Part II

Preference Reasoning

In the previous part, we presented a metamodel that allows representing preferences provided by users. The vocabulary that this metamodel provides was built based on patterns and expressions extracted from a study of how humans express preferences, so it models high-level preferences adopted by people. While proposing this preference metamodel, we were not concerned with the feasibility or existence of a way of reasoning about the set of preferences that can be modelled with our metamodel. We have focused on developing a metamodel that shows which kind of information should be ideally used as input for algorithms to reason about preferences, in scenarios where users state their preferences, i.e. preferences are not captured implicitly. So, in this part, starting from our preference metamodel, we tackle the problem of reasoning about preferences in order to choose one option of a set of available.

We begin with a literature review of work in the context of reasoning about preferences, presented in Chapter 5. As taking into account individual preferences is relevant for different purposes, such as content personalisation, decision support systems and automation of user tasks, this problem was investigated in different research areas within computer science, such as artificial intelligence and databases. To capture similarities and differences between these different works, we present them in the form of a systematic review, thus enabling their comparison. We identify issues left unaddressed.

Next, we propose in Chapter 6 a technique for reasoning about preferences and making decisions, addressing these identified issues. Preferences that our technique receives as input are expressed in a language that is built based on a restricted version of our preference metamodel. In addition, this technique uses principles of human decision making, in order to deal with trade-off situations. We also compare our technique with existing work and evaluate it with information collected in our study of how humans express preferences.

5
A Systematic Review of Reasoning about Preferences

Preferences are involved with many problems of computer science. People who have complex decisions to be made can be assisted with decision support systems, and such decisions must be made based on preferences of a particular individual. Information provided in the web is nowadays massive, and even with search tools it may be hard to find the information that is needed in a short time as it is common to obtain lots of websites as results of a search. These different scenarios lead to the emergence of research on preference in many research areas, such as artificial intelligence (AI) and databases. As a consequence, there is a large number of approaches proposed. In addition, new approaches are typically compared to existing ones in the same research area; however, there is limited investigation of the relationship of approaches across different areas.

Work on preferences investigates different problems, often associated with one of these topics: preference elicitation (how to obtain preferences from users), preference representation (how to model preferences), and preference reasoning (how to use preferences for making decisions, ordering search results, and other purposes). In this chapter, we present a systematic review[1] of approaches that propose mechanisms to *reason* about preferences, which include algorithms and algebras. The goal of this comparison is to obtain a clear overview of the approaches and find out how existing approaches differ with respect to preference reasoning. As this is a topic investigated in many research areas, our aim is to provide a comprehensive review of these approaches to identify the kinds of issues they are addressing, which structures they use as input, how they work and their limitations. We begin by describing the evaluation framework adopted to analyse each investigated approach in Section 5.1. Approaches are then presented from Section 5.3 to Section 5.7, organised in the following way.

We first describe a background on Multi-Attribute Utility Theory (Section 5.2), which is one of the oldest approaches to help decision makers to make choices, and inspired much of the existing work on reasoning about preferences in different research areas. It established definitions that were adopted in most of the

[1]We use the term *systematic* because this literature review was performed in a systematic way. Nevertheless, the review is not a *systematic review* in the sense of social sciences (Petticrew and Roberts 2006).

presented approaches. The first group of approaches we present consists of those based on *utility functions* (a function that represents preference quantitatively), which propose specific models to represent preferences and forms to derive utility functions from these models, which are supposed to be more natural for users (Section 5.3). Section 5.4 describes approaches that extend Constraint Satisfaction Problems (CSPs) to incorporate *soft constraints*, i.e. constraints that can remain unsatisfied. Utility functions and CSPs are two classical approaches for dealing with preferences and making decisions, and approaches detailed in Section 5.5 take another direction: they propose new *graphical structures* to represent and reason about preferences. Databases are another research area that has been investigating preferences, and works in this area are presented in Section 5.6. These works propose *extensions of query languages* to incorporate preferences and algorithms to provide query results taking the specified preferences into account. Finally, more recently, researchers from argumentation in AI investigated the explicit use of preferences in *argumentation frameworks* to make decisions, which is discussed in Section 5.7. After presenting all these approaches, we further discuss them and show how they are related in Section 5.8, followed by Section 5.9, which concludes this chapter.

