificial data set was used to evaluate the scalability of the approach in terms of the number of configuration options. This scalability study evidenced that the running time of the introduced technique grows exponentially in terms of the number of options. In fact, the approach in its original formulation seems feasible for handling only up to the equivalent of 14 binary configuration options within reasonable time. This situation constitutes a strong limitation for the use of the proposed framework in real-world scenarios with a considerable number of configuration options, where setting a large number of options automatically becomes increasingly important.

Drawing from these experiences, this thesis is tasked with the following objectives:

i. The first objective is to propose and evaluate empirically alternative methods for solving the formulated MDP to allow the scalability limitations of the framework introduced by Hamidi et al. [5] to be overcome.

ii. The second objective involves the generalization of the configuration model considered by Hamidi et al. [5] to enable using the framework in scenarios other than the recommendation of default options for the configuration of software systems. The assumption of a model based on association rules is rather restrictive and leaves out configuration processes that rely on constraints that cannot be expressed as simple rules. For greater flexibility, it is desirable to be able to specify the constraints that regulate the interactions of the configuration components as arbitrary Boolean expressions.

1.3 Outline of the Proposed Solutions

This thesis studies the problem of optimizing the user interaction in a broader class of configuration processes, not necessarily related to the configuration of software systems. An arbitrary artifact may be described in terms of two alternative configuration models. In the first model, the constraints that regulate the interactions between the aspects of the artifact that can be configured are given in the form of *if-then* rules comparable to the association rules used by Hamidi et al. [5]. In the second supported specification, the constraints are given as part of a formulation of the configuration model as a constraint satisfaction problem (CSP) [11].

As it was done by Hamidi et al. [5], the problem of minimizing the sequence of questions to be presented to the users is formulated as an MDP. However, a different approach is followed for finding the optimal policy. Hamidi et al. [5] use classical dynamic programming algorithms such as policy iteration [12] and value iteration [9], which require operating on a complete specification of the dynamics of the MDP. A configuration process formulated as an MDP will have an exponential number of states in terms of the number of components that can be configured, and this characteristic restricts the application of the dynamic programming algorithms to very small configuration models—an issue coined in the literature as the curse of dimensionality [9].

Reinforcement learning techniques [13] are employed in this thesis to overcome the aforementioned limitations. These solutions methods do not require a complete specification of the dynamics of the MDP and they rely instead on simulating interactions with the environment described by the MDP. This property makes reinforcement learning more suitable for dealing with problems with large state spaces. Two reinforcement learning solution methods are studied in this thesis. The first one constructs an artificial neural network that computes an estimate of the expected number of questions that could be skipped by asking one specific question. The second solution method solves the reinforcement learning problem as a stochastic optimization problem, relying on an evolutionary optimization algorithm to find a good approximation of the optimal solution.

1.4 Thesis Structure

The remainder of this thesis is organized as follows. Chapter 2 provides the necessary background for understanding the models and techniques that are employed, including several references for additional information. Specifically, the chapter covers the topics of configuration knowledge representation and the reinforcement learning approach for the solution of sequential decision-making problems. Chapter 3 presents the configuration models considered in the thesis as well as the proposed solution methods for the optimization of the user interaction in a configuration process. The results of a group of experiments performed to evaluate the introduced techniques are reported in Chapter 4. These experiments consider a diverse assortment of configuration models including artificially-generated configuration problems, a case study on a small real-world example, and a number of publicly-available configuration models from different domains. Finally, Chapter 5 summarizes the presented results, discusses how the current work compares to similar studies and gives an outlook of the future work.

2 Background

This chapter provides a description of the models and techniques that constitute the foundations of the solutions developed for the optimization of the user interaction in a configuration process. The presentation of the configuration topics follows Felfernig et al. [14], while the description of the reinforcement learning approach for the solution of sequential decision-making processes is mainly based on Sutton and Barto [13], and Wiering and van Otterlo [15].

2.1 Configuration Knowledge Representation

One important issue in configuration design is selecting a correct representation for expressing the characteristics of the component types and constraints, considering that there may exist a large number of possible configurations and/or complex constraint interactions. This section presents an overview of the two representations studied in the thesis: rule-based and constraint-based configuration models. In both cases, the study is restricted to component types with discrete domains—i.e., a selection among a finite set of possible choices.

2.1.1 Rule-Based Representation

The use of rule-based models dates back to the late 1970s with the introduction of the R1 system (later called XCON, for *eXpert Configure*) developed to support the customization of VAX computer orders [16]. Knowledge representation is based on *if-then* rules that encode legal configuration steps (in the more modern cellphone example, selecting Bluetooth wireless connection capabilities in a configuration may by implication mean that a Li-Ion battery is also chosen).

The rules operate on a working memory of configured component types that represent the stage in the configuration process. The memory is initially empty and each configuration decision triggers a revision of the rules. If a rule is activated (provided that the preconditions listed in the *if* part of the rule are satisfied), it modifies the content of the working memory by including all the assignments listed in the *then* part of the rule.

Although configuration systems with rule-based representations (such as the R1/XCON system) were very successful and are still used in practice, they present some major drawbacks. The application of the rules strongly depends on the order in which the component types are considered for configuration, and more importantly, this order may have a strong influence on the outcome of the configuration process. In the interest of achieving certain outcomes, heuristics had to be developed to ensure that the rules were activated in a particular order [17]. Furthermore, rule-based representations are often hard to maintain due to the lack of appropriate methods to check for hidden interactions between different rules.

Rule-based models are nonetheless well-suited for certain tasks within the configuration process, such as the recommendation of default values. As the number and complexity of the component types that can be specified with a configurator may be high, users are often overwhelmed by the large amount of configuration options [18]. Also, confronting users with too much information is known to lead to a decreased decision performance [19]. The recommendation of default configuration values can assist the users in the specification of their requirements and thus help to achieve higher user satisfaction.

The use of default configuration values supports the idea of the users specifying only those component types that are most important to them and letting the system automatically determine values for the remaining ones. The selection of the default configuration values can be based on previously-observed user requirements (for example, users who prefer a cellphone with a color display, generally select some form of wireless connectivity—such as Bluetooth). This approach becomes particularly relevant in a collaborative setting based on exploiting information from configuration sessions completed by other users to provide a personalized configuration experience for new users. Collaborative filtering, content-based filtering, and association rule mining are some of the technologies that support the construction of recommender systems (see Ricci et al. [20] for a comprehensive handbook on the topic).

2.1.2 Constraint-Based Representation

The shortcomings of the rule-based approach mentioned in the previous section motivated the development of representations supporting a clear separation between the domain knowledge and the problem-solving knowledge [21]. In a constraint-based knowledge representation, the information about when the constraints are checked is independent of the configuration model itself, and consequently, the order in which the constraints are enforced has no effect on the outcome of the configuration process. This thesis focuses on (static) CSPs as a prominent example of model-based knowledge representation in configuration.

2.1.2.1 Constraint Satisfaction Problems

The solution of a CSP involves finding values for a group of variables subject to limitations on which combination of values are allowed (see Tsang [11] for a rigorous book on CSPs, or Russell and Norvig [22] for a more didactic introduction). A CSP is defined by a set of variables $V = \{V_1, V_2, \ldots, V_n\}$, a nonempty domain D_i of the possible values for each variable $V_i \in V$, and a set of constraints $C = \{C_1, C_2, \ldots, C_m\}$. Each constraint in C involves some subset of the variables (the number of variables involved determines the cardinality of the constraint), and they specify the allowable combinations of the values for that subset, either extensionally (as a subset of the Cartesian product) or in terms of predicates (i.e., Boolean functions).

CSPs have been a widely-used formalism for describing configuration knowledge. In addition to providing a natural declarative representation for configuration models (with the component types as variables), their multidirectionality allows overcoming one of the central problems of rule-based approaches. While the use of a rule-based representation presupposes a direction from the *if* part to the *then* part, CSP solvers can propagate the consequences of the assignments to any other variable—and the order of the variable assignments does not affect the semantics of the constraints. Moreover, given an existing configuration (partially or completely specified), its correctness can be verified by checking the corresponding CSP for consistency.

The goal when solving a CSP is to assign a value to each variable, such that all the constraints are simultaneously satisfied. Constraint satisfaction is NP-complete in general, with polynomial-time algorithms existing to solve particular problem subclasses (such as acyclic constraint networks of binary constraints) [23]. The basic approach to solve a CSP is to apply backtracking search—i.e., a depth-first search that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign—combined with some form of constraint propagation to detect inconsistencies in advance and reduce the search branching factor. Constraint propagation is the general term for techniques that propagate the implications of a constraint on one variable onto other variables. Once an assignment is made to a variable, the domains of the unassigned variables can be pruned from values that are known to lead to inconsistent assignments by enforcing some level of consistency on the CSP.

Consistency can be established at different levels. The weakest form of (local) consistency is node consistency, which dictates that every unary constraint on a variable must be satisfied by all values in the domain of the variable. Node consistency can be enforced simply by restricting the domain of each variable to the values that satisfy all unary constraints on that variable—as a result, unary constraints are usually incorporated into the variable domains. A more powerful and commonly-used form of local consistency is generalized arc-consistency; which requires that for any constraint, the current assignment could be extended by some values for the unassigned variables in the constraint so that the constraint will be satisfied. There are several methods available for enforcing generalized arc-consistency derived from a group of algorithms for binary constraints (simply arc consistency) [24] and their generalizations to constraints involving more than two variables. There also exist other forms of domain-filtering local consistencies (see a review by Debruyne and Bessiere [25]), and each alternative comes with a trade-off between the computational cost of the algorithms that propagate the constraints and their pruning efficiency—since local consistency mechanisms does not necessarily remove all possible search dead-ends. The strongest form of consistency is global consistency, which implies that every value in the domain of the unassigned variables appears in at least one solution of the CSP. Therefore, it effectively guarantees that there exists at least one solution that satisfies all the constraints and it can be thought of as being equivalent to solving the CSP.

Constraint-propagation methods are essential for supporting configuration processes [26], given that they allow inferring information about unassigned variables from a group of previously-given user decisions. These inference mechanisms facilitate, for example, removing values that are incompatible with the current partial configuration from the domains of the unassigned variables, and also detecting inconsistent decisions potentially early in the configuration process. An inconsistent decision will cause a variable domain to become empty, therefore creating a dead-end in the configuration process.

2.1.3 Feature Models and Decision Models

Certain configuration application domains have prompted the development of specific models for configuration knowledge representation. One example of such domains is the software product lines paradigm for the efficient development of software-intensive systems sharing a common, managed group of features [27]. This section discusses how the semantics of some of these domain-specific representations can be transformed into the more general configuration models presented in the previous sections (CSPs in particular).

An increasing trend observed in software engineering is the need for building multiple, similar software systems instead of just individual unrelated products. Software product lines constitute a systematic framework for modelling and exploiting the common and variable aspects of the software systems in order to support deriving products tailored to the specific needs of different customers—which effectively results in a configuration process during the product-derivation phase. Feature models and decision models are among the most commonly-used approaches for modelling the commonality and variability in software product lines [28].

Feature models [30] are hierarchical tree-based structures represented graphically as the example in Figure 2.1 (additional extensions are possible [31, 32], but will not be covered in the thesis). Each node in the tree corresponds to a feature that can be selected as part of the product (i.e., an aspect or characteristic of the system, for example, the type of display on a cellphone), and a distinctive feature represented by the root node is assumed to be part of all the derived systems. The connections between a parent node and its children denote different types of relations: 'mandatory' (the children are required), 'optional' (the children may be selected or not), 'or' (at least one of the children must be selected), and 'exclusive or' (only one of the children can be selected). In addition to the connections supporting the tree structure, a group of cross-tree constraints are also allowed—the most common being 'implication' (the selection of one feature implies the selection of another), and 'mutual exclusion' (which indicates that two features cannot be part of the same product). The transformation of a feature model into an

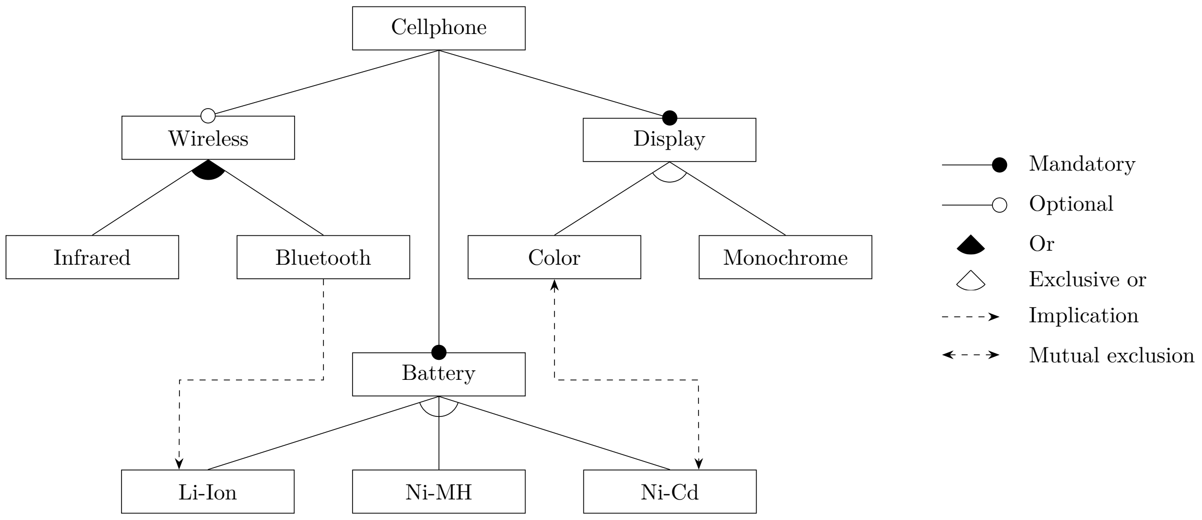

Figure 2.1: Example of a feature model for a cellphone (adapted from von der Maßen and Lichter [29]).

equivalent CSP involves representing each feature node as binary variable and translating the node connections into Boolean predicates with the same semantics as they have in the tree (for a detailed explanation, see Benavides et al. [33]).

Decision models, on the other hand, present a more diverse structure (see Schmid et al. [34] for a comparison of representative modelling approaches). Probably the most distinctive characteristic of these models is that they focus on supporting product derivation as opposed to describing the problem domain. As a consequence, decision models represent only the decisions that must be made to derive a specific product from the product line. The decisions are generally represented in the form of questions with a defined set of possible answers (but not necessarily binary as in feature models). The hierarchy among the questions is secondary in decision models, but they do support defining relations between the questions similarly to the cross-tree constraints in feature models. In spite of having a more diverse structure, the semantics of the decision models can be traced back to a CSP with the questions as variables and constraints representing the relations between those questions.

2.2 Sequential Decision Making Under Uncertainty

In order to optimize the interaction of a user in a configuration process, it is necessary to rely on a model able to properly represent the users' sequential decisions and in particular the uncertainty in their response. This section presents the fundamental formalism for reasoning about sequential decision-making problems and a group of techniques that can be used to design optimal plans to act in uncertain environments—such as a configuration process.

2.2.1 Markov Decision Processes

MDPs constitute an integral formalism for modelling sequential decision-making problems (see Puterman [35] for an extensive book on MDPs, or Russell and Norvig [36] for a more gentle introduction). In these models, the environment is represented by a set of states along with a set of actions that can be performed to potentially modify the current state. A defining characteristic of the environment is that it has a stochastic nature, i.e., there is a certain level of uncertainty about the effect of the actions. The goal in this context is to design a strategy to operate on the environment such that some reward criterion is maximized.

More formally, a (finite, fully-observable) MDP is defined by the following components:

i. A finite set of states S. Each state $s \in S$ characterizes the environment of the problem being modelled at a particular moment in time.

ii. A finite set of actions A. When an action $a \in A$ is performed while being at a state $s \in S$, the environment responds by transitioning to a (potentially different) state $s' \in S$.

iii. A transition function $T(s, a, s')$ that specifies the probability of reaching state $s' \in S$ from state $s \in S$ by performing action $a \in A$. By this definition, it is assumed that the result of an action depends only on the current state and not on the previous actions and visited states. The transition function is generally stored in a table with one entry for every combination of s, a, and s'.

iv. A reward function $R(s, a, s')$ that returns a scalar value corresponding to the immediate reward received from leaving state $s \in S$ by taking action $a \in A$ and reaching state $s' \in S$. As with the transition function, the reward function is usually stored in tabular form.

Among the different variations of MDPs that can be modelled by this definition, this thesis focuses on a particular group called episodic MDPs. In these processes, there is a notion of episodic tasks in which the intention is to transition from an initial state to a certain terminal state, while progressing through a finite sequence of intermediate states based on the actions performed. As each task ends when the terminal state is reached, all transitions from the terminal state move to that same state with probability one (so, for a terminal state $s \in S$ in an episodic MDP, it holds that $T(s, a, s) = 1$ for all actions $a \in A$). The rewards at the terminal state are also adjusted accordingly.