5.1 Review Method

Our comparison of approaches to reasoning about preferences follows an evaluation framework composed of seven different criteria, which are described below. Criterion that is not applicable for an approach is omitted in its corresponding section.

Goals. Many of the approaches have the goal of answering a user question that is related to preferences and decision making (see next criterion), so they propose preference representation models and/or algorithms to answer those questions. Other approaches start from an existing (lower-level) preference representation model, which has associated algorithms, and propose means for transforming high-level preference models into low-level ones. Therefore, in these criterion we classify approaches according to the following goals: (i) proposition of *preference representation models*; (ii) proposition of *algorithms*, *algebras* or *semantics* to answer preference-related questions or interpret preference constructions; and (iii) proposition of X *to* Y *transformation*, i.e. transform preferences represented in a model X to preferences represented in a model Y.

Questions. *Which are the preference-related questions addressed by the proposed approach?* As stated before, many of the approaches aim to answer preference-related questions. So, in cases that the approach has this goal, we show which are the questions addressed by each approach. The types of questions are detailed below, from which the first three have been defined by Boutilier et al. (Boutilier et al. 2004).

- *Outcome optimisation* — which is the optimal outcome according to a given preference structure?
- *Dominance queries* — is an outcome o preferred to another outcome o'? If so, it is said that o *dominates* o'.
- *Ordering queries* — is an outcome o' not preferred to another outcome o? If so, it is said that o is *consistently orderable* over o' with respect to the given preference structure.
- *Non-dominated outcomes* — which are the non-dominated outcomes according to a given preference structure?

Input. *What is the input required by the approach?* If the approach relies on a particular structure, specific for the approach, we provide details. Some of these structures were already introduced in Chapter 4, such as CP-nets.

Interpretation. *Which is the interpretation adopted for preference statements?* Most of the approaches use preference statements (or a particular representation of it) as input, but they can be interpreted in different ways: (i) *Ceteris paribus* (Hansson 1996) — meaning "all else being equal", i.e. if one states *"I prefer value x to value x' with regard to attribute X,"* it means that this preference can be considered only when the values of all other attributes are equal; (ii) *Not ceteris paribus* — meaning, when a preference is provided, it is applied to any context. For example, if one states that "lowest" price is a preference for cars, any cheaper car is preferred to any more expensive car.

How it works. An explanation of how each approach works is provided but, as our goal is not to give an extensive description of the involved algorithms and all their steps, we give only a broad understanding of them.

Complexity. *Which is the complexity of the proposed algorithm?* Some approaches do not provide complexity analysis, so this aspect is omitted in their sections.

Limitations. *Which are the limitations of the proposed approach?* Presented approaches may have limitations in different directions, such as having algorithms that takes exponential time to be executed or preferences expressed in a way that are hard to elicit. In Chapter 4, which focused on preference representation, we detailed the kinds of preferences addressed by most of the approaches described here and their particular representation models. Therefore, the lack of expressivity of the approaches will not be reported in this chapter, as it was discussed previously.

5.2 Background on Multi-Attribute Utility Theory

Multi-Attribute Utility Theory (MAUT) is a set of methods designed to handle decision problems involving multiple objectives and trade-offs. According to Dyer (Dyer 2005), MAUT has become synonymous in the view of many scholars with the theory proposed by Keeney and Raiffa (Keeney and Raiffa 1976), which emphasised the use of multi-attribute preference models based on the theories of von Neumann and Morgenstern (von Neumann and Morgenstern 1944), who presented a set of axioms about preferences and utilities such that any decision maker satisfying these axioms has a utility function.

MAUT is concerned with the valuation of the *outcomes* of an *option* of a decision maker. A problem is described in terms of *attributes* $X = \{x_1, ..., x_n\}$, and each of them can have a value assigned according to their respective domains $D_1, ..., D_n$, which establish the range of possible values of an attribute. The set of *all possible outcomes* is the cartesian product of attribute domains $D_1 \times ... \times D_n$. *Feasible outcomes* are a subset of all possible outcomes, and are those that consist of valid combinations of attribute values. A (cardinal) utility function is a function that maps outcomes to a real value that represents the preference for each outcome. The higher is the value, the more preferred the outcome is. In Table 5.1, we give the definition of the three main terms (option, outcome and attribute) adopted in MAUT and decision making, which were already introduced above, together with alternative terminology, as different approaches may use different terms. In some cases, options lead to only one possible outcome, and in such cases the term option and outcome are used as synonyms, for example choosing a product A to be bought (option) leads to the single consequence of buying this product (outcome), therefore A can be used to refer to both the option and the outcome.

Keeney and Raiffa (Keeney and Raiffa 1976) discuss decisions that can be made under *certainty* or *uncertainty*. The first refers to situations in which the outcome, represented by multiple attributes, of an option is known. In these scenarios, the main issue is to resolve trade-offs among preferences, which typically conflict because in general they cannot be maximised at the same time. For instance,

Term	Definition	Alternative Terminology
Options	Elements of the same conceptual class, which is the subject of the decision making process	Actions, Alternatives, Record, Tuples
Outcomes	Results obtained by choosing an option	Payoffs, Consequences
Attribute	Characteristics used to describe options	Variables, Features

Table 5.1: Main terms adopted in decision making.

two preferences might be maximise quality and minimise costs, but lower costs are normally achieved by compromising quality. In the second, the cases in which uncertainty is present, or in other words *risky* situations, there is a probability associated with the possible outcomes of an option. In their work, Keeney and Raiffa consider that this probability is given.

A preference representation function under certainty is referred to as a *value function* v. v associates a real number with each point x in the outcome space, representing the preference structure of the decision maker, provided that

$$\forall\, x', x'' \in D\ x' \sim x'' \Leftrightarrow v(x') = v(x'') \text{ and } x' \succ x'' \Leftrightarrow v(x') > v(x'')$$

$\sim$ and $\succ$ represent *indifference* and *preference*, respectively. Based on v, the goal is to find an outcome $x \in D$ to maximise $v(x)$.

When the decision making is based on assumption of the existence of value functions, two concepts (*preference independence* and *mutual preference independence*) play an important role, as they are related to properties of the value function, which can facilitate its elicitation process. The definition of these two concepts are given below.

Preference Independence. An attribute set Y, where $Y \subset X$, is *preferentially independent* of its complement $\overline{Y}$ if the preference order of two outcomes involving different values assigned to Y does not depend on the fixed values assigned to attributes in $\overline{Y}$. For example, if the laptop colour is preferentially independent of the remaining laptop attributes, and one states that prefers *silver* to *black*, by fixing values of the remaining attributes, any silver laptop is preferred to any black laptop.

Mutual Preference Independence. The attributes $x_1, ..., x_n$ are mutually preferentially independent if every subset Y of X is preferentially independent of its complementary set.

These definitions are used to identify a very important property of value functions: if the mutual preference independence is held, then it can be proved

that the form of the value function is *additive*, and this form of representation is one of the most common approaches for evaluating multiattribute alternatives. In this representation, attribute values are considered independent, and therefore the value of an outcome can be calculated by adding values of individual attributes. This approach is ideal because it is *compact*, as a specific value does not need to be specified for each possible outcome, whose number is exponential on the size of attribute domains. In this thesis, we adopt the term compact as an adjective to approaches that with little information (preference statements, utility values) one can derive a preference order between two outcomes.

When uncertainty is considered, the preference representation function is referred to as *utility function*, which takes into account the probability of outcomes. We now introduce definitions related to decisions under this kind of scenario — many of these definitions come from the formalisation of *expected utility theory* by von Neumann and Morgenstern (von Neumann and Morgenstern 1944).