When an MDP is solved, the goal is to find a function $\pi : S \to A$ that returns an action $a \in A$ for each state $s \in S$. This function is called a (deterministic) policy and it dictates the action that must be taken at each state to operate successfully on the environment. The execution of a policy is done in the following way. First, the environment begins at an initial state s_0. Then, the policy π dictates the action $a_0 = \pi(s_0)$ and this action is performed. Based on the transition and the reward functions T and R, a transition is made to a state s_1 with probability $T(s_0, a_0, s_1)$ and a reward $R(s_0, a_0, s_1)$ is received. This process continues producing an environment history $s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, \ldots, s_n$; where s_n is a terminal state.

Each time a given policy is executed starting from an initial state, the stochastic nature of the environment will lead to a different sequence of transitions to the terminal state. The quality of a policy is therefore measured by the expected reward of the possible environment histories generated by that policy. Consequently, an optimal policy (denoted by π^*) is a policy that yields the highest expected reward. Although there are different ways of combining the immediate rewards, this thesis concentrates on an additive rewards model in which all the immediate rewards are summed together. The upcoming sections describe a group of algorithms designed to find optimal policies for problems formulated as MDPs.

2.2.2 Dynamic Programming and Value Iteration

Dynamic programming refers to a class of algorithms that can be used to compute optimal policies given complete information about the environment represented as an MDP. The key idea is to use so-called value functions to organize the search for good policies. A value function quantifies the importance of the different states and this information can be used to select the best action to be performed at a given state by considering the possible transitions to the other states.

Value functions are defined in terms of a policy, given that they depend on a sequence of transitions. The value of a state s under a policy π, denoted as $V^{\pi}(s)$, is the expected reward when starting in s and following π thereafter:

$$V^{\pi}(s) = \mathrm{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid \pi, s_0 = s\right]. \tag{2.1}$$

In the formula, $\gamma \in [0, 1]$ denotes a discount factor that diminishes the importance of future rewards and it guarantees that the rewards sum is finite even for infinite transition sequences. In the case of episodic MDPs γ can be set to 1, given that the tasks will always reach the terminal state in a finite number of transitions.

A fundamental characteristic of the value functions is that they satisfy a recursive property known as the Bellman equations [9]. In particular, given that the goal when solving an MDP is to find the policy with the highest reward, it can be proven that the value function V^{π^*} for an optimal policy π^* can be written as follows:

$$V^{\pi^*}(s) = \max_{a \in A} \sum_{s' \in S} T(s, a, s') \left(R(s, a, s') + \gamma V^{\pi^*}(s')\right), \tag{2.2}$$

which means that the value of a state under the optimal policy must be equal to the expected reward from selecting the best action in that state and continuing to act optimally in the future. Consequently, given the optimal value function V^{π^*}, it is possible to compute a corresponding optimal policy by greedily selecting the action to be performed at each state s:

$$\pi^*(s) = \operatorname*{argmax}_{a \in A} \sum_{s' \in S} T(s, a, s') \left(R(s, a, s') + \gamma V^{\pi^*}(s')\right). \tag{2.3}$$

The Bellman equations are the basis of the value iteration algorithm for solving MDPs [9]. Its general idea is turning Equation (2.2) into an update rule for improving successive approximations of the optimal value function. The algorithm begins with an arbitrary initial approximation V_0 (e.g., with all entries set to zero), and iteratively updates the value of each state s according to the following assignment:

$$V_{k+1}(s) = \max_{a \in A} \sum_{s' \in S} T(s, a, s') \left(R(s, a, s') + \gamma V_k(s')\right), \tag{2.4}$$

which produces a sequence that converges in the limit to the optimal value function. A practical stopping condition is to measure the quantity $\max_{s \in S} |V_{k+1}(s) - V_k(s)|$ after each iteration and

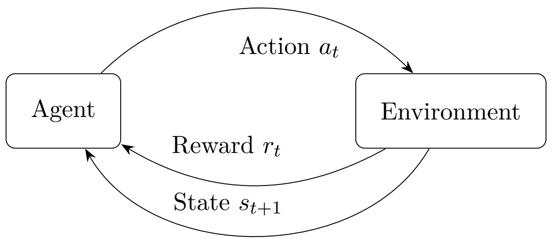

Figure 2.2: The reinforcement learning scenario (adapted from Sutton and Barto [13]).

stop when it becomes sufficiently small. Once a good approximation is computed, a corresponding optimal policy can be derived by means of Equation 2.3.

A major drawback of value iteration, and dynamic programming methods in general, is that they involve repeated operations over all the states of the MDP. In problems in which the state space is considerably large, even a single operation over the entire set S can be prohibitively expensive. Besides, the complete definition of the functions T and R may be difficult if an analytic representation of the environment cannot be fully-computed in advance. As an alternative, the algorithms presented in the remaining two sections do not require a complete specification of the MDP, they rely instead on simulating interactions with the environment.

2.2.3 Value-Based Reinforcement Learning

Reinforcement learning [13] is a machine learning and artificial intelligence paradigm that deals with learning in sequential decision-making problems in which there is limited feedback. The class of problems that can be solved with reinforcement learning techniques are often the same that can be represented as MDPs, but contrary to the dynamic programming methods, reinforcement learning algorithms do not require a complete specification of the environment (i.e., they are model-free). This characteristic makes reinforcement learning a more suitable solution technique for problems with large state spaces or incomplete information.

Figure 2.2 illustrates the interaction between an agent and the environment in a reinforcement learning scenario. At a given time step t, the environment is assumed to be in a state s_t and the agent performs the action a_t. Then, the environment responds by transitioning to an updated state s_{t+1} and it returns this information to the agent along with an immediate reward r_t for the action a_t. In an episodic task, this interaction loop is repeated until the terminal state is reached. The current state and the reward information constitute the only perceptions that the agent gets from the environment. The goal of the agent is to learn the actions that it must perform to maximize the long-term expected reward.

Given that there is limited information about the environment, the optimal policy must be learned by trial-and-error. The agent has to explore the environment by performing actions and perceiving their consequences in order to learn the best action to be performed at each state. This learning approach comes with certain challenges. At some point in the interaction loop, the agent faces a decision between reinforcing the knowledge it has already acquired about what it believes to be the best action in a given state, or trying out other less explored actions that could lead to higher rewards. A simple and popular way of balancing this exploration and exploitation trade-off is to use the so-called ϵ-greedy exploration criterion, in which the agent chooses the best action with probability $1 - \epsilon$, and a new action uniformly at random with probability ϵ. The value $\epsilon \in [0, 1]$ is a parameter that weighs the relative importance of exploration and exploitation.

As in dynamic programming, many reinforcement learning algorithms proceed to solve the problem by estimating a value function—although an alternative representation is used in this scenario. Instead of learning an optimal state-value function such as Equation 2.2, the targeted function is an analogous optimal action-value function defined as:

$$Q^{\pi^*}(s,a) = \sum_{s' \in S} T(s,a,s') \left(R(s,a,s') + \gamma \max_{a' \in A} Q^{\pi^*}(s',a') \right), \tag{2.5}$$

which returns the expected cumulative reward from performing an action a in the state s and continuing to act optimally according to the policy π^* thereafter. Given an estimation of $Q^{\pi^*}(s,a)$, a corresponding optimal policy π^* can be calculated similarly to Equation 2.3:

$$\pi^*(s) = \operatorname*{argmax}_{a \in A} Q^{\pi^*}(s,a). \tag{2.6}$$

However, in this case the definition of the transition function T is not necessary, given that all the required information is obtained through the Q function. The remaining of this section describes two different algorithmic approaches for the estimation of optimal action-value functions.

2.2.3.1 Q-learning

One of the most popular model-free methods to estimate an optimal action-value function for a reinforcement learning problem is the Q-learning algorithm [37]. The basic idea of Q-learning is to use a similar update rule as in value iteration to adjust the estimated value of state-action pairs based on the immediate rewards and the current estimated value of the best future actions:

$$Q_{k+1}(s_t, a_t) = (1-\alpha)Q_k(s_t, a_t) + \alpha \left(r_t + \gamma \max_{a' \in A} Q_k(s_{t+1}, a') \right). \tag{2.7}$$

The parameter $\alpha \in [0,1]$ is a learning rate that balances the the influence of the previous estimation and the newly-acquired information.

In episodic tasks, the update rule can be executed either online, at the time each reward is received during the interaction loop, or offline at the end of an episode when a complete environment history has been collected (the latter is the approach followed in this thesis). Q-learning will converge to the optimal policy under the assumption that each state-action pair is visited an infinite number of times (hence the need for a separate exploration mechanism) and that the learning parameter α is sufficiently small [38]. In practice, the algorithm is generally run for a fixed number of simulated episodes.

Although Q-learning does not require a complete representation of the environment to learn an optimal policy, in its basic form it is still unsuitable for solving MDPs with a large number of states. The problem is that, as in value iteration, it is assumed that the estimates of the value function are stored in a look-up table with one entry for each state-action pair. This assumption causes problems, not only with the amount of memory necessary to store large tables, but also with the number of simulated episodes needed to fill them accurately. This situation motivates the use of function approximation techniques to build more compact representations of the Q function, which are both smaller in size than the tabular representation and that may potentially generalize information from one state onto others. The idea of using a function approximation technique is not to store the $Q(s,a)$ values for every state-action pair separately, but to store them as a function of the state and action. The next section presents one way of achieving this goal through the use of artificial neural networks.

2.2.3.2 Neural Fitted Q-iteration

Neural fitted Q-iteration (NFQ) [39], which in turn is a realization of the fitted Q-iteration model [40], is a reinforcement learning method that can be used to efficiently learn an action-value function represented by an artificial neural network. This algorithm follows an alternative learning process for estimating the Q function. Instead of discarding each transition after updating the Q function as it is done in the basic Q-learning, all previous transitions are stored and this information is reused every time the Q function is updated. The consideration of the entire transition history instead of a single online sample enables the application of advanced supervised learning methods to train the Q function, hence requiring fewer episodes to obtain sufficiently good approximations.

Specifically, NFQ operates in a so-called growing-batch learning process (see Lange et al. [41] for a review), which consists of two phases repeated one after the other. The first phase is an exploration phase where interactions with the environment are simulated following some exploration policy. During this phase, the Q function remains unchanged and it is only used to guide the interaction with the environment. After enough transitions are collected, the interaction with the environment stops and a learning phase begins in which the Q function is updated considering all the previous history of transitions. These two phases are repeated iteratively, thus incrementally growing the batch of stored transitions using intermediate policies.

Figure 2.3 shows the main loop of the NFQ algorithm, which runs during the learning phase of the growing-batch learning process. It consists of two major steps: the generation of the set of training patterns P and the training of a neural network with those patterns. Each pattern in P corresponds to an input to the neural network and its corresponding target output generated from a transition $\langle s, a, r, s' \rangle$ in the data set D. The input is the state-action pair $\langle s, a \rangle$ and the target output is computed using the formula:

$$r + \gamma \max_{a' \in A} Q_k(s', a'), \tag{2.8}$$

which is essentially the Q-learning update rule from Equation 2.7 except that it does not include the learning rate parameter α. The use of a learning rate is not necessary in this case, given that all previous transitions are remembered. The RPROP algorithm [42] is used for the supervised training of the neural network. The Q function is represented using a multilayer perceptron,

```
Input: A data set $D$ of transition samples $\langle s, a, r, s' \rangle$.
Output: An approximation of the $Q$ action-value function
represented by a neural network.

1: Initialize $Q_0$ arbitrarily.
2: $k \leftarrow 0$
3: repeat
4:     Generate a set of training patterns $P$ from $D$ and $Q_k$.
5:     Use RPROP and $P$ to train a neural network representation of $Q_{k+1}$.
6:     $k \leftarrow k + 1$
7: until a maximum number of iterations is reached.
8: return the action-value function $Q_k$.
```

Figure 2.3: Main loop of neural fitted Q-iteration (adapted from Riedmiller [39]).

which is a type of feedforward artificial neural network. Section 3.3.2 gives additional information about the particular neural network structure used in this thesis.

2.2.4 Evolutionary Algorithms for Reinforcement Learning

An alternative method to find a good policy is to search directly in the policy space, in which case the reinforcement learning problem can be expressed as a stochastic optimization problem. Under this formulation, the goal is to find a policy π^* that maximizes an objective function f among all possible policies $\pi \in \Pi$. Given the stochastic nature of the environment, $f(\pi)$ can be thought as a random variable, and the optimal policy becomes:

$$\pi^* = \operatorname*{argmax}_{\pi \in \Pi} \mathrm{E}\left[f(\pi)\right], \tag{2.9}$$

i.e., the policy that maximizes the expected value of the objective function. In an episodic MDP, the objective function may be defined as the expected cumulative reward obtained in an episode executed following the policy, and it can be approximated by the average cumulative reward obtained in a number of simulated episodes. This thesis focuses on evolutionary algorithms among the different optimization methods available to solve this problem (see Moriarty et al. [43] and Whiteson [44] for surveys on the use of evolutionary optimization for reinforcement learning).

Genetic algorithms (GA) [45, 46] are one of the earliest and most representative examples of evolutionary computation. These algorithms approach the problem of finding the global optimum

```
Input: A fitness function $f$ and a group of control parameters—e.g.,
population size ($N$), crossover probability ($p_c$), mutation probability ($p_m$).
Output: A solution that maximizes the fitness function $f$.

 1: Randomly generate an initial population $P_0$ with $N$ solutions.
 2: $g \leftarrow 0$
 3: repeat
 4:     Calculate the fitness of the solutions in $P_g$ according to $f$.
 5:     Initialize a new empty population $P_{g+1}$.
 6:     Copy the best solution from $P_g$ into $P_{g+1}$ (elitism).
 7:     repeat
 8:         Choose parent solutions according to a selection operator.
 9:         With probability $p_c$, apply the crossover operator.
10:         With probability $p_m$, apply the mutation operator.
11:         Add the resulting solutions to the new population $P_{g+1}$.
12:     until the new population $P_{g+1}$ contains $N$ solutions.
13:     $g \leftarrow g + 1$
14: until a termination criterion is satisfied.
15: return the solution with the maximum fitness in $P_g$.
```

Figure 2.4: Pseudocode of a genetic algorithm.

of a function by following a procedure inspired by the process of natural selection—i.e., survival of the fittest. GAs iteratively improve a population of solutions, by creating new solutions that share characteristics of the best solutions in the population. The general idea is that structures associated with good solutions can be combined to form even better solutions.

Figure 2.4 shows the pseudocode of a GA as used in this thesis. The algorithm starts by generating an initial population of solutions and evaluating each generated solution in the objective function (i.e., computing their fitness). Then, a group of parent solutions are selected from the population by using a selection operator. Various selection operators can be used, but all share the basic idea of promoting the selection of better solutions with higher probability. Once the parent solutions have been selected, new candidate solutions are created by applying crossover and mutation operators. The crossover operator combines subsets of the parent solutions by exchanging some of their parts, while mutation slightly perturbs the recombined solutions to introduce variation in the population. The crossover and mutation steps can be skipped with certain probabilities, which offers the possibility of copying unmodified parents or offspring that did not suffer mutation into the new population. Each newly created solution becomes part of a new population and this completes an iteration (referred to as a generation). The best-performing solution is always kept across generations, as part of a mechanism known as elitism that ensures that it is never lost because of the effect of the selection operator. After a generation is completed, the new population replaces the current population and the next iteration is executed (starting with the evaluation), unless a termination criterion is satisfied.

Direct policy-search methods maintain explicit representations of the policies and consider only their overall performance, without computing value estimates for each state-action pair. The selection against poor individual actions is implicitly made by selecting against poorly-performing policies. As a consequence, evolutionary computation methods for reinforcement learning sometimes require more interactions with the environment than value-based methods to find a good policy—especially in highly stochastic environments in which many simulated episodes may be necessary to obtain reliable estimations of the expected cumulative reward of the policies [47]. However, since policies need only to specify an action for each state instead of the value of each state-action pair, direct policy-search methods facilitate the design of representations tailored to the problem at hand—especially with the use of evolutionary optimization techniques, such as GAs, that give great flexibility for choosing a solution representation.

2.3 Summary

This chapter presented an overview of the models and techniques supporting the solution methods developed in this thesis. The first part introduced two models for configuration knowledge representation: rule-based and constraint-based models. The latter model based on CSPs offers greater flexibility given that it allows specifying the constraints as arbitrary Boolean expressions and the order of the variable assignments does not affect the model semantics. In the second part of the chapter, the MDP formalism for sequential decision-making problems and a group of techniques for solving these problems were described. The techniques based on the reinforcement learning paradigm (in particular, NFQ and the GA-based approach) are the most relevant to this thesis, given that they facilitate solving MDPs with a large number of states. In the next chapter, those two MDP solution methods and the described configuration models are combined into a framework for calculating optimal configuration processes efficiently.

3 Proposed Solutions

This chapter introduces the proposed techniques to calculate optimal configuration processes efficiently. The first section begins by defining the configuration models studied in this thesis, along with a configuration process and some necessary terminology. In the remainder of the chapter, the problem of optimizing the user interaction is formulated as an MDP, and two solution methods are presented.

3.1 Configuration Models

As considered in this thesis, a *configuration model* for an arbitrary artifact consists of a tuple with three elements. The first and second elements are a set of variables defining the aspects that can be configured and their possible choices, respectively. The third element is a set of restrictions that determines how the different variables interact with each other to form a valid instantiation of the artifact. The term artifact refers to anything that can be subject to configuration, for example, an object such as a cellphone or a software system.

Each configuration variable V_i, from the set $V = \{V_1, V_2, \ldots, V_n\}$ that constitutes the first element of the configuration model, has an associated nonempty discrete set D_i containing the $|D_i|$ possible values in the domain of the variable. Moreover, a *configuration* is defined as a set of assignments of specific values to some or all of the variables in V, where each individual assignment formed by a variable and its corresponding value is called a *configuration decision*. A configuration without any decision is said to be *empty*, and a configuration is considered to be *complete* if it contains a decision for each one of the n variables in the model. Consequently, a *partial* configuration has at least one unassigned variable.

The restrictions, on the other hand, can be specified in one of two alternative forms—resulting in either a *rule-based* or a *constraint-based* configuration model:

i. In rule-based models, the restrictions are given as a set R of *if-then* rules of the form $X \to Y$, where both the *if* part (i.e., X) and the *then* part (i.e., Y) are sets of variable assignments. The semantics of each rule are such that, if the decisions in a partial configuration satisfy the assignments in the *if* part of the rule (i.e., each variable has assigned the same value), then the rule is applied by extending the configuration to include the decisions dictated by the *then* part of the rule. It is assumed that all the rules have an associated priority that allows resolving any possible inconsistencies among the dictated configuration decisions. This specification corresponds to an implementation of the rule-based configuration representation discussed in Section 2.1.1.

ii. Constraint-based configuration models offer a more flexible representation. In this case, restrictions are given as a set C of constraints in the form of arbitrary Boolean predicates defined for subsets of the configuration variables in V. As it was explained in Section 2.1.2, this formulation of the configuration model corresponds to a CSP. Notice that constraint-based models generalize rule-based models, since any rule of the form $X \to Y$ can be transformed to the logically equivalent predicate $\neg X \vee Y$. Given as a predicate in a CSP, this representation has the added benefit of inference being performed from Y onto X, in addition to from X onto Y as it is only done with the rule-based representation.

A partial or complete configuration is said to be *consistent* if it satisfies all the restrictions in the configuration model, particularly in the case of constraint-based models. Therefore, the goal of the configuration process is to generate complete and consistent sets of variable assignments, that correspond to correct configurations of the given artifact. The process of configuring an artifact consists of starting with an empty configuration and making sequential configuration decisions about variables that have not yet been decided upon, until a complete configuration is obtained.

Figures 3.1 and 3.2 illustrate the configuration processes for rule-based and constraint-based configuration models, respectively. Both procedures follow the same general structure: they start with an empty set of variable assignments and make a configuration decision about an unassigned variable at each iteration of the main configuration loop. After the main loop ends, the set of assignments A contains a complete configuration of the artifact described by the input model.

```
Input: A rule-based configuration model $\langle V, D, R \rangle$.
Output: A complete configuration of the variables in $V$.

1: Initialize an empty set of assignments $A \leftarrow \{\}$.
2: $q \leftarrow 0$
3: repeat
4:     Choose an unassigned variable $V_i$ from $V$.
5:     Select a value $v_i \in D_i$ for $V_i$.
6:     $A \leftarrow A \cup \{V_i = v_i\}$
7:     repeat
8:         for each $X \rightarrow Y \in R$ do
9:             if $A$ satisfies $X$ then
10:                $A \leftarrow A \cup Y$
11:            end if
12:        end for
13:    until no new assignments are added to $A$.
14:    $q \leftarrow q + 1$
15: until $A$ is complete.
16: return the complete configuration $A$.
```

Figure 3.1: Configuration process with a rule-based configuration model.

```
Input: A constraint-based configuration model $\langle V, D, C \rangle$.
Output: A complete configuration of the variables in $V$.

1: Initialize an empty set of assignments $A \leftarrow \{\}$.
2: $q \leftarrow 0$
3: repeat
4:     Choose an unassigned variable $V_i$ from $V$.
5:     Select a value $v_i \in D_i$ for $V_i$.
6:     $D_i \leftarrow \{v_i\}$
7:     Enforce consistency in the CSP $\langle V, D, C \rangle$.
8:     for $j = 1, \ldots, |V|$ do
9:         if $V_j$ is unassigned and $|D_j| == 1$ then
10:            $A \leftarrow A \cup \{V_j = v_j \in D_j\}$
11:        end if
12:    end for
13:    $q \leftarrow q + 1$
14: until $A$ is complete.
15: return the complete configuration $A$.
```

Figure 3.2: Configuration process with a constraint-based configuration model.

The procedures differ in the way they propagate the implications of each configuration decision (i.e., lines 6–13 in Figure 3.1 versus lines 6–12 in Figure 3.2). In rule-based models (Figure 3.1), each rule is considered for application after a configuration decision has been added to the set of assignments. Notice that this must be done also for decisions dictated by the *then* part of a rule, and thus a loop is introduced in line 7 to revise the rules again in that case. In constraint-based models (Figure 3.2), a configuration decision on a variable essentially reduces the domain of the variable to a single value for the rest of the configuration process. A certain level of consistency is then enforced in the corresponding CSP to propagate the effect of each decision onto the domains of the unassigned variables. If as the result of the constraint-propagation mechanism, the domain of an unassigned variable is reduced to a single value, then the corresponding decision is added back to the set of assignments in line 10. This situation means that the value of an unassigned variable was automatically inferred from the set of currently assigned variables. Ideally, global consistency should be enforced after every configuration decision, but generally only a weaker form of consistency can be used in practice due to the high computational cost of the algorithms that propagate the implications of the assignments.

In an interactive configuration process (i.e., with a 'human in the loop'), the step in line 5 of both Figure 3.1 and 3.2 is generally implemented by asking the users a question about their desired value for the chosen configuration variable. In both procedures, the variable q counts the number of questions that were asked to obtain a complete configuration of the artifact. This value is an indication of the user's involvement in the configuration process, and it is precisely the target for minimizing the user interaction in the study carried out in this thesis. In the next section, this optimization problem is formulated as an MDP.

3.2 Markov Decision Process Formulation

As mentioned in the previous section, the aim pursued for optimizing the user interaction in a configuration process is to minimize the number of questions being asked to the user (i.e., the final value of the variable q in Figures 3.1 and 3.2). The rule-based and constraint-based models considered in this thesis offer mechanisms that allow inferring the values of unassigned variables from a group of previously-assigned variables. In rule-based models, inference is performed by expanding the *then* part of a rule, while constraint-based models rely on constraint-propagation algorithms. These mechanisms can be used to avoid asking questions to the user about variables that could potentially be answered automatically from other user's decisions.

This optimization problem can be formulated as a sequential decision-making task, where it must be decided which question to ask to the user at each step of the configuration process (i.e., choosing an unassigned variable in line 4 of Figures 3.1 and 3.2). Clearly, the sequential decisions depend on the user's responses, and thus it is not generally possible to establish a particular order for asking the questions that minimizes all possible user interactions. Therefore, solution methods for this problem must act by optimizing the expected performance criterion considering the uncertainty in the user's responses. MDPs offer a well-suited formalism for describing this task in terms of a set of states S, a set of actions A, and transition and reward functions T and R, respectively (see Section 2.2.1).

An episodic MDP can be used to model the task of optimizing the user interaction in a configuration process. The states in the set S capture the knowledge that has been acquired so far about the user's configuration decisions (represented as a set of variable assignments), while each action in A denotes asking a question to the user about a configuration variable. The initial state in the MDP corresponds to an empty configuration, given that no decision has been made at the time. A single terminal state is used to represent the situation in which a complete configuration of the artifact is known. It is not necessary to have separate states for every possible complete configuration, given that no questions are asked after the configuration is complete. However, there is one state for every possible partial configuration of the variables, where the intermediate decisions of which question to ask next take place.

When a question about an unassigned variable is asked in a given state, an answer is received back from the user and it causes a transition to another state in which the value of a greater number of configuration variables is known. Given that each user's configuration decision (and consequently, the resulting state) is not known beforehand, the uncertainty in the user's responses is modelled in the MDP as a transition function $T(s, a, s')$ that gives the probability of leaving the state s by performing an action a and moving to a state s'. Specifically, each entry in the transition function of the MDP is determined by the conditional probability of receiving a particular answer from the user, given his or her current configuration decisions. This probability distribution gives great flexibility for modelling the users' preferences in the configuration process. If no prior information is available about the users' intentions, the probability distribution may be defined as returning the same value for every possible answer. However, in situations where configuration records from previous users are available, this information can be used to compute the conditional probabilities from the data, reflecting the actual interests of the users configuring the artifact.

The destination state s' in a transition is the result of both the configuration decision made by the user and the application of the inference mechanisms of the configuration model. Due to the effect of the expansion of the *then* part of the rules in rule-based models and the use of constraint-propagation in constraint-based models, more than one variable may be assigned after a transition is completed. Given that the goal is to minimize the user interaction in the configuration process, the rewards in the MDP are defined in terms of the number of variables that are automatically discovered in each transition. By solving the formulated MDP towards actions (i.e., questions asked to the users) that maximize the expected reward (i.e., number of configuration variables automatically discovered), the expected user interaction is minimized. The user will not need to provide a decision for the configuration variables that were automatically discovered.

Each entry of the the reward function $R(s, a, s')$ is thus given by the difference $|s'| - |s| - 1$, where $|s|$ and $|s'|$ denote the number of variable assignments in the states s and s', respectively. For transitions in which only the user-given configuration decision becomes known, the immediate reward will be zero. Moreover, all transitions that go from a state s to the same state s will be assigned a reward $R(s, s) = -1$ (except in the terminal state, where the usual convention for episodic MDPs of having zero rewards for all transitions is followed in order to ensure that the cumulative rewards sum is finite). Although it is assumed by this definition of $R(s, a, s')$ that every question asked to the users comes with a constant cost of one unit of reward, more elaborated reward models could be employed. For example, questions associated with variables that demand a considerable effort from the user in order to come up with a configuration decision (because, e.g., they involve a large number of possible answers) might be penalized in the $R(s, a, s')$ function with the goal of promoting policies that try to avoid overwhelming the users with difficult decisions.

The solution of the MDP is expressed in terms of a policy that dictates which question should be asked at each state of the configuration process, considering the knowledge acquired so far about the user's preferred configuration. As the result of the rewards being defined in terms of the number of automatically-discovered variables, this policy is optimal in the sense that it minimizes the expected number of questions to be asked to the users in order to obtain a fully-specified configuration. However, given that the policy is defined for every possible configuration state, the user still has complete control over the order in which he or she answers the questions if inclined to do so. At any point of the configuration process, the user can safely ignore the suggested question and answer a different one, continuing later with an optimized sequence of questions for the remainder of the configuration process. This additional flexibility can help to deal with situations in which the sequence of questions proposed by the optimized policy results unnatural or not logically evident to the user.

3.3 Solution Methods

The number of states in the formulated MDP grows exponentially with the number of configuration variables. As an illustrative example, the MDP corresponding to a configuration problem with n binary variables would have $(2+1)^n - 2^n + 1$ states in total. In this formula, one is added to the possible values of the variables to account for the states in which the variables have not yet been assigned, and the terminal state is counted only once for all the 2^n possible complete configurations. As it will be evidenced in Section 4.2.1, an MDP with this many states imposes serious scalability limitations for solution methods that keep track of pieces of information regarding every individual state. On the one hand, the computation of a complete definition of the functions T and R becomes a prohibitively expensive task, given that it is necessary to compute and evaluate all possible transitions beforehand. On the other hand, even if model-free techniques are used, it becomes rapidly unmanageable to store any explicit information for every possible state in the MDP.

In order to solve the optimization problem associated with minimizing the user interaction in a configuration process efficiently, it is necessary to employ methods that simulate interactions with the environment described by the MDP, and are able to learn effectively from a limited number of experienced transitions compared to the total number of states in the MDP. The study presented in this thesis focuses on the use of the NFQ algorithm and a GA to solve the resulting reinforcement learning problem (see Sections 2.2.3.2 and 2.2.4, respectively). The remainder of this section discusses how to adapt these techniques for the problem in question. Given that both algorithms rely on an existing method for simulating interactions with the environment, the first part of the section briefly describes the simulation of a configuration process.

3.3.1 Simulation of a Configuration Episode

Figures 3.1 and 3.2 outline the processes of instantiating an artifact described in terms of the configuration models studied in this thesis. The configuration processes guide the user starting with an empty assignment of the variables and ending up with a complete configuration of the artifact, while transitioning through a number of intermediate partial configurations. The resulting set of transitions corresponds precisely to an episode in the MDP formulated for the optimization of the user interaction in the process. The initial state corresponds to the empty configuration at the beginning, and the terminal state to a complete configuration being obtained in the end. Given that the only required user interaction is to provide an answer to the question asked in line 5, a configuration episode can be easily simulated by sampling the user's responses to the questions. The answers can be generated according to the individual conditional probabilities of the user's responses, given his or her current configuration decisions as discussed in Section 3.2. Each transition in the simulated configuration process corresponds to an experienced transition $\langle s, a, r, s' \rangle$ in the MDP. These transitions serve as the input to the NFQ algorithm and the GA for collecting information about the environment and deciding on an optimal policy.

3.3.2 Neural Fitted Q-iteration

In NFQ, all the generated transitions $\langle s, a, r, s' \rangle$ resulting from the simulated configuration episodes are stored in a data set. This data set is used throughout the execution of the algorithm to train an artificial neural network approximation of the action-value function $Q(s, a)$ (see Section 2.2.3.2 for details). The function $Q(s, a)$ returns the expected cumulative reward (i.e., the total number of automatically-discovered answers) that can be obtained from asking the question corresponding to the action a in the configuration state s. This information can be used to derive an optimal policy for minimizing the user interaction in the configuration process. The only aspect of the NFQ algorithm that needs to be adapted to the problem at hand is the structure of the artificial neural network representing the Q action-value function. A feedforward neural network with a single hidden layer is used in this thesis. The general inner workings of this model as a function approximation technique are presented first, before explaining the structure of the specific network used to represent the action-value function for the formulated MDP.

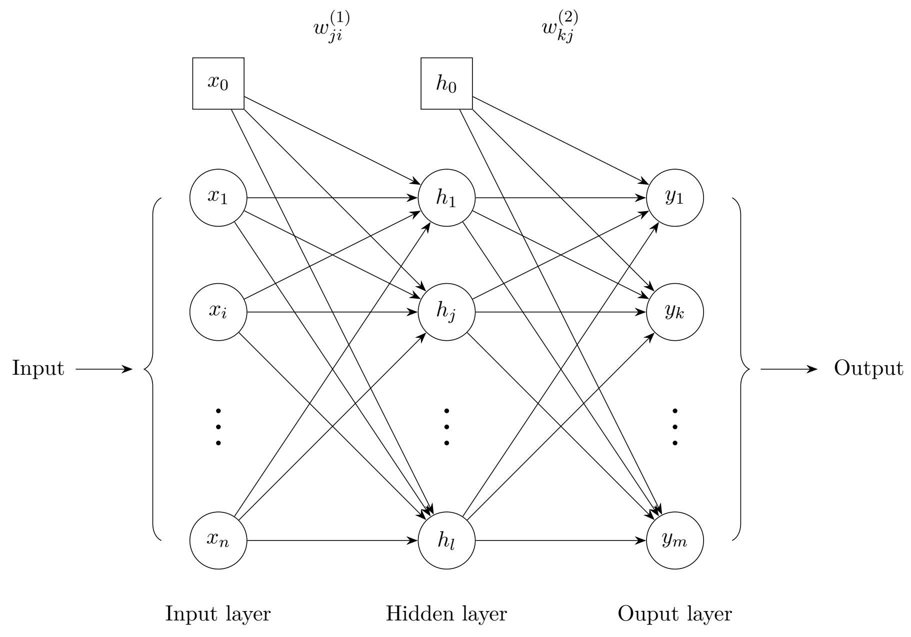

Figure 3.3: Feedforward neural network with a single hidden layer (adapted from van Eck and van Wezel [49]).

3.3.2.1 Feedforward Neural Network with a Single Hidden Layer

Artificial neural networks are function approximation techniques inspired by the way biological neural networks work in the brain (see Bishop [48] for a good introduction to the topic). They are generally represented as a system of interconnected units (also called neurons) which pass information to each other. Although their structure may vary, the neurons are usually arranged together forming a number of network layers. One of the layers is distinguished as receiving the input of the function represented by the network, and another as returning the computed output. The connections go from one neuron to another and they have associated numeric weights that can be adjusted to obtain an approximation of a function implicitly given by pairs of input and output training examples.