- A *lottery* is any option with an uncertain outcome. *Examples*: investment, roulette, football game.
- A *probability* of an outcome (of a lottery) is the likelihood that this outcome occurs. *Example*: the probability often is estimated by the historical frequency of the outcome.
- The *probability distribution* of the lottery depicts all possible outcomes in the lottery and their associated probabilities. The probability of any particular outcome is between 0 and 1 and the sum of the probabilities of all possible outcomes is equal to 1.

The definitions of (mutual) preference independence are then extended by considering *uncertainty*: an attribute x_i is said to be **utility independent** of its complementary attributes if preferences over *lotteries* with different values of x_i do not depend on the fixed values of the remaining attributes. Attributes $x_1, x_2, ..., x_n$ are **mutually utility independent** if all proper subsets of these attributes are utility independent of their complementary subsets.

We discuss next goals and limitations of MAUT, which can be related to many of the approaches that will be presented, which rely on utility or value functions.

Goals. A preference representation model in the form of utility functions. The approach presented in MAUT does not focus directly on presenting algorithms to answer questions related to preference reasoning, because once utility functions are correctly defined for a decision maker, typical preference tasks and associated questions, such as comparing and ranking outcomes, can be easily performed and

answered, as it requires only a numerical comparison. The challenge is to reveal the utility function of decision makers.

Limitations. The main issue related to MAUT and other utility function-based approaches is the elicitation process. The process of obtaining the information required to generate a good utility function requires considerable effort on the part of the user. Users may be able to express preferences in other kinds of preference statements, but revealing the utility function underlying such preferences is a challenging task. Moreover, existing techniques for revealing utility functions typically rely on presenting users with several pairs of options and requesting them to compare these pairs, which is a time-consuming task that users may not be willing to perform, unless the consequences of making a wrong choice are very important for the decision maker.

5.3 Utility Function-based Approaches

In this section, we present approaches that use utility functions[2] to reason about preferences and, as discussed above, the main challenge is to identify which is the utility function of the decision maker. The first presented approach proposes a graphical structure to represent utility functions and algorithms, and the two remaining approaches propose methods to transform qualitative preference statements into utility functions. As our focus is not the elicitation process, we do not describe approaches that propose revealing utility functions by means of user interaction.

5.3.1 CUI networks

CUI networks are claimed by Engel and Wellman (Engel and Wellman 2008) as a compact graphical representation of utility functions over multiple attributes. These networks model multiattribute utility functions using the concept of conditional utility independence — CUI stands for (Conditional) Utility Independence — which requires a (cardinal) preference order over a subset of the attributes to be independent of another subset of attributes.

Goals. A preference representation model (CUI networks) and algorithms.

Questions. Outcome optimisation.

[2]Although these approaches claim to adopt utility functions, they rely on *value* functions according to the definition of Keeney and Raiffa (Keeney and Raiffa 1976).

Input. The required input for answering outcome optimisation queries based on CUI networks is a set of CUI conditions. These conditions indicate when a set of attributes Y is utility independent of a set of attributes X, meaning that the utility of an assignment for Y does not depend on X. In addition, the utility of the assignment for Y may depend on values assigned to a set of attributes Z (conditionality). Considering these sets of attributes, the utility function can be decomposed into

$$U(X, Y, Z) = f(X, Z) + g(X, Z)U(X', Y, Z), g(\cdot) > 0$$

where X, Y, Z are sets of attributes, X' is an assignment for X. The function sums the utility of attributes in X (conditioned to Z), with the utility of attributes in Y (conditioned to Z) — considering a weight $g(\cdot) > 0$ of the utility of Y. In addition, since $U(X', Y, Z)$ is a function only of Y and Z (its value does not change for different assignments X'), it can also be written as $U(Y, Z)$.

The set σ of CUI conditions on the attribute set $S = \{x_1, ..., x_n\}$, such that for each $x \in S$, σ contains a condition of the form $CUI(S\backslash(x \cup P(x)), x \mid P(x))$. The CUI condition indicates a set of attributes $P(x)$, which separates the rest of the attributes from x. If we apply this condition to x_1, for example, we have $U(S) = f_1(x_1, P(x_1)) + g_1(x_1, P(x_1))\, U_{x_1^0}(S\backslash\{x_1\})$.

Interpretation. Not ceteris paribus.

How it works. The proposed approach consists of initially transforming a given set of CUI conditions into a graphical model. For accomplishing this, a procedure is described by Engel and Wellman (not detailed here), which builds a graph whose nodes are attributes based on this given set of conditions and an order on the set S, namely the CUI network. The goal of providing a graphical form for CUI conditions is that, according to the authors, CUI networks provide a potentially compact representation of the multi-attribute utility function, via functional decomposition to lower-dimensional functions that depend on a node and its parents.

Based on CUI networks, two optimisation algorithms for discrete domains were developed. This first is applied only to CUI trees, which are a CUI network in which no node has more than one child (upside down version of a standard directed tree). The main idea of the algorithm is to select an optimal value function (OVF) for a node based on parents and child. However, as the algorithm runs from roots to leaf, the OVF of the child is calculated after visiting the parent, so when the value for the child is set, the parent values may be corrected, and this is propagated to the roots of the tree. The second algorithm applies to general directed acyclic graphs (DAGs). In the tree case, correcting the value of the child of a node x is

sufficient in order to separate x from the rest of the graph, excluding ancestors. Each value of the child is considered at a time, so it also determines the values for all the ancestors. In a general DAG it is no longer sufficient for the OVF to depend on the children, because they do not provide sufficient information to determine the values of the ancestors of x. So this notion is generalised to be the scope of x ($Sc(x)$), which is a set of nodes on which the OVF of x must depend, in order for an iterative computation of the OVF to be sound. With this generalisation, the DAG algorithm is similar to the tree algorithm. Further details can be seen elsewhere (Engel and Wellman 2008).

Complexity. For CUI networks structured as a tree, in case the numeric data at the nodes is available, factoring in the time it takes to recover the utility value for each outcome (which is $O(n)$), the algorithm runs in time $O(n^2 max_i \mid D(x_i) \mid^2)$, where n is the number of attributes and $D(x_i)$ is the domain of attribute x_i. For CUI networks structured as a DAG, the performance of the optimisation algorithm is exponential in the size of the largest scope $Sc(x)$ (plus one).

Limitations. This approach is restricted to dealing only with conditional utility independent functions, which, on the one hand, are a weaker independence condition in comparison to additive independence and, on the other hand, are still a limitation, because it assumes a particular kind of preference independence. In addition, as other quantitative approaches, it requires capturing numeric utility values from users, which is not a trivial task.

5.3.2 Utility functions for Ceteris Paribus Preferences

McGeachie and Doyle (McGeachie and Doyle 2008) present a set of methods for translating preference information from a qualitative representation to a quantitative representation. They consider ceteris paribus preferences, represented with a propositional language augmented with an ordering relation used to express preferences over propositional combinations of a set of elementary attributes. This qualitative representation is converted to an ordinal utility function in order to be used in reasoning processes.

Goals. A transformation from *qualitative statements* to *utility functions*.

Input. The input required by the method is preference statements interpreted under the ceteris paribus semantics and represented in a formal logic. A restricted logical language $\mathcal{L}$ is adopted, using only two logical operators: $\neg$ (*negation*) and $\wedge$

(*conjunction*) to construct finite sentences over a set of atoms, each corresponding to an attribute from a space of binary attributes describing possible worlds. A complete consistent set of literals is a *model*.

A preference order is a complete preorder (reflexive and transitive relation) $\succsim$ over the set of all models of $\mathcal{L}$. When, given two models m and m', $m \succsim m'$, it is said that m is weakly preferred to m'. If $m \succsim m'$ and $m' \not\succsim m$, it is said that m is strictly preferred to m', written $m \succ m'$. If $m \succsim m'$ and $m' \succsim m$, then it is said m is indifferent to m', written $m \sim m'$.