Feedforward neural networks (also known as multilayer perceptrons), are a commonly-used type of artificial neural networks. In these models, the neurons that form the different layers have connections that go only in one direction: from the input layer, passing through a number of intermediate hidden layers, and going into the output layer. Figure 3.3 shows a graphical representation of a feedforward neural network with a single hidden layer, such as the one used

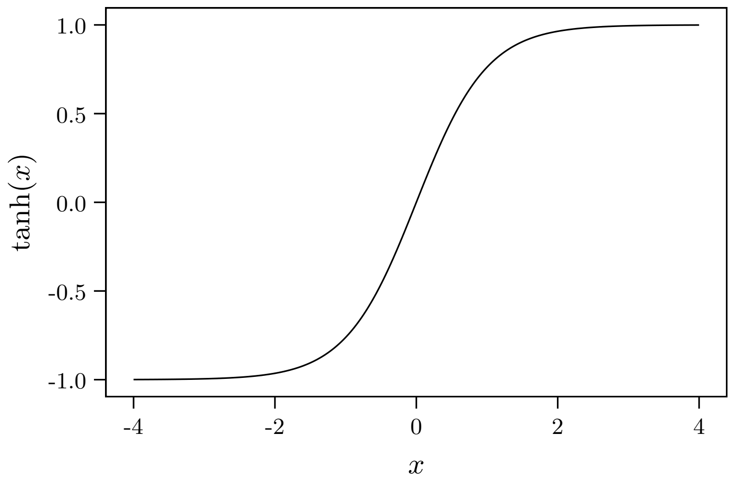

Figure 3.4: Plot of the hyperbolic tangent activation function.

in this thesis. The input layer contains n units denoted by $x_1, \ldots, x_n$ which are all connected to the l units $h_1, \ldots, h_l$ in the hidden layer. Each unit in the hidden layer is in turn connected to the m units $y_1, \ldots, y_m$ that constitute the output layer of the network. The network also contains two additional neurons x_0 and h_0 (called bias units) connected to the hidden and output layers, respectively. The weights associated with the connections that go into the hidden layer are denoted by $w_{ji}^{(1)}$, and the ones that go into the output layer by $w_{kj}^{(2)}$.

Given the network structure described above, the function approximated by the artificial neural network is evaluated as follows. Each neuron in the network has an associated function (called the activation function) that is used to compute the output of the unit. The output of each neuron is computed by calculating first the sum of its input values scaled by their respective connection weights, and then applying the activation function to this sum. In the experiments performed in this thesis, identity activation functions are employed with the neurons in the input layer, and the hyperbolic tangent activation function given by

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.1}$$

is used in the hidden and output layers (see Figure 3.4). The bias units use an activation function that returns a constant value of 1 to allow obtaining a nonzero output from the input and hidden layers even if the the output of all the neurons in these layers is zero. Overall, the value of each output unit $y_k \in \{y_1, \ldots, y_m\}$ shown in Figure 3.3 is calculated as follows:

$$y_k = \tanh\left(\sum_{j=0}^{l} w_{kj}^{(2)} \tanh\left(\sum_{i=0}^{n} w_{ji}^{(1)} x_i\right)\right). \tag{3.2}$$

When the network is trained with a set of input and output examples (such as the patterns generated from the $\langle s, a, r, s' \rangle$ transitions in the NFQ algorithm), the weights $w_{ji}^{(1)}$ and $w_{kj}^{(2)}$ are adjusted to make it return the same output observed in the training data. This training process can be seen as an optimization process in which the error between the returned values and the desired output must be minimized. In the context of artificial neural networks, this minimization is carried out by specialized gradient-descent algorithms (such as the RPROP algorithm [42] used in NFQ) that propagate the error backwards through the network and adjust the weights in the direction of the negative gradient of the error.

3.3.2.2 Representation of the Action-Value Function

Several alternative artificial neural network architectures can be employed to represent the action-value function $Q(s, a)$ [50]. Considering that the neural network must be responsible for mapping the state-action pairs $\langle s, a \rangle$ to their corresponding scalar values $Q(s, a)$, it is possible to use either one of the following approaches:

i. A separate network for each action with s as input and a single scalar output $Q(s, a)$.

ii. One network with both s and a as input and a single scalar output $Q(s, a)$.

iii. One network with s as input and a separate scalar output $Q(s, a)$ for each action a.

The third approach was selected in this thesis, taking into consideration the experiences reported by van Eck and van Wezel [49], Riedmiller [51], and Mnih et al. [52]. Specifically, a single neural network with separate outputs for each action offers two important advantages. First, because a single network is used to represent the Q function, it is possible to generalize the information provided by training examples over both states and actions, whereas generalization is only possible over states when a separate network is used for each action. Secondly, by having one output for each action in the same network, a single forward evaluation of the neural

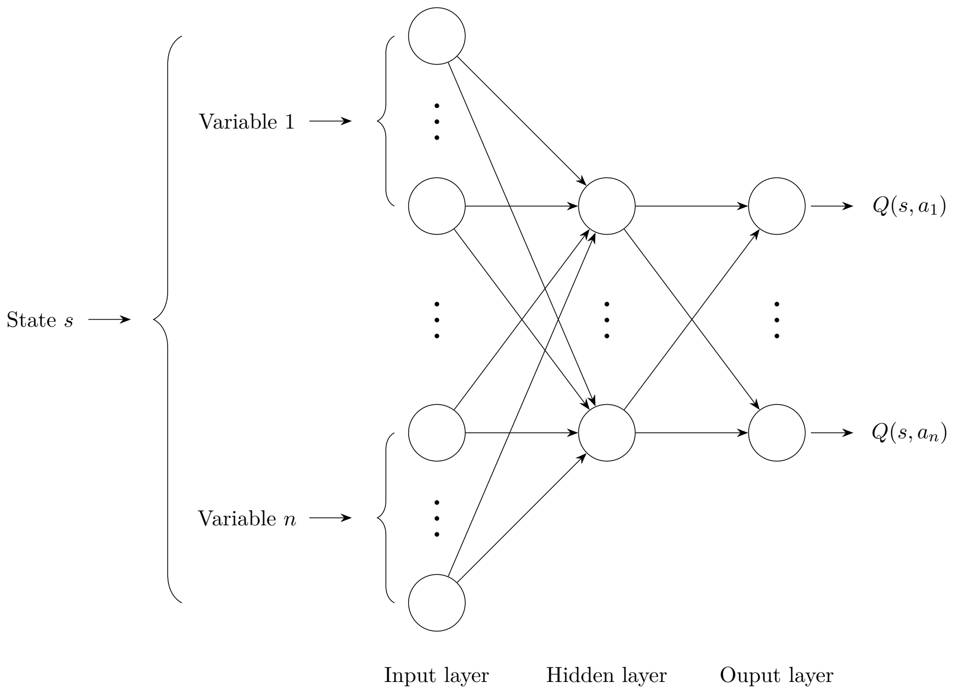

Figure 3.5: Illustration of the neural network used in neural fitted Q-iteration. The bias units connected to the hidden and output layers have been omitted for brevity.

network is sufficient to determine the expected reward of all actions at a given state—reducing the computational cost of selecting the best action at each state as a result.

Figure 3.5 illustrates the architecture of the artificial neural network used along with NFQ for the optimization of the user interaction in a configuration process. The network is composed of three layers of neurons: an input layer, one hidden layer, and an output layer connected in a feedforward network structure as described in the previous section. The input to the neural network specifies a configuration state s of the MDP, represented with a number of input units for each configuration variable. The input layer is followed by a single hidden layer with as many neurons as the number of variables in the configuration model. Lastly, the output layer contains one output unit for every action $a_1, \ldots, a_n$ in the MDP (i.e., also the same as the number of variables). The output units return the estimated Q values for every action in the state s given as the input to the network.

The configuration variables in the state s are coded using a number of so-called ‘dummy’ auxiliary variables (see e.g. Draper and Smith [53] for a detailed explanation). Using this coding scheme, the values in the domain D_i of each variable $V_i \in V$ are represented as $|D_i|$ binary variables, each one corresponding to one of the possible values of the variable. If a variable does not have a value assigned in the configuration state s, then all the auxiliary variables are set to zero. Otherwise, only the auxiliary variable corresponding to the assigned value is given the value one, and the rest is assigned the value zero. As it is usually suggested for improving the training efficiency of the neural network (e.g., by LeCun et al. [54]), the input values one and zero are presented to the neural network as 1 and -1, respectively.

As mentioned in the previous section, the units in the output layer of the neural network use the hyperbolic tangent activation function. Therefore, the network returns values between -1 and 1 (see Figure 3.4), and they must be scaled before being interpreted as cumulative rewards in the formulated MDP. Given that at least one question must be asked in a configuration episode with n configuration variables, the cumulative rewards will always be between 0 and $n-1$. Hence, the output values are linearly scaled from the $[-1, 1]$ interval to $[0, n-1]$.

One important remark about this neural network representation of the action-value function $Q(s, a)$ is that no specially-crafted features about the problem being solved are given to the model. As it was described above, the input to the neural network is simply the variable assignments in a particular configuration state, and thus the network has to learn by itself useful features for assessing the importance of taking each action at every configuration state. Intuitively, the neural network achieves this goal by discovering interactions between the configuration variables and how they influence the result of the action-value function. These interactions are represented by means of the weights assigned to the different connections between the neurons in the network.

3.3.3 Direct Policy Search Using a Genetic Algorithm

Solving a sequential decision-making problem formulated as an MDP using evolutionary algorithms involves choosing an explicit policy representation and searching directly in the policy space for a solution that gives the highest expected cumulative reward (see Section 2.2.4 for details). Each candidate policy is evaluated by considering the average cumulative reward obtained in a number of simulated episodes as its expected cumulative reward. Under this formulation, the problem of finding an optimal policy for an MDP becomes a stochastic optimization problem.

The second solution method for the optimization of the user interaction in a configuration process proposed in this thesis consists of solving the associated MDP using a GA. The first part of this section describes the developed solution representation and the fitness evaluation. The second part provides the details of the genetic operators used in the GA implementation (i.e., selection, crossover and mutation).

3.3.3.1 Policy Representation and Fitness Evaluation

The use of a GA to solve an MDP gives great flexibility for choosing a policy representation. The solution of the MDP associated with the optimization of the user interaction in a configuration process presented in this section exploits this flexibility to develop a compact policy representation by considering more closely the underlying structure of the MDP state space.

Following the MDP formulation described in Section 3.2, if the action selection is restricted to questions about variables whose values have not yet been assigned in a given state, then every transition will lead to a different state in which the value of a greater number of configuration variables is known. This is a direct consequence of the incremental definition of the configuration process described in Section 3.1. Every interaction with the user produces a new configuration decision. Therefore, the state space in the formulated MDP is acyclic in the sense that it is not possible to return to a previously-visited state (or more generally, to a state with fewer known configuration variables).

This acyclic structure of the MDP, along with the episodic constitution of the configuration task, motivates the use of a simpler policy representation that is still able to capture the essential information about the solutions of the problem. In order to minimize the user interaction, the policy must provide a way of deciding which question (related to an undecided variable) should be asked at each iteration of the configuration process. This purpose can be achieved by defining preference relations between the questions, given in terms of their expected cumulative reward when asked before or after other questions. Specifically, when the GA is used to solve the MDP, the goal of finding an optimal mapping of every possible state to an action is transformed into searching for a permutation sequence of the configuration variables producing the highest expected cumulative reward.

A (permutation) sequence of the configuration variables determines an order that must be followed for asking their corresponding questions, and this order effectively constitutes a policy for guiding the configuration process. The first question dictated by the sequence is asked at the beginning of the configuration process, followed by the next unassigned variable in the sequence after receiving each user response and propagating its implications. The GA can evaluate the fitness of a solution by computing the average cumulative reward (i.e., the number of variables that are automatically discovered) obtained in a number of simulated episodes following the permutation sequence. Notice that in this case the values of the variables are not considered as part of the policy representation, but are nonetheless taken into account indirectly through the episodes simulated to evaluate the corresponding policy's performance.

3.3.3.2 Genetic Operators

The remainder of this section presents a description of the genetic operators that work with the permutation sequence policy representation as part of the complete pseudocode of the GA presented in Figure 2.4. The initial population of solutions evolved by the GA is generated by sampling uniformly random permutations of the indices $1, 2, \ldots, n$ corresponding to the configuration variables in V. Then, the GA proceeds to produce subsequently improving populations of candidate solutions after evaluating the existing solutions and generating new ones based on the best-performing policies in each population.

There are several methods available for selecting the best-performing solutions, given a population of solutions and their associated fitness evaluation. The implementation of the GA used in this thesis relies on binary tournament selection [46]. This procedure involves randomly selecting two solutions using a uniform probability distribution, and then choosing the best one (i.e., the one with highest expected cumulative reward) as the tournament winner and considered for recombination. Two such tournaments are performed to select the parents to be recombined with the crossover operator. This selection procedure is equivalent to assigning the selection probability of the solutions linearly according to their fitness rank [55].

By having the MDP policies encoded as permutation sequences, it is possible to take advantage of the considerable number of genetic operators that have been proposed and widely studied for this representation in the context of combinatorial optimization problems [56, 57]. However, not all crossover operators working on permutation sequences seem appropriate for the recombination of policies as encoded in the problem at hand. As reviewed by Chen and Smith [58, 59], some operators (e.g., order crossover [60]) were designed for problems in which the solutions are modelled

Parent 1:	$\langle 2, 5, 1, 3, 4 \rangle$
Parent 2:	$\langle 2, 4, 3, 5, 1 \rangle$
Parent selection:	$1, 2, 1, 2, 2$
	$\langle \rangle$
	$\langle 2 \rangle$
	$\langle 2, 4 \rangle$
	$\langle 2, 4, 5 \rangle$
	$\langle 2, 4, 5, 3 \rangle$
Offspring:	$\langle 2, 4, 5, 3, 1 \rangle$

Figure 3.6: Offspring generation using the precedence preservative crossover operator.

as a Hamiltonian cycle, and their cyclic constitution is irrelevant in this scenario. Others, such as the edge recombination operator [61], focus on how the different permutation elements occur side-by-side (i.e., their relative order). Conversely, in a sequence of questions encoding an MDP policy for optimizing the user interaction in a configuration process, it is of particular interest to consider how the different permutation elements precede each other regardless of how many elements may lay in between. Once a question is asked, the value of the associated variable remains available for all the upcoming configuration decisions. Therefore, an appropriate crossover operator for this problem must be able to work with the absolute order between the different elements of the permutation sequence.

The precedence preservative crossover operator (or PPX) introduced by Bierwirth et al. [62] in the domain of job-shop scheduling problems [63] satisfies those requirements. This operator passes on absolute precedence relations among the elements of two parental solutions to one offspring as follows. A list of the same length as the solutions is randomly generated by sampling with replacement over the set $\{1, 2\}$. This list serves as a reference that defines the order in which elements are successively drawn from each one of the parents identified with the numbers 1 and 2, respectively. The PPX operator first initializes an empty offspring, and then the leftmost element in one of the two parents is selected in accordance to the sampled reference list. After an element has been selected, it is deleted from both parents and appended to the offspring. These steps are repeated until both parental sequences are empty, and hence the offspring contains all elements in the sequence. Figure 3.6 illustrate this procedure with an example.

By using the precedence preservative crossover operator to generate new solutions, it is guaranteed that every preference relation encoded in a newly generated solution comes from at least one of the two parents. Therefore, if it were used as the the only mechanism for generating new permutation sequences, the evolution of the GA would be restricted only to precedence relations that occurred in the initial population. This situation may cause the search for good policies to become stalled because of the lack of diversity in the generated solutions. A mutation operator is incorporated with low probability to counteract this effect. A simple swap mutation operator that interchanges the positions of two randomly selected components in the permutation sequence suffices for this purpose.

Solutions representing the best-performing policies have higher changes of being selected as tournament winners. Accordingly, they also have higher changes of passing on to new solutions the good preference relations among the different questions they encode. By repeating this procedure generation after generation in the GA, the population of solutions is expected to increasingly improve towards better-performing policies.

3.4 Summary

This chapter introduced the two alternative configuration models considered in the thesis along with the formulation as an MDP of the problem of optimizing the user interaction in a configuration process and the proposed solution methods. The introduced framework works in a similar way with both configuration models, relying on the corresponding inference mechanisms to avoid asking questions about variables that can be answered automatically from other user's decisions. The task of the algorithms solving the formulated MDP is then to find an optimal policy that dictates which question should be asked at each state of the configuration process in order to minimize the necessary user interaction.

Two solution methods are proposed for finding an optimal policy, each one employing a different policy representation. The first one consist of using the NFQ algorithm to train a neural network receiving a representation of the current configuration state as the input and returning the expected number of automatically-discovered configuration decisions that can be obtained from asking each unanswered question at that state as the output. The second solution method uses a GA to find a permutation sequence of the configuration variables that determines the order that must be followed for asking their corresponding questions.

It must be noted that the solution of the MDP (using either NFQ or the GA) takes place offline and not during the interactive configuration process in which the users intervene. The optimization problem is solved beforehand by considering all possible configuration scenarios and finding an optimal policy that minimizes their expected user interaction. The interactive configuration processes with the users take place afterwards and it simply follows the questions dictated by the optimal policy at each configuration state (i.e., the optimal neural network or permutation sequence if NFQ or the GA are used, respectively). This approach means that each inquired variable is not the result of a decision about which question seems the most appropriate locally, but a consideration of its global effect in the entire configuration process. The next chapter evaluates the performance and scalability of the proposed solution methods.

4 Evaluation

This chapter presents the results of the study conducted for the evaluation of the solution methods proposed in Chapter 3. The first part of the chapter discusses the evaluation goals and outlines the experimental setup. The results of the specific experiments are described next, with one section dedicated to the analysis of rule-based configuration models and another one to constraint-based models. Lastly, the overall evaluation results are summarized in Section 4.4.

4.1 Goals and Experimental Setup

The pursued evaluation goals are directly related to the objectives of the thesis discussed in Section 1.2. The experiments that were carried out aim to validate the following hypotheses:

i. First, that the proposed solution methods overcome the scalability limitations of the framework introduced by Hamidi et al. [5] for rule-based configuration models.

ii. And secondly, that these new techniques are also able to work effectively with the more general CSP specification of the configuration models.

The study considers a group of rule-based and constraint-based configuration models, while focusing on two evaluation metrics. The first one is the scalability of the solution methods measured in terms of the running time of the algorithms versus the number of configuration variables. The second evaluation metric is the performance of the techniques given by the number of questions asked to a user during the configuration of an artifact.

The interpretation of the second evaluation metric deserves special attention to ensure that an adequate assessment is made about the effectiveness of the different techniques. Given a complete configuration containing the answers of a user to all the configuration variables in a model, the best of two or more alternative solution methods can be easily selected by choosing the one that requires the fewest questions to replicate that configuration starting from an empty set of variable assignments. However, this comparison may not provide enough information regarding how well the techniques solved the problem. It is more useful to work with a performance baseline that allows evaluating the quality of the solutions provided by each technique taking into consideration the characteristics of the configuration model. This baseline may be set at either one of the two extreme performances—i.e., the best or worst attainable performances.

Considering the worst attainable performance looks appropriate and simpler to compute at first sight, given that it intuitively seems to correspond to the total number of configuration variables in the model. However, a number of questions may be always skipped because of the effect of the inference mechanisms, regardless of the order in which the configuration variables are inquired. As a result, a configuration process that requires asking a question for every configuration variable may never be attainable in practice. This characteristic may lead to an overestimation of the the number of questions that are skipped, and a claim about the effectiveness of the algorithms made considering the total number of configuration variables may not be necessarily meaningful.

A more adequate evaluation can be accomplished by comparing the results of the alternative solution methods with the best attainable performance on a selected group of complete configurations. From this comparison, it is possible to calculate an error value representing the number of additional questions asked by each technique. This evaluation criterion depends on having computed beforehand the shortest sequence of questions that could be asked to a particular user, based on a complete specification of his or her configuration intentions. This requirement effectively translates into the solution of an optimization problem that involves finding a permutation of the variables that minimizes the number of questions asked to that specific user. Notice that this resulting optimization problem is similar to the stochastic optimization problem solved by the proposed GA-based solution method (see Section 3.3.3), but with the important difference that the uncertainty in the user's responses is removed because they are given beforehand—making it non-stochastic. The optimal performance on each complete configuration selected for the evaluation of the alternative solution methods can be computed exactly using exhaustive search for small configuration models (i.e., trying out all possible permutation sequences of the variables), or approximately if brute-force search is not feasible because of the problem dimensions.

The overall experimental setup to evaluate the performance of the alternative solution methods on a particular configuration model proceeds as follows. First, a number of complete configurations representing the intentions of a group of users are selected for evaluating the techniques. For each one of these configurations, the minimum number of questions that could be asked to a user to reproduce it is computed as described in the previous paragraph, and this number serves as the performance baseline to compare all the alternative solution methods. This group of selected configurations along with their best attainable performances constitute the testing data set. Then, each alternative solution method is used to construct a policy for optimizing the user interaction in the configuration model, and these alternative policies are evaluated according to their performance when reproducing the configurations in the testing data set. Given that the proposed solution methods rely on simulations of configuration episodes, the algorithms are run 30 times with different random seeds and their expected performance in the testing data set is calculated as the average performance observed in the 30 runs. The reported results correspond to the error values obtained on each configuration in the testing data set.

In addition to the solution methods based on NFQ and the GA proposed in the thesis, the performance experiments also include the results of a policy that corresponds to asking the questions in a random order. This solution method intends to represent the idea of not using an advanced technique to select the order in which the questions are asked. The motivation for comparing with a random sequence is twofold. On the one hand, it shows statistically that the techniques exhibit an 'intelligent' behaviour in the selection of the sequence of questions presented to the users—i.e., that the policies capture useful information for the optimization of the user interaction. On the other hand, it facilitates quantifying the effect of using the proposed techniques in the studied models. The reported results for the random solution method also correspond to the average performance observed in 30 independent runs.

Further details are given in the remainder of the chapter along with the description of each specific experiment. All the alternative solution methods were implemented in the Python programming language as part of the work carried out in this thesis[2]. The algorithms were run on a 32-bit Linux platform with an Intel(R) Xeon(R) CPU E5405 at 2.0 GHz and 2 GB of RAM.

[2] The source code is available at `http://github.com/yasserglez/configurator`.

4.2 Rule-Based Configuration Models

The experiments described in this section are related to the first evaluation goal. Their purpose is to compare the solution methods proposed in the thesis with the dynamic programming techniques used by Hamidi et al. [5]. The main consideration is the scalability of the running time of the algorithms as the number of configuration variables grows. In addition, the case study employing the configuration records from the Facebook social network platform collected by Hamidi [10] is recreated using the techniques proposed in this thesis.

4.2.1 Scalability Results

In the absence of a collection of rule-base configuration models with an increasing number of variables to support performing a scalability study, Hamidi [10] resorted to generating artificial models from a categorical data set known to contain an abundant number of association rules. Each column in the data set can be interpreted as a configuration variable whose domain is given by the values of the variable in the data set. In turn, each row can be seen as a complete configuration; and consequently, it is possible to use association rule mining to discover rules that are relevant to the (artificial) configuration model. By considering an increasing number of columns and repeating the association rule mining process, it is possible to generate a collection of related configuration models with a growing number of variables that facilitate measuring the scalability of the running time of the algorithms. The same experimental setup was followed to obtain the results reported in this section.

The selected categorical data set is the mushroom data set available at the UCI Machine Learning Repository[3]. This data set contains descriptions given in terms of 23 categorical variables of hypothetical samples corresponding to different mushroom species. There is a total of 8,124 observations in the original data set, but their use in this thesis was limited to the 5,644 existing complete observations. After filtering out the observations that have at least one missing value, the domains of the 23 discrete variables vary form 2 to 9 values.

[3] https://archive.ics.uci.edu/ml/datasets/mushroom

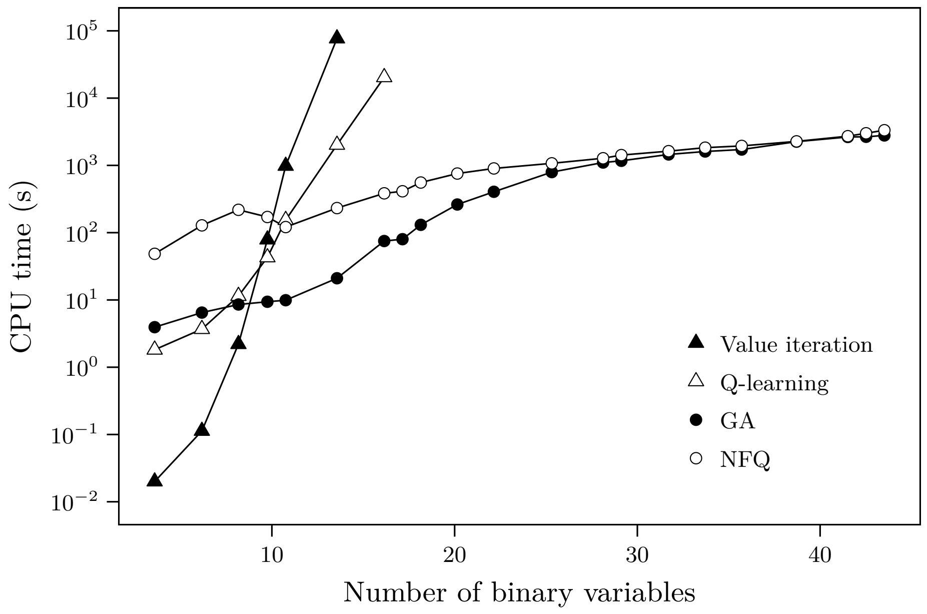

Figure 4.1: Scalability results on the rule-based configuration models artificially generated using the mushroom UCI data set. The vertical axis is in logarithmic scale to compensate for the considerable differences in the running time of the algorithms.

The generation of the artificial configuration models to measure the running time of the alternative solution methods begins with a problem involving two variables from the data set. Subsequently, one variable is added at a time, resulting in a new configuration model. All the variables were selected randomly, but the same order was followed for all the solution methods. For each subset of the variables corresponding to an artificial configuration model, the association rules were mined using the FP-grow algorithm [64] (see Borgelt [65, 66] for information about the specific implementation used in the experiments). The rule mining process was targeted at rules with a minimum support of 50%, a minimum confidence of 90%, and having only one variable in the *if* part and the *then* part of the rule, respectively. The number of rules in the configuration models ranged from one rule in the smallest model to 70 rules in the largest.

Figure 4.1 illustrates the results of the scalability experiments on rule-based configuration models. The dimensions of the configuration models are given in terms of the number of binary variables necessary to encode all the possible configurations. This number is calculated as the logarithm to the base 2 of the total number of configurations, and it facilitates comparing the results obtained on configuration models with different variable domain cardinalities. The artificially-generated configuration models correspond to models having between 2 and 44 binary variables. The algorithms were run 30 times on each configuration model, recording the CPU time elapsed while searching for the optimal policy. The data points in the figure correspond to the mean CPU time observed in the 30 runs (the error bars were removed from the plot because they were not visible with the selected scale).

Four algorithms were considered in the comparison. The first one is value iteration, which is an example of the dynamic programming techniques employed by Hamidi et al. [5]. Q-learning is the first reinforcement learning technique discussed in Section 2.2.3, which uses a tabular representation of the action-value function and is known to have limited applicability in practice but was included in the study for completeness. The final two algorithms, the GA and NFQ, are the solution methods proposed in this thesis. Appendix A includes a table that summarizes the values of the parameters of each algorithm as they were used in these experiments.

All the algorithms were run on configuration models with an increasing number of variables until the mean elapsed CPU time for solving one problem surpassed one hour (i.e., roughly $10^{3.5}$ seconds). Value iteration required approximately 21 hours to solve a problem with 14 binary variables before it was stopped, and Q-learning 6 hours for solving a problem with 17 variables. The GA and NFQ both completed the entire experiment, solving the largest configuration problem with 44 binary variables in approximately 46 and 56 minutes, respectively. More importantly, the steepness of the curves associated with the different algorithms in Figure 4.1 characterizes the scalability of the different solution techniques. A comparison of the curves corresponding to value iteration, the GA and NFQ evidences that the former two scale considerably better as the number of configuration variables in the model becomes larger. This result confirms that the solution methods proposed in the thesis are more suitable for optimizing the user interaction in configuration models with a large number of variables. Nevertheless, it must be noted that the curves corresponding to the GA and NFQ still show a slight linear growth in the logarithmic scale and this indicates that the algorithms might face scalability limitations for larger models.

4.2.2 Case Study: Facebook Privacy Settings

The second part of the experiments using rule-based configuration models consists of recreating the case study conducted as part of the validation of the framework introduced by Hamidi et al. [5]. The authors collected configuration records corresponding to the privacy settings of a group of users for the Facebook social network platform, and proceeded to study the performance of the approach based on the dynamic programming solution methods for the optimization of the user interaction in this configuration scenario. The motivation behind the experiments reported in this section is to evaluate whether the techniques proposed in this thesis effectively reduce the number of questions asked to the users in this real-world problem.

Facebook[4] is an online social network platform with over 1.4 billion monthly active users as of the first quarter of 2015[5]. Similar to other social network sites [67], Facebook allows its users to create a personal profile with customized visibility, define a list of users with whom they share a connection, and interact with the resulting network of connections (by e.g. viewing other users' profiles in the network, posting textual information or in a variety of multimedia formats, and interacting with what other users have published) [68]. As a way of controlling the amount of information a user shares with other users, the platform offers a set of variables that the users can configure to meet their privacy needs. These configuration variables are set to certain default values upon creation of a new profile, and at any time the users can adjust them by going to a settings page where they are presented along with their possible choices.

As surveyed by Hamidi [10], privacy in Facebook has been a subject of intense academic research [68–75] as well as discussion in mass media communication. Studies have indicated that not all users are aware of the implications of the privacy settings [70, 72], and that they tend to overestimate the strength of their chosen privacy options [74]. In addition, the default values of the configuration variables are targeted mostly at sharing information broadly [73], which is concerning given that a considerable number of users have reported to keep the default settings [74]. Therefore, the problem of privacy configuration in Facebook constitutes a good example of the importance of assisting users in the configuration of complex software systems.

[4] http://www.facebook.com

[5] See Facebook's *Q1 2015 Earnings* report at http://investor.fb.com/eventdetail.cfm?eventid=158508.

Table 4.1: Facebook privacy settings.

Section	Question
1. Privacy	a) Who can see future posts? b) Who can look you up using the email address or phone number you provided? c) Do you want other search engines to link to your timeline?
2. Timeline & Tagging	a) Who can post on your timeline? b) Who can see what others post on your timeline? c) Who can see posts you've been tagged in on your timeline? d) Review posts friends tag you in before they appear on your timeline?
3. Friends	a) Who can send you friend requests? b) Who can see friend list?

Hamidi et al. [5] collected a data set with complete configurations of the privacy-related configuration variables of 45 Facebook users. The participants in the study were at the time undergraduate students attending the School of Information Technology at York University, Ontario, Canada. A detailed description of the data collection procedure and summary statistics about the participants are given by Hamidi [10]. Table 4.1 presents the nine selected privacy configuration options. As of April 2014, these options were located under the 'Privacy', 'Timeline & Tagging', and 'Friends' sections of the Facebook settings page. The possible answers for questions 1c) and 2d) are 'Yes' and 'No'. The other questions refer to access levels in the user's connection network and their possible answers are 'Everyone', 'Friends of friends', 'Friends', 'Custom', and 'Only me'. Under the 'Custom' answer, the users can specify any subset of their connections, but it was treated in the study as a single coarse-grained answer.

The nine discrete variables account for a total of 312,500 possible configurations, which is roughly equivalent to a configuration model with 19 binary variables ($\log_2 312,500 \approx 18.25$). However, in the experiments conducted by Hamidi et al. [5], a reduced version of the model was studied. The authors considered only the values of the variables occurring in the 45 collected configuration records when defining the domains of the configuration variables in the model. As a result, the configuration model is reported as having the equivalent of 12 binary variables (instead of 19). This consideration translates into an important limitation: the resulting policies for the optimization of the user interaction in the configuration process cannot be used to guide the users when they give an answer that is not part of the considered domains. Therefore, in this replication of the case study the original formulation of the configuration model for the Facebook privacy settings is used, and not the reduced version. Since it is impractical to solve a configuration problem with 19 binary variables using the dynamic programming techniques employed by Hamidi et al. [5] (see the scalability results reported in the previous section), this new study focuses only on the solution methods proposed in the thesis and does not attempt to make a direct comparison with the performance obtained with value iteration—or the tabular Q-learning algorithm for that matter.

The solution methods proposed in the thesis rely on simulating configuration episodes in the form of interactions with the MDP formulated for minimizing the number of questions asked during a configuration process (see Sections 3.2 and 3.3). Given that in the case of the Facebook privacy settings there exists a data set with configuration records from a group of real users, it is possible to exploit this information and consider the intentions of the users during the simulation of the configuration episodes. Specifically, the collected configuration records can be used to compute the conditional probabilities of the user's responses given each partial configuration during the simulated episodes. In order to assess whether the constructed policies would be able to generalize to unseen configuration records, a cross-validation approach is followed for the evaluation of the performance of the different solution techniques.

The experiments were carried out following a k-fold cross-validation setup with $k = 9$. The 45 configuration records available were randomly divided into 9 equal-sized partitions of 5 records. Then, each individual partition of 5 configurations was considered in turn as a testing data set for computing the error values in terms of the best attainable performance as discussed in Section 4.1, while using the remaining 40 configuration records as a training data set to compute the conditional probabilities for finding the optimal policy. Figure 4.2 shows a histogram of the best attainable performance (i.e., the minimum possible number of questions) on the 45 complete configurations computed exactly using exhaustive search. These results are considered as the baseline for comparing the performance of the different algorithms.

The mean and standard deviation of the error values (with respect to the performance baseline

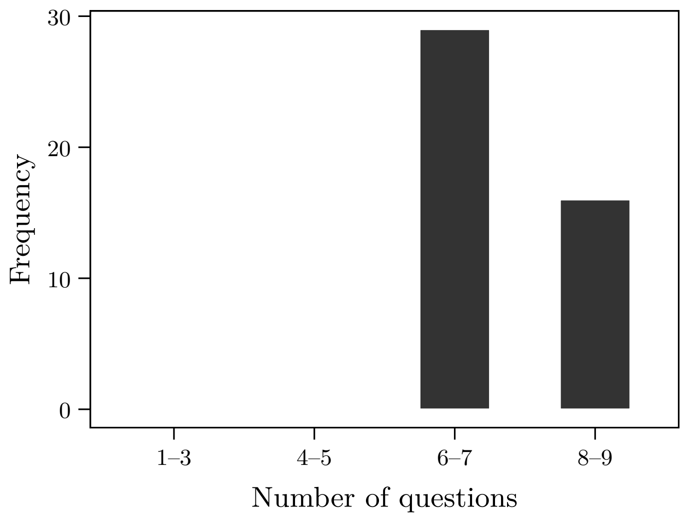

Figure 4.2: Histogram of the best attainable performance (i.e., the minimum possible number of questions) on the Facebook privacy settings problem.

given by the minimum possible number of questions) obtained by the alternative solution methods on each fold are given in Table 4.2. In addition, Figure 4.3 presents a box plot of the distribution of the errors for each algorithm and cross-validation partition. Similarly, Figure 4.4 shows a box plot of the distribution of the errors obtained by the algorithms on all the individual folds combined (the corresponding mean and standard deviation values are given in the last row of Table 4.2). As mentioned before, the results of each algorithm on the testing data sets correspond to the average performance observed in 30 independent runs with the parameters given in Appendix A.