Interpretation. Ceteris paribus.

How it works. The general idea of the approach is to generate an ordinal UF from a set C of ceteris paribus preferences over attributes F. An initial step is performed, which translates the provided statements represented in a logical representation of ceteris paribus preferences to an attribute-vector representation. The latter is then used to produce the UF in the following way. The structure of preference statements in the attribute-vector representation is used to infer additive utility independence among attributes. Next, subutility functions are defined for each utility independent set of attributes. This is done based on the representation of preorders consistent with the preferences by building a graph over assignments to the attributes. Finally, to assign relative weights of satisfaction of different attributes, a linear programming problem is solved. In the end, a utility function that can be used to evaluate the utility of different assignments to values of F is built.

Complexity. This work has two different complexity questions, which are associated with: (i) time and space required to construct the utility function u; (ii) the time and space required to evaluate $u(m)$ on a particular model m. Both of them take exponential time, in worst cases. Constructing u involves solving a satisfiability problem, where time required depends on the number of rules involved in conflicts on each utility independent set. On the other hand, computing $u(m)$ involves computing $u_i(m)$, where u_i is a minimising utility function, for all the utility independent attribute sets. Each of the subutility functions can be exponential in the size of the utility independent set.

Limitations. The main limitations of the approach presented by McGeachie and Doyle (McGeachie and Doyle 2008) are that it is able to deal only with boolean attributes, and the intractability of the solution.

5.3.3 Learning Utility Functions with SVM

Domshlak and Joachims (Domshlak and Joachims 2007) proposed an approach that takes a new direction for reasoning about preferences. First, it adopts a new form of representing preferences: instead of seeing an option as values (parameters) specified for a set of attributes, it linearises the possible combinations of attribute values and each of these combinations is seen in a *unified non-parametric way*, i.e. as a unique specification. For instance, "red big suitcase" is not an option (suitcase) parameterised with values assigned to colour and size, but as a single option. Second, machine learning techniques are adopted to generate a utility function to model user preferences.

Goals. A transformation from *qualitative statements* to *utility functions*.

Questions. Experiments used utility functions constructed with the approach to generate an *ordering* and rank of k best options for users.

Input. The presented approach receives as input qualitative preference statements, which can be of three different types: (i) **dyadic** statements; (ii) **monadic** statements; and (iii) **A is preferred to B more than C is preferred to D**. These are represented with logic-based qualitative *preference expressions*, as detailed in Section 4.5.

Interpretation. Not ceteris paribus. Each parameter u_i of the utility function can be seen as representing the *marginal utility* of the interaction between the attributes associated with u_i when these take specific values. This means that each individual statement does not state a particular interpretation, such as preference independence, but gives a contribution for (in)dependency among attributes.

How it works. The two main aspects from this work are the proposed underlying preference representation, i.e. how the the qualitative preference statements are represented, and how the utility function is generated.

The situation explored by Domshlak and Joachims is when the system cannot assume a significant independence structure on outcome attributes. So, the basic idea is that if no useful preferential independence information in the original representation space is provided, a different space is adopted in which no such independence information is required. This new space is the following: assuming that the attributes X are all binary-valued, there is a map from the options χ into a new, higher dimensional space $\mathcal{F}$ using a certain mapping $\Phi = \chi \mapsto \mathcal{F} = \mathbb{R}^{4n}$.

The mapping Φ is not arbitrary, and it establishes a connection between the dimensions χ and $\mathcal{F}$, in which each element of $\mathcal{F}$ is a combination of values of attributes. This mapping is used for representing preference statements. Each element of $\mathcal{F}$ is associated with weights, and these must be in accordance with provided statements. Statements are used as constraints over weights together with an objective function in an optimisation problem. For calculating weights, techniques frequently used in machine learning in the context of Support Vector Machiness (SVMs) (Vapnik 1998) are adopted.