Overall, the results obtained by all the solution methods on the Facebook privacy settings problem are very close to the best attainable performance. The mean error on the different cross-validation folds is always below 0.5, which indicates that the techniques ask a number of questions that is in all cases less than one question away from the optimal performance on average. These results also suggest that the solution methods based on NFQ and the GA do not tend to overfit and are able to generalize from the configurations observed during the simulated episodes. The mean error values reported in the last row of Table 4.2 and the visualization of the combined error distributions shown in Figure 4.4 suggest that the GA-based technique obtains better results than NFQ, and that using any of these two solution methods is better than asking the questions in a random order.

Table 4.2: Error values obtained on the Facebook privacy settings problem.

Fold	GA		NFQ		Random	
	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
1	0.23	0.29	0.33	0.25	0.41	0.33
2	0.00	0.00	0.09	0.19	0.19	0.13
3	0.02	0.03	0.20	0.18	0.26	0.11
4	0.16	0.34	0.27	0.25	0.33	0.22
5	0.11	0.24	0.25	0.43	0.37	0.40
6	0.01	0.01	0.03	0.04	0.16	0.13
7	0.00	0.00	0.03	0.03	0.15	0.10
8	0.01	0.01	0.15	0.18	0.23	0.11
9	0.11	0.24	0.13	0.28	0.28	0.22
All	0.07	0.19	0.16	0.23	0.26	0.22

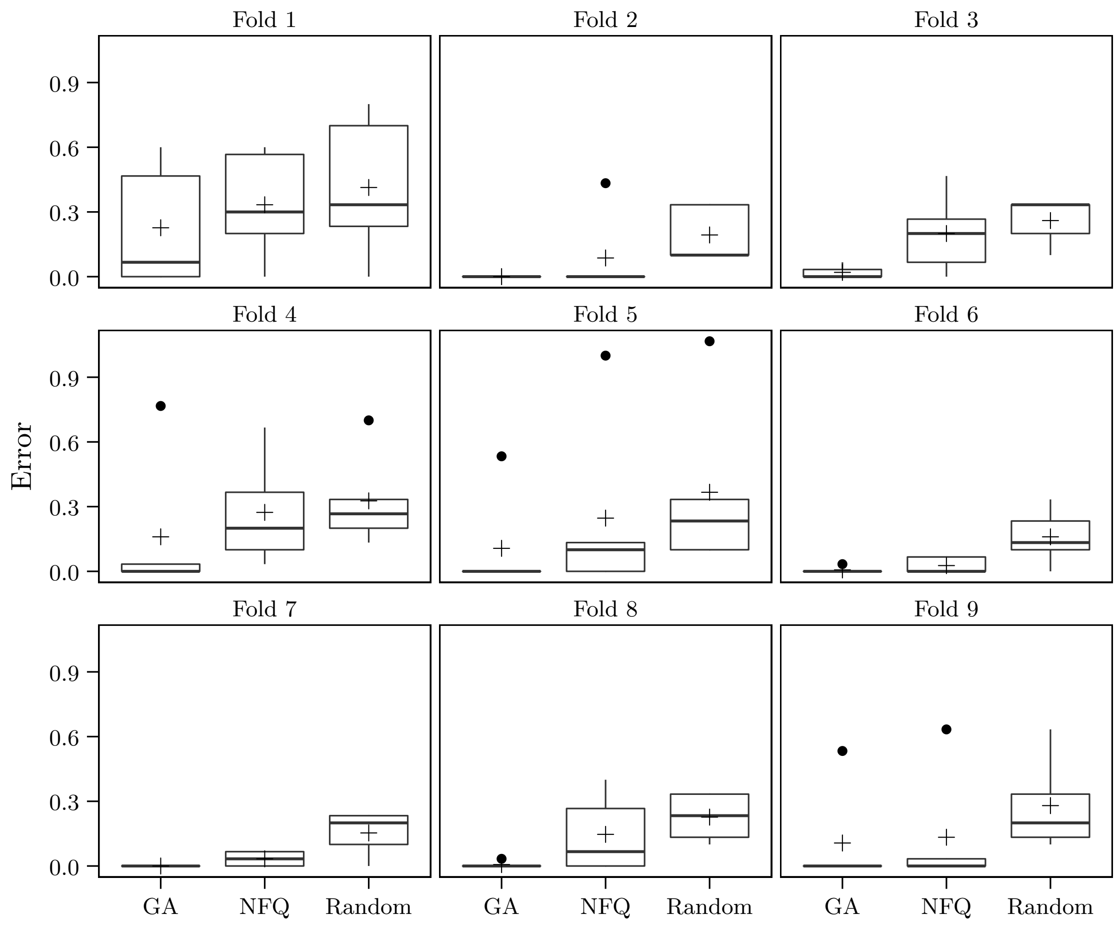

Figure 4.3: Box plots of the individual cross-validation results on the Facebook privacy settings problem. The crosses represent the mean error values.

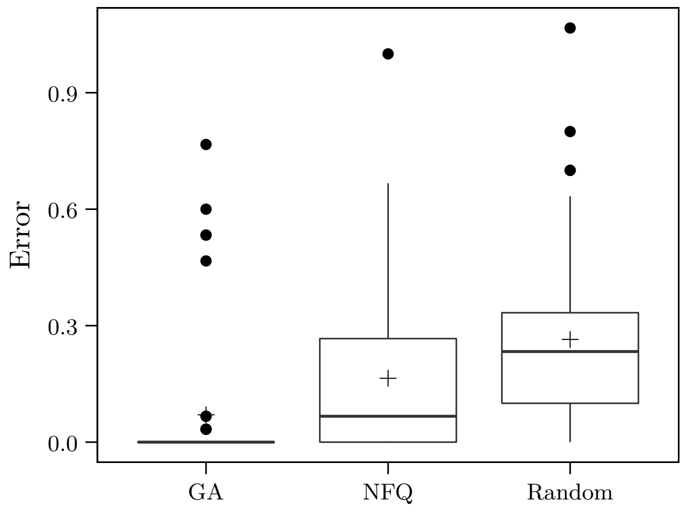

Figure 4.4: Box plot of the combined cross-validation results on the Facebook privacy settings problem. The crosses represent the mean error values.

Since asking the questions arbitrarily also seems to produce good results in this problem, this last conclusion was examined more closely. A statistical analysis was carried out using non-parametric tests, given that the shape of the distributions shown in Figure 4.4 departs from normality. First, the distributions of the errors obtained by the three solution methods were compared using the Friedman statistical test [76] to decide whether statistically significant differences occur among the examined algorithms. A p-value of 2.14e$-$14 was obtained, which suggests that there is strong evidence in the collected results (at the 1% significance level) that at least one of the algorithms is superior to the others. Then, a number of post-hoc tests were performed to compare all the pairs of algorithms looking for statistically significant differences. The error distributions of NFQ vs Random, GA vs Random, and GA vs NFQ were compared using a paired Wilcoxon signed rank test [77] obtaining the following p-values adjusted for multiple comparisons using the Bonferroni correction [78]: 6.75e$-$06, 5.06e$-$08 and 0.000126, respectively. The null hypothesis that the samples come from the same error distributions can be rejected at the 1% significance level in all cases. Therefore, the statistical analysis in effect supports the conclusion that the GA has better performance than NFQ, and in turn the use of any of these two algorithms produces better results than following an arbitrary order for asking the questions during the configuration process.

Table 4.3: Characteristics of the constraint-based benchmark problems.

Problem	Variables		Constraints		Configurations	
	Discrete	Binary	Number	Cardinality	Possible	Consistent
Cellphone [29]	11	10	8	2–4	1,024	14
Text editor [79]	18	17	15	2–4	13,1072	396
Graph manipulation [80]	30	29	32	2–3	5.3e+08	192
Dell laptops [81]	27	57	98	2	7.3e+16	> 1e+06

Given that essentially different interpretations of the configuration model and distinct association rules were considered, it is not possible to make a direct comparison of the results reported by Hamidi et al. [5] on the case study on the Facebook privacy settings and the ones reported in this thesis. Still, both studies seem to give consistent results in showing that it is possible to reduce the number of questions asked to the users. Hamidi et al. [5] reports to avoid asking two questions to the users on average, and in the present study at least one question was skipped on average. However, it is worth mentioning that considering the total number of variables in the configuration model as a baseline performance measure does not necessarily reflect the algorithms' effectiveness as it was discussed in Section 4.1.

4.3 Constraint-Based Configuration Models

The group of experiments presented in this section evaluates the proposed solution methods for the optimization of the user interaction in a configuration process with a constraint-based configuration model formulated as a CSP. They contribute to the second evaluation goal discussed in Section 4.1. As in the case of rule-base models, the evaluation focuses on the performance of the techniques in terms of the number of questions asked to reproduce a complete configuration, and the scalability of the algorithms as the number of configuration variables increases.

4.3.1 Benchmark Problems

Table 4.3 describes the four constraint-based configuration models selected for the evaluation of the algorithms. All the problems except the Dell laptops model were obtained from the Software

Product Lines Online Tools (S.P.L.O.T.) [82] feature model repository[6] and they cover different application domains. The cellphone model corresponds to the feature model shown in Figure 2.1 and used as illustration throughout the thesis. The text editor model describes a configurable text editor including features such as syntax highlighting, spell checking and automatic backups. Graph manipulation is a feature model corresponding to a software for editing graphs with features associated with deleting and selecting nodes, zooming in and out, etc. The Dell laptops model is a decision model for a laptop configurator reverse-engineered from the Dell website and available for download at the C2O (Configurator 2.0) [83] homepage[7]. Additionally, the table includes references to the academic publications that introduced each model (in some cases as simplified or alternative versions).

All the models were transformed to an equivalent CSP formulation compatible with the proposed algorithms (see Section 2.1.2). The following characteristics of the resulting CSP are summarized in Table 4.3: the number of variables, the number of constraints and their minimum and maximum cardinality (i.e., the number of variables that appear in the constraints), and the total number of possible and consistent configurations. The reported number of discrete and binary variables for the feature models differ by one because the root feature must always be selected and its domain is reduced to a single value after enforcing node consistency in the CSP. In the Dell laptops model, the enumeration of the consistent solutions was stopped after verifying that there exist more than one million consistent configurations.

Contrary to the Facebook privacy settings problem, in this case there is no data set available with configuration records for the models. The complete configurations used for training and testing the algorithms had to be generated artificially. A data set of interesting configurations for evaluating the algorithms' performance was created as follows.

First, an estimation of the best attainable performance was computed (as described earlier in Section 4.1) for each consistent solution on the smaller models, and a random sample of 1,000 consistent solutions on the Dell laptops model. The dimensions of the constraint-based benchmark problems make it unfeasible to use brute-force search to compute the shortest sequence of questions that could be asked to reproduce each complete configuration, so an approximation

[6] `http://www.splot-research.org`

[7] `http://www.jku.at/isse/content/e139529/e126342/e126343`

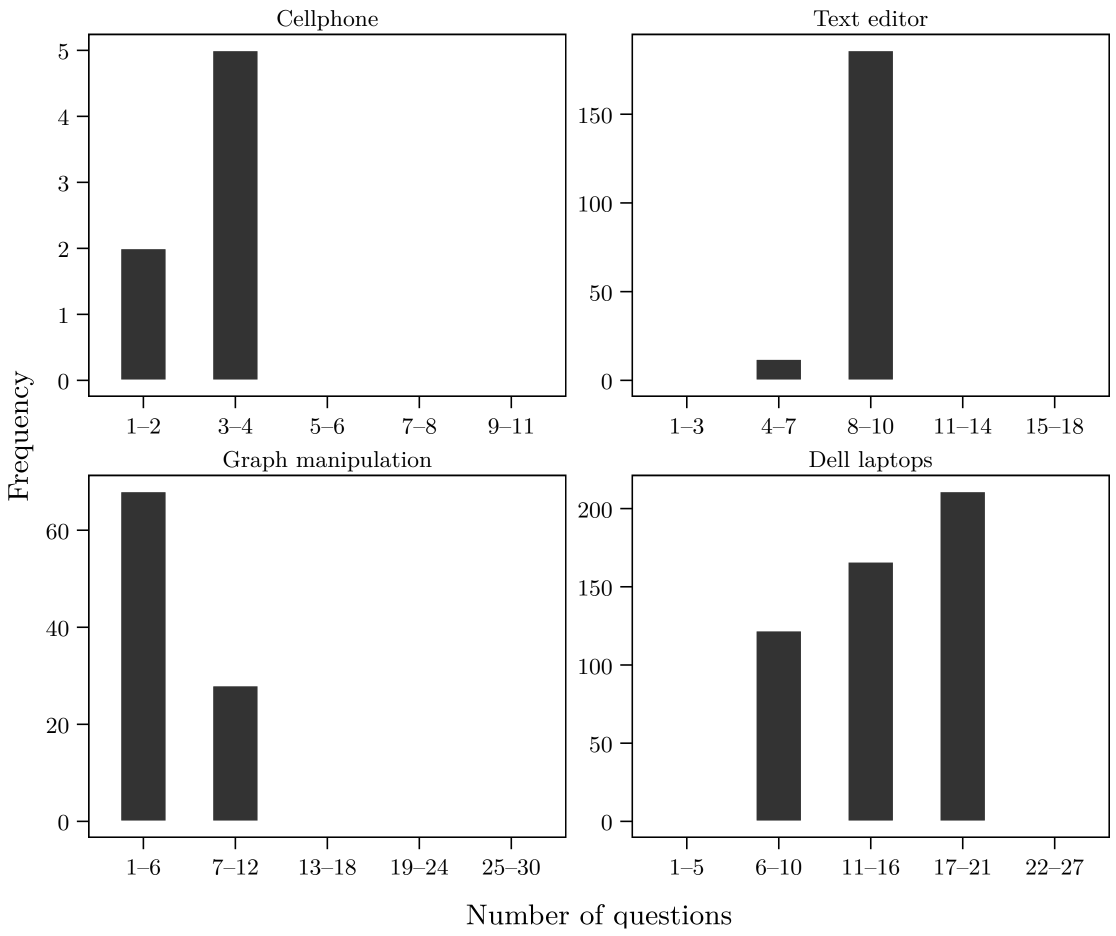

Figure 4.5: Histograms of the best attainable performance (i.e., the minimum possible number of questions) on the constraint-based benchmark problems.

had to be employed. The solution of the optimization problem associated with determining the optimal performance baseline on each configuration was solved using a GA with the same selection, crossover and mutation operators described in Section 3.3.3.2; but allocating a total number of 100 generations instead of 50 to facilitate finding the best possible results (cf. Appendix A).

Finally, 50% of the evaluated configurations on each model were selected as the configuration sample to evaluate the performance of the algorithms. The selected configurations correspond to the ones that can be reproduced with the fewest questions—i.e., the configurations associated with the users that would benefit the most from using the techniques proposed in the thesis. Figure 4.5 illustrates the best attainable performance (i.e., the minimum possible number of questions) on the selected configurations. The evaluation samples of the cellphone, text editor, graph manipulation and Dell laptops model have 7, 198, 96, and 500 configurations, respectively. These results are considered as the performance baseline to compare the algorithms in the experiments reported in the next section.

4.3.2 Performance Results

The study of the performance of the proposed solution methods based on NFQ and the GA on the constraint-based configuration models follows the experimental setup discussed in Section 4.1. The algorithms were used to construct policies for optimizing the user interaction in the different configuration models. Each algorithms' policy performance was evaluated with respect to the number of additional questions asked compared to the best attainable performance when reproducing the group of complete configurations shown in the previous section. The same data set of configurations was used to compute the conditional probabilities of the user's responses given each partial configuration during the simulated training episodes. No attempt was made to evaluate the generalization of the algorithms to unseen configurations during training, given that the testing data set is intentionally biased towards the configurations that require asking the fewest questions. As it was done in the experiments with the rule-based models, the algorithms were run 30 times with different random seeds and the parameters summarized in Appendix A. The reported results also include the performance obtained from asking the questions following a random order.

Table 4.4 presents the mean and standard deviation of the error values obtained by the different solution methods on the four studied constraint-based configuration models. Additionally, Figure 4.6 shows box plots of the distributions of the errors achieved by each algorithm on each model. In general, the results show that the proposed solution methods work effectively with constraint-based configuration models, obtaining results that are close to the best attainable performance on the tested configurations. Both the GA and NFQ require on average a number of questions that is roughly one question away from the optimal performance on the small models, and three questions away on the Dell laptops problem. The solution method based on the GA exhibits the best performance in terms of the mean error values across all four studied models, and it achieves the most consistent results on the testing sample according to the spread of the

Table 4.4: Error values obtained on the constraint-based benchmark problems.

Problem	GA		NFQ		Random	
	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
Cellphone	0.80	0.27	1.10	0.39	1.38	0.28
Text editor	0.96	0.85	1.20	0.54	1.51	0.27
Graph manipulation	1.23	0.21	1.65	0.33	1.73	0.43
Dell laptops	3.13	2.08	3.88	1.85	5.24	3.99

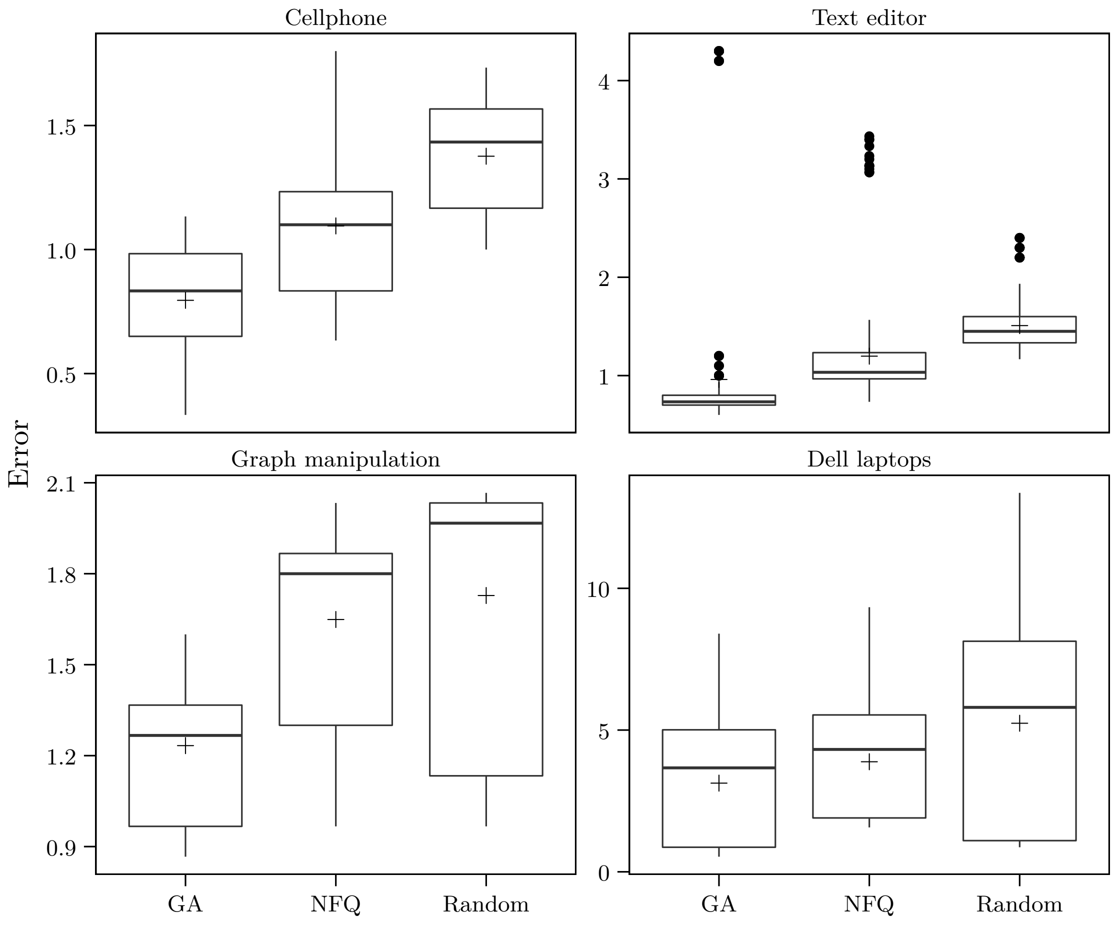

Figure 4.6: Box plots of the results on the constraint-based benchmark problems. The crosses represent the mean error values.

Table 4.5: Statistical analysis of the results on the constraint-based benchmark problems. The p-values of the post-hoc tests have been adjusted for multiple comparisons.

Problem	Pre-test	Post-tests		
		NFQ vs Random	GA vs Random	GA vs NFQ
Cellphone	0.00584	0.103	0.0668	0.325
Text editor	$< 2e{-16}$	$< 2e{-16}$	$< 2e{-16}$	$< 2e{-16}$
Graph manipulation	$7.05e{-10}$	0.000162	$8.84e{-12}$	$5.24e{-12}$
Dell laptops	$< 2e{-16}$	$< 2e{-16}$	$< 2e{-16}$	$< 2e{-16}$

error distributions (particularly in the graph manipulation model).

In line with what was observed earlier in the Facebook privacy settings model, asking the questions in an arbitrary order also seems to produce noticeable good results in the these problems. Motivated by these results, a statistical analysis of the error distributions was also performed in this case in order to ensure that there is enough evidence supporting the claim that the GA and NFQ solution methods produce better results than following a random order for asking the questions on the studied constraint-based models. Table 4.5 summarizes the results of the analysis on each model. As it was done before, a Friedman test [76] was applied first to decide whether statistically significant differences were observed among the three algorithms. If the null hypothesis was rejected, then a series of post-hoc Wilcoxon signed rank test [77] were applied to look for statistically significant differences between the specific pairs of algorithms. The p-values of the post-hoc tests were adjusted for multiple comparisons using the Bonferroni correction [78]. In all cases except the post-hoc tests on the results on the cellphone model, the null hypotheses were rejected at the 1% significance level. Hence, the collected data supports the conclusions that the GA-based solution method obtains better results than NFQ, and both perform better than selecting an arbitrary order for asking the questions—except maybe on small configuration models where their performance is indistinguishable.

As it was explained in Section 3.1, a configuration process using a constraint-based model relies on enforcing a certain level of consistency in the corresponding CSP after every configuration decision to propagate the implications of one variable assignment onto the unassigned variables (see Figure 3.2). The experiments reported in this section used generalized arc-consistency as the constraint-propagation mechanism during the simulated training episodes, given that it provides a good balance between the computational cost of the algorithm and its inference power.

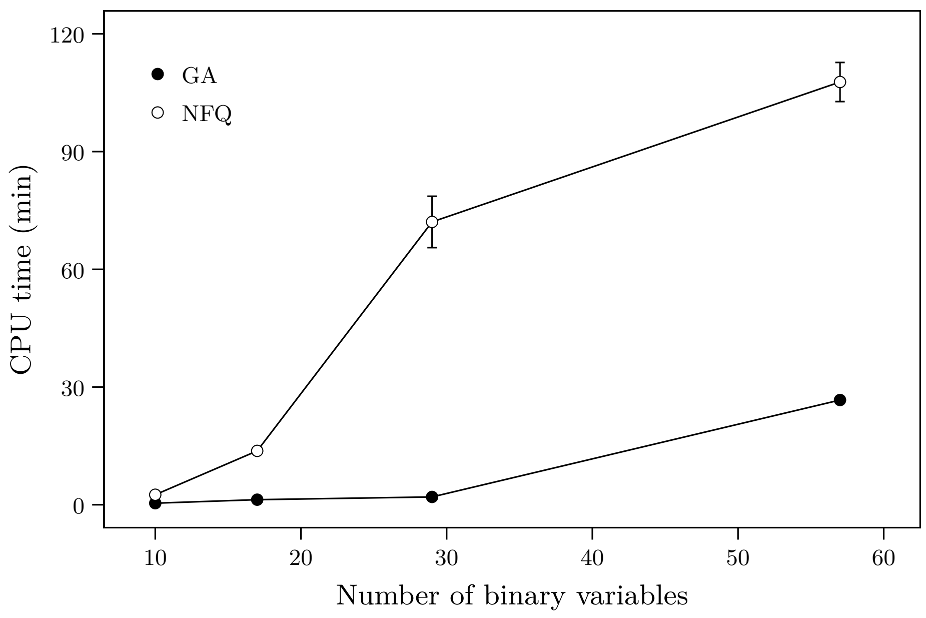

Figure 4.7: Scalability results on the constraint-based benchmark problems.

Nonetheless, since generalized arc-consistency is a local constraint-propagation mechanism, it does not necessarily detect all inconsistencies and it is possible to reach dead-ends in the simulated configuration processes. In these experiments, the training episodes that reached a dead-end were simply discarded, given that it was observed that they had a minimal impact on the studied constraint-based configuration models.

4.3.3 Scalability Results

The running time of NFQ and the GA was studied empirically considering the CPU time elapsed while searching for the optimal policy on each one of the constraint-based configuration models listed in Table 4.3. This information was recorded during the 30 independent runs executed for the performance experiments reported in the previous section and it is shown in Figure 4.7. The data points in the figure correspond to the mean CPU time observed in the 30 runs of each algorithm. The plot also include error bars for the standard error of the mean in the cases where they were visible with the selected scale.

The scalability results indicate that the GA solution method is more efficient than NFQ for the calculation of optimal configuration processes with larger number of variables. The GA was found to require a greater number of simulated episodes to obtain sufficiently good policies on the studied problems (see the parameters in Appendix A); however, the operations performed to learn the optimal policy are considerably more efficient than the NFQ training procedure. NFQ relies on a supervised training algorithm for estimating the parameters of the neural network representing the Q action-value function considering data collected from all previously simulated episodes, while the GA perform simpler crossover and mutation operations on a more compact policy representation (see Sections 3.3.2 and 3.3.3).

4.4 Summary

This chapter reported the results of a group of experiments performed to evaluate the NFQ and GA-based solution methods proposed in this thesis for the optimization of the user interaction in a configuration process. The study focused mainly on two aspects: the scalability of the algorithms compared to the dynamic programming solution methods employed by Hamidi et al. [5], and their ability to work with the more general specification of the configuration model based on CSPs.

An analysis carried out on a group of artificially-generated configuration models evidenced that the proposed techniques overcome the scalability limitations of the framework introduced by Hamidi et al. [5]. The study of the performance of the proposed algorithms on a collection of rule-based and constraint-based configuration models showed that they are able to work effectively with both types of configuration models, obtaining results that were in all cases close to the best attainable performance. In the rule-based configuration model based on the Facebook privacy settings, the performance of GA and NFQ was on average 99.1% and 97.8% optimal, respectively. The performance of GA on the four studied constraint-based models ranged from 77.3% to 89.0% optimal, and it was 71.2–86.6% optimal for NFQ on average. The lowest percentage values correspond to the smallest configuration models, where asking one additional question represents a larger fraction of the entire configuration process. Among the two proposed solution methods, the GA-based technique obtained better results than NFQ on the studied models in terms of the performance and the scalability of the algorithms.

An interesting result was observed in the experiments with both rule-based and constraint-based configuration models. In addition to the GA and NFQ algorithms, the study considered a solution method that corresponds to presenting the questions to the users during the configuration process in an arbitrary order. This random solution method produced noticeable good results in the the studied models: 96.6% optimal results on the Facebook privacy settings problem, and between 66.4% and 83.7% optimal on the constraint-based models. Although these results were confirmed to be statistically significantly worse than the GA and NFQ results, the effect of using the more 'intelligent' techniques may be considered small. For example, the absolute effect (measured as the difference between the mean error values) of following the sequence of questions obtained with the GA compared to the random solution method varied roughly between one and two questions across all the studied configuration models (see Tables 4.2 and 4.4). Nonetheless, notice that this situation is to a great extent an inherent characteristic of the studied configuration problems and not of the proposed solution methods. As an illustration, the absolute effect of the optimal results compared to the random solution method varied between one and five questions. The fact that the random solution method shows a good performance suggests that for many of the tested configurations it is possible to infer the values of a considerable number variables from the user's responses, regardless of the order in which the questions are asked. Also, this comparison only considers the mean error values of the algorithms, but for specific configurations the differences obtained may be larger.

5 Conclusions

This final chapter begins with a summary of the results presented in the thesis, followed by a discussion of the impact of the present work compared to related studies. Lastly, the thesis concludes with an outline of future work directions.

5.1 Summary

This thesis studied the efficient calculation of optimal configuration processes. Its main motivation comes from the earlier work of Hamidi et al. [5], in which the authors proposed a framework for calculating the minimum number of questions asked to the users of a software system in order to configure it to their needs. Hamidi et al. [5] formulated the problem associated with the optimization of the user interaction in a configuration process as an MDP and used classical dynamic programming techniques to solve it. A study of the performance of the techniques showed that the approach faced considerable scalability limitations, working in practice only with a very small number of configuration variables. As a consequence, this thesis was tasked with two objectives. Firstly, to develop and study more efficient and scalable methods to solve the formulated MDP. And secondly, to generalize the proposal of Hamidi et al. [5] in order to make it suitable for a broader class of configuration processes.

Two alternative solution methods were proposed as part of the work carried out in this thesis. The algorithms make use of the reinforcement learning paradigm, which means that they simulate interactions with the environment described by the associated MDP in order to find a policy for guiding the configuration process towards minimal user interaction. The new techniques are able to work with one of two possible configuration model specifications. The first one, comparable to the model used by Hamidi et al. [5], represents the constraints that regulate the interactions between the aspects that can be configured as *if-then* rules. The additional supported configuration model is more flexible and it allows using Boolean predicates to specify the constraints as part of a formulation of the configuration model as a CSP. In both cases, the configuration processes rely on inference mechanisms to avoid asking questions to the users about variables that can be answered automatically from the decisions of other users. The first solution method consists of using the NFQ algorithm to train a neural network representation of the policy that computes an estimate of the expected number of questions that could be skipped by asking one particular question. The second solution method represents the policy simply as a permutation of the configuration variables which dictates the order that must be followed for asking their associated questions to the users. In this latter case, a GA is used to find a permutation that maximizes the expected number of automatically-discovered configuration decisions.

The newly proposed solution methods were evaluated on an experimental study considering artificially-generated configuration models, a case study on a small real-world example related to the privacy settings of the Facebook social network platform, and a number of publicly-available configuration models from different domains. The analysis of the scalability of NFQ and the GA confirmed that they both overcome the limitations of the framework introduced by Hamidi et al. [5], resulting in more suitable algorithms to work with configuration models involving a large number of configuration variables. The study of the performance of the proposed algorithms also showed that they are able to work effectively with both rule-based and constraint-based models, reducing the number of questions asked to the users and obtaining results that were in all cases close to the best attainable performance. Among the proposed techniques, the GA-based approach obtained the best results on the studied models in terms of the performance and running time of the algorithms. The experimental study also uncovered some interesting underlying characteristics of the problem of optimizing the user interaction in a configuration process.

5.2 Contributions

This thesis is framed in the field of artificial intelligence and its application in the area of knowledge-based configuration—also referred to as product configuration or customization. Additionally, the thesis touches on aspects of software engineering because of the relevance of configuration-related topics in that discipline. The most important contributions are the following:

i. The thesis constitutes one of the first attempts to study the problem of optimizing the user interaction in a discrete-variable configuration process, independently of a particular configuration domain. Related works have targeted specific configuration models used in application domains such as software engineering, but the approach discussed in this thesis—especially with the specification of the configuration model as a CSP—is applicable to a wide range of configuration models with discrete variables.

ii. The techniques recommended in the thesis for the efficient calculation of optimal configuration processes offer a global solution of the associated optimization problem, and at the same time are suitable algorithms to work with models involving a large number of configuration variables. Also, the simulation-based approach followed by the solution methods provides a straightforward mechanism to consider previously-existing configuration records in order to adjust the optimized policies to specific user interests. These characteristics constitute attractive features for using the proposed techniques in practical configuration scenarios.

iii. Additionally, the thesis reports potentially valuable experiences in the solution of sequential decision-making problems formulated as MDPs with large state spaces. These experiences may be useful to practitioners facing similar scalability challenges in other problem domains.

5.3 Related Work and Discussion

The problem of optimizing the user interaction in a configuration process, as in the explicit minimization of the number of questions necessary to obtain a complete configuration, has not been widely studied in the literature. The development of techniques to support the users within the scope of a configuration process has concentrated mostly on the design of user interfaces, providing explanations in the face of inconsistent decisions, and the application of recommendation technologies (see e.g. Felfernig et al. [14]). Apart from the framework proposed by Hamidi et al. [5], which is the direct precedent of this thesis, an earlier work by Nöhrer and Egyed [81] introduced a technique (and later a supporting tool [83]) for automatically optimizing the order of the decisions made during the product derivation phase of a software product line.

The approach followed by Nöhrer and Egyed [81] does not attempt to solve the associated optimization problem globally, it relies instead on a local heuristic that guides the users during the decision-making process. The heuristic consists of calculating a 'gain' score for each unanswered question after every user decision and selecting the question associated with the maximum score as the next question to be presented the user. The gain score computed for each question reflects its potential of being automatically answered as well as its impact on reducing the possible choices of the remaining questions. The configuration model is represented as a decision model involving a number of questions with discrete choices. Two types of relations between the questions were considered as part of the initial proposal: 'constraint' relations where the choice of one question restricts the choices of another question, and 'relevancy' relations in which the choice of one question makes another question relevant or irrelevant (the model was extended later to admit arbitrary relations in conjunctive normal form [83]). The configuration model is expressed as a Boolean satisfiability problem for reasoning purposes.