Complexity. Solving the optimisation problem of this approach poses several complexity issues. First, though this constraint system is linear, it is linear in the exponential space $\mathbb{R}^{4n}$. Second, the very description size of the optimisation problem, and, in fact, of each individual constraint of it, can be exponential in n. Nevertheless, Domshlak and Joachims show that these complexity issues can be overcome by using some duality techniques from optimisation theory and Reproducing Kernel Hilbert Spaces (RKHS).

Limitations. This work is highly associated with machine learning. It interprets each pairwise comparison as likes and dislikes of options with similar values for attributes and builds a model with values that generalises these comparisons, which can also be a comparison with the whole set of options, when monadic statements are provided. Therefore, there is a statistical generalisation, and for achieving good results with this approach a large amount of statements may be needed. This is reflected in the future work reported by the authors, whose goal is to specify the number of preference statements that a user has to provide for inferring a utility function that is effective.

5.4 Constraint Programming

Constraint Satisfaction Problems (CSPs) are mathematical problems defined as a set of objects (options) whose state must satisfy a number of constraints or limitations. Constraints can be seen as preferences that should ideally be satisfied, but when no solution is found for an over constrained problem, these (soft) constraints can be relaxed. Typically, each constraint is associated with a penalty for their not satisfaction (or a degree of preference for their satisfaction), and there is an objective of minimising penalty (or maximising preference satisfaction). In this section, we present approaches that use CSPs for dealing with preferences.

5.4.1 Semiring-based Constraint Satisfaction

Different approaches were proposed to deal with soft constraints, i.e. when constraints are used to formalise desired (preferred) properties rather than requirements that cannot be violated. Examples of these approaches are *fuzzy* and *weighted* constraints, which associate a value with constraints indicating the *preference* for a constraint satisfaction and the *cost* of not satisfying a constraint, respectively. Bistarelli et al. (Bistarelli et al. 1997) defined a constraint solving framework where all such extensions, as well as classical CSPs, can be cast. The main idea is based on the observation that a *semiring* (i.e. a domain plus two operations satisfying certain properties) is all that is needed to describe many constraint satisfaction schemes.

Goals. A preference representation model in the form of an abstract CSP.

Questions. The main question addressed by soft constraints is to find a solution for over constrained problems, so we can say that the typical question answered is *outcome optimisation*, in which the optimal outcome is that satisfying the most important constraints. In this approach, each constraint of a CSP is associated with a preference value (or a penalty) and the goal is to find a solution that maximises the overall preference value (or minimises the penalty of not satisfied constraints).

Input. The required input for soft constraint-based approaches is a Soft Constraint Satisfaction Problem (SCSP), which was introduced in Section 4.1.1, which consists of a constraint system and a set of constraints, which are associated with a *value* representing the penalty for unsatisfied constraints or the preference for their satisfaction.

Interpretation. Not ceteris paribus.

How it works. Semiring-based constraints rely on a simple algebraic structure, called a c-semiring since it is very similar to a semiring, to formalize the notion of satisfaction degrees, or preference levels. Recall that, in this term, “c” stands for “constraint,” meaning that this kind of semiring is a natural structure to be used when handling constraints.

The structure is specified by a set E of satisfaction degrees, where two binary operators are defined: $\times_s$ specifies how to *combine* preferences, while $+_s$ is used to induce a *partial ordering* on E. Additional axioms, including the usual semiring

Table 5.2: Different specific frameworks modelled as c-semirings (Meseguer et al. 2006).

Semiring	**E**	$\times_s$	$+_s$	$\succcurlyeq_s$	0	1
Classical	t, f	$\wedge$	$\vee$	$t \succcurlyeq_s f$	f	t
Fuzzy	[0, 1]	min	max	$\geq$	0	1
k-weighted	$0, ..., k$	$+^k$	min	$\leq$	k	0
Probabilistic	[0, 1]	xy	max	$\geq$	1	0
Valued	E	$\otimes$	min_v	$\preccurlyeq_v$	$\top$	$\bot$

axioms, are