Compared to the previous work of Hamidi et al. [5] and Nöhrer and Egyed [81], this thesis considers a more general interpretation of the configuration process that is independent of a particular configuration domain—i.e., it is not necessarily related to the configuration of software systems or the derivation of a software product from a software product line. As discussed in Section 2.1.2, the formulation of the configuration model as a CSP is a well-studied representation that is more flexible than a rule-based specification and it is also capable of expressing the feature models and decision models frequently used in software product line engineering. In relation to the work of Nöhrer and Egyed [81], the most significant difference is that the solution methods studied in this thesis tackle the optimization problem associated with the minimization of the user interaction globally, instead of looking for a local solution. This approach has two important consequences. First, each question asked to the users is the result of evaluating all the possible ramifications of their future decisions and not only the next step in the configuration process. Second, all the expensive computations for deciding the order in which the questions must be asked occur separately from the interactive configuration processes where the users intervene. Therefore, the users do not have to wait for the decisions about the next question to be computed in real time, ensuring a prompt response of the configuration system. Additionally, the solution methods studied in this thesis allow the consideration of the user's configuration interests as part of the optimized policies, by relying on the simulation of configuration episodes to solve the associated optimization problem.

5.4 Future Work

The work presented in this thesis motivates different avenues for future research. This work can include extending the empirical experimentation with the NFQ and GA-based solution methods to consider configuration models with a larger number of variables. The S.P.L.O.T. feature model repository [82] may constitute a valuable resource for conducting the aforementioned study, given that at this time it includes eight models with more than 100 variables and as many as 366 variables in the largest model available. In addition, a deeper examination of the influence of the different algorithm parameters could lead to a better understanding of the capabilities of the proposed solutions. On a broader scope, it becomes imperative to conduct a careful analysis of the practical impact of using techniques for optimizing the user interaction in a configuration process compared to following an arbitrary order for asking the questions, in order to identify the types of configuration models that would benefit the most from using such techniques.

Bibliography

[1] B. Joseph Pine II. *Mass Customization: The New Frontier in Business Competition.* Harvard Business School Press, 1993. (Cited on page 1.)

[2] Mitchell M. Tseng and Jianxin Jiao. *Handbook of Industrial Engineering*, chapter Mass Customization, pages 684–709. John Wiley & Sons, 2007. (Cited on page 1.)

[3] Markus Stumptner. An overview of knowledge-based configuration. *AI Communications*, 10(2):111–125, 1997. (Cited on page 1.)

[4] Daniel Sabin and Rainer Weigel. Product configuration frameworks – A survey. *IEEE Intelligent Systems*, 13(4):42–49, 1998. (Cited on page 1.)

[5] Saeideh Hamidi, Periklis Andritsos, and Sotirios Liaskos. Constructing adaptive configuration dialogs using crowd data. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, pages 485–490, 2014. (Cited on pages 2, 3, 4, 40, 43, 45, 46, 47, 48, 52, 59, 61, 62, 63, and 64.)

[6] Wendy E. Mackay. Triggers and barriers to customizing software. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 153–160, 1991. (Cited on page 2.)

[7] Sotirios Liaskos, Alexei Lapouchnian, Yiqiao Wang, Yijun Yu, and Steve Easterbrook. Configuring common personal software: A requirements-driven approach. In *Proceedings of the 13th IEEE International Conference on Requirements Engineering*, pages 9–18, 2005. (Cited on page 2.)

[8] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, 1993. (Cited on page 3.)

[9] Richard E. Bellman. *Dynamic Programming.* Princeton University Press, 1957. (Cited on pages 3, 4, and 14.)

[10] Saeideh Hamidi. Automating software customization via crowdsourcing using association rule mining and Markov decision processes. Master's thesis, York University, Ontario, Canada, 2014. (Cited on pages 3, 43, 46, and 47.)

[11] Edward Tsang. *Foundations of Constraint Satisfaction.* Academic Press, 1993. (Cited on pages 4 and 8.)

[12] Ronald A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, 1960. (Cited on page 4.)

[13] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998. (Cited on pages 5, 6, and 15.)

[14] Alexander Felfernig, Lothar Hotz, Claire Bagley, and Juha Tiihonen. *Knowledge-based Configuration: From Research to Business Cases*. Morgan Kaufmann Publishers, 2014. (Cited on pages 6 and 63.)

[15] Marco Wiering and Martijn van Otterlo, editors. *Reinforcement Learning: State-of-the-Art*, volume 12 of *Adaptation, Learning, and Optimization*. Springer, 2012. (Cited on page 6.)

[16] John McDermott. R1: A rule-based configurer of computer systems. *Artificial Intelligence*, 19(1):39–88, 1982. (Cited on page 6.)

[17] Elliot Soloway, Judy Bachant, and Keith Jensen. Assessing the maintainability of XCON-in-RIME: Coping with the problems of a very large rule-base. In *Proceedings of the 6th National Conference on Artificial Intelligence*, pages 824–829, 1987. (Cited on page 7.)

[18] Naresh K. Malhotra. Information load and consumer decision making. *Journal of Consumer Research*, 8(4):419–430, 1982. (Cited on page 7.)

[19] Jacob Jacoby, Donald E. Speller, and Carol Kohn Berning. Brand choice behavior as a function of information load: Replication and extension. *Journal of Consumer Research*, 1 (1):33–42, 1974. (Cited on page 7.)

[20] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors. *Recommender Systems Handbook*. Springer, 2011. (Cited on page 7.)

[21] Sanjay Mittal and Felix Frayman. Towards a generic model of configuraton tasks. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 1395–1401, 1989. (Cited on page 8.)

[22] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*, chapter Constraint Satisfaction Problems, pages 137–160. Prentice Hall, 2nd edition, 2002. (Cited on page 8.)

[23] Eugene C. Freuder. A sufficient condition for backtrack-free search. *Journal of the ACM*, 29(1):24–32, 1982. (Cited on page 9.)

[24] Alan K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977. (Cited on page 9.)

[25] Romuald Debruyne and Christian Bessiere. Domain filtering consistencies. *Journal of Artificial Intelligence Research*, 14:205–230, 2001. (Cited on page 9.)

[26] Jérôme Amilhastre, Hélène Fargier, and Pierre Marquis. Consistency restoration and explanations in dynamic CSPs – Application to configuration. *Artificial Intelligence*, 135 (1–2):199–234, 2002. (Cited on page 9.)

[27] Paul Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2001. (Cited on page 10.)

[28] Lianping Chen, Muhammad Ali Babar, and Nour Ali. Variability management in software product lines: A systematic review. In *Proceedings of the 13th International Software Product Line Conference*, pages 81–90, 2009. (Cited on page 10.)

[29] Thomas von der Maßen and Horst Lichter. Determining the variation degree of feature models. In *Proceedings of the 9th International Software Product Line Conference*, volume 3714 of *Lecture Notes in Computer Science*, pages 82–88. Springer, 2005. (Cited on pages 11 and 52.)

[30] Kyo Kang, Sholom Cohen, James Hess, William Novak, and A. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pennsylvania, United States, 1990. (Cited on page 10.)

[31] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. Staged configuration using feature models. In Robert L. Nord, editor, *Software Product Lines*, volume 3154 of *Lecture Notes in Computer Science*, pages 266–283. Springer, 2004. (Cited on page 10.)

[32] David Benavides, Pablo Trinidad, and Antonio Ruiz-Cortés. Automated reasoning on feature models. In Oscar Pastor and João Falcão e Cunha, editors, *Advanced Information Systems Engineering*, volume 3520 of *Lecture Notes in Computer Science*, pages 491–503. Springer, 2005. (Cited on page 10.)

[33] David Benavides, Antonio Ruiz-Cortés, and Pablo Trinidad. Coping with automatic reasoning on software product lines. In *Proceedings of the 2nd Groningen Workshop on Software Variability Management*, 2004. (Cited on page 11.)

[34] Klaus Schmid, Rick Rabiser, and Paul Grünbacher. A comparison of decision modeling approaches in product lines. In *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems*, pages 119–126, 2011. (Cited on page 11.)

[35] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 1994. (Cited on page 12.)

[36] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*, chapter Making Complex Decisions, pages 613–648. Prentice Hall, 2nd edition, 2002. (Cited on page 12.)

[37] Christopher J. C. H. Watkins. *Learning from Delayed Rewards*. Phd thesis, King's College, Cambridge, UK, 1989. (Cited on page 17.)

[38] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992. (Cited on page 17.)

[39] Martin Riedmiller. Neural fitted Q iteration – First experiences with a data efficient neural reinforcement learning method. In *Proceedings of the 16th European Conference on Machine Learning*, pages 317–328, 2005. (Cited on pages 18 and 19.)

[40] Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556, 2005. (Cited on page 18.)

[41] Sascha Lange, Thomas Gabel, and Martin Riedmiller. Batch reinforcement learning. In Marco Wiering and Martijn van Otterlo, editors, *Reinforcement Learning: State-of-the-Art*, volume 12 of *Adaptation, Learning, and Optimization*. Springer, 2012. (Cited on page 18.)

[42] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 586–591, 1993. (Cited on pages 18 and 32.)

[43] David E. Moriarty, Alan C. Schultz, and John J. Grefenstette. Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research*, 11:241–276, 1999. (Cited on page 19.)

[44] Shimon Whiteson. Evolutionary computation for reinforcement learning. In Marco Wiering and Martijn van Otterlo, editors, *Reinforcement Learning: State-of-the-Art*, volume 12 of *Adaptation, Learning, and Optimization*. Springer, 2012. (Cited on page 19.)

[45] John H. Holland. *Adaptation In Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. University of Michigan Press, 1975. (Cited on page 19.)

[46] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989. (Cited on pages 19 and 36.)

[47] Shimon Whiteson, Matthew E. Taylor, and Peter Stone. Critical factors in the empirical performance of temporal difference and evolutionary methods for reinforcement learning. *Journal of Autonomous Agents and Multi-Agent Systems*, 21(1):1–27, 2010. (Cited on page 21.)

[48] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995. (Cited on page 30.)

[49] Nees Jan van Eck and Michiel van Wezel. Application of reinforcement learning to the game of Othello. *Computers & Operations Research*, 35(6):1999–2017, 2008. (Cited on pages 30 and 32.)

[50] Leslie P. Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996. (Cited on page 32.)

[51] Martin Riedmiller. 10 steps and some tricks to set up neural reinforcement controllers. In *Neural Networks: Tricks of the Trade*, pages 735–757. Springer, 2nd edition, 2012. (Cited on page 32.)

[52] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. (Cited on page 32.)

[53] Norman R. Draper and Harry Smith. *Applied Regression Analysis*, chapter "Dummy" Variables, pages 299–325. John Wiley & Sons, 1998. (Cited on page 33.)

[54] Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural Networks: Tricks of the Trade*, pages 9–48. Springer, 2nd edition, 2012. (Cited on page 34.)

[55] Kenneth A. De Jong. *Evolutionary Computation: A Unified Approach*. MIT Press, 2006. (Cited on page 36.)

[56] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003. (Cited on page 36.)

[57] Malti Baghel, Shikha Agrawal, and Sanjay Silakari. Survey of metaheuristic algorithms for combinatorial optimization. *International Journal of Computer Applications*, 58(19):21–31, 2012. (Cited on page 36.)

[58] Stephen Chen and Stephen F. Smith. Commonality and genetic algorithms. Technical report, Robotics Institute, Carnegie Mellon University, Pennsylvania, USA, 1996. (Cited on page 36.)

[59] Stephen Chen and Stephen F. Smith. Improving genetic algorithms by search space reduction (with applications to flow shop scheduling). In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 135–140, 1999. (Cited on page 36.)

[60] Lawrence Davis. Applying adaptive algorithms to epistatic domains. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 162–164, 1985. (Cited on page 36.)

[61] Darrell Whitley, Timothy Starkweather, and D'Ann Fuquay. Scheduling problems and traveling salesman: The genetic edge recombination operator. In *International Conference on Genetic Algorithms*, pages 133–140, 1989. (Cited on page 37.)

[62] Christian Bierwirth, Dirk C. Mattfeld, and Herbert Kopfer. On permutation representations for scheduling problems. In *Parallel Problem Solving from Nature – PPSN IV*, pages 310–318. Springer, 1996. (Cited on page 37.)

[63] Simon French. *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop*. John Wiley & Sons, 1982. (Cited on page 37.)

[64] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 1–12, 2000. (Cited on page 44.)

[65] Christian Borgelt. An implementation of the FP-growth algorithm. In *Proceedings of the 1st International Workshop on Open Source Data Mining: Frequent Pattern Mining Implementations*, pages 1–5, 2005. (Cited on page 44.)

[66] Christian Borgelt. Frequent item set mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(2):437–456, 2012. (Cited on page 44.)

[67] danah m. boyd and Nicole B. Ellison. Social network sites: Definition, history, and scholarship. *Journal of Computer-Mediated Communication*, 13(1):210–230, 2007. (Cited on page 46.)

[68] Cliff Lampe, Nicole B. Ellison, and Charles Steinfield. Changes in use and perception of Facebook. In *Proceedings of the 2008 ACM Conference on Computer Supported Cooperative Work*, pages 721–730, 2008. (Cited on page 46.)

[69] Ralph Gross and Alessandro Acquisti. Information revelation and privacy in online social networks. In *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*, pages 71–80, 2005. (Cited on page 46.)

[70] Alessandro Acquisti and Ralph Gross. Imagined communities: Awareness, information sharing, and privacy on the Facebook. In George Danezis and Philippe Golle, editors, *Privacy Enhancing Technologies*, volume 4258 of *Lecture Notes in Computer Science*, pages 36–58. Springer, 2006. (Cited on page 46.)

[71] Katherine Strater and Heather Richter Lipford. Strategies and struggles with privacy in an online social networking community. In *Proceedings of the 22nd British HCI Group Annual Conference on People and Computers: Culture, Creativity, Interaction*, pages 111–119, 2008. (Cited on page 46.)

[72] Bernhard Debatin, Jennette P. Lovejoy, Ann-Kathrin Horn, and Brittany N. Hughes. Facebook and online privacy: Attitudes, behaviors, and unintended consequences. *Journal of Computer-Mediated Communication*, 15(1):83–108, 2009. (Cited on page 46.)

[73] danah m. boyd and Eszter Hargittai. Facebook privacy settings: Who cares? *First Monday*, 15(8), 2010. (Cited on page 46.)

[74] Yabing Liu, Krishna P. Gummadi, Balachander Krishnamurthy, and Alan Mislove. Analyzing Facebook privacy settings: User expectations vs. reality. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, pages 61–70, 2011. (Cited on page 46.)

[75] Maritza Johnson, Serge Egelman, and Steven M. Bellovin. Facebook and privacy: It's complicated. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, pages 1–15, 2012. (Cited on page 46.)

[76] Milton Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701, 1937. (Cited on pages 51 and 57.)

[77] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6): 80–83, 1945. (Cited on pages 51 and 57.)

[78] Olive Jean Dunn. Multiple comparisons among means. *Journal of the American Statistical Association*, 56(293):52–64, 1961. (Cited on pages 51 and 57.)

[79] Krzysztof Czarnecki, Simon Helsen, and Eisenecker Ulrich. Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice*, 10: 7–29, 2005. (Cited on page 52.)

[80] Hong Mei, Wei Zhang, and Fang Gu. A feature oriented approach to modeling and reusing requirements of software product lines. In *Proceedings of the 27th Annual International Computer Software and Applications Conference*, pages 250–256, 2003. (Cited on page 52.)

[81] Alexander Nöhrer and Alexander Egyed. Optimizing user guidance during decision-making. In *Proceedings of the 15th International Software Product Line Conference*, pages 25–34, 2011. (Cited on pages 52, 63, and 64.)

[82] Marcilio Mendonca, Moises Branco, and Donald Cowan. S.P.L.O.T.: Software product lines online tools. In *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications*, pages 761–762, 2009. (Cited on pages 53 and 65.)

[83] Alexander Nöhrer and Alexander Egyed. C2O configurator: A tool for guided decision-making. *Automated Software Engineering*, 20(2):265–296, 2013. (Cited on pages 53, 63, and 64.)

A Algorithm Parameters

The following table summarizes the parameters of the algorithms used in the experiments described in Chapter 4. Each parameter value was determined empirically based on experiences reported in the literature and limited preliminary experimentation.

Algoritm	Parameter	Value
Value iteration	Maximum number of iterations	1,000
	Termination criterion	$\max_{s \in S} \lvert V_{k+1}(s) - V_k(s) \rvert < 0.01$
Q-learning	Total number of simulated episodes	1,000
	Exploration	ϵ-greedy with $\epsilon = 0.1$
	Learning rate	$\alpha = 0.3$
NFQ	Total number of simulated episodes	1,000
	Exploration	ϵ-greedy with $\epsilon = 0.1$
	Learning batch size (i.e., number of episodes simulated before each learning phase)	10
	Iterations of the NFQ main loop (i.e., maximum k value in Figure 2.3)	1
	Maximum number of RPROP training epochs	300
	RPROP termination criterion	Mean squared error less than 0.001
GA	Total number of simulated episodes	10,000
	Number of episodes simulated to evaluate each policy	10
	Total number of generations (i.e., maximum g value in Figure 2.4)	50
	Population size	$N = 20$
	Selection method	Binary tournament selection
	Crossover probability	1.0
	Mutation probability	0.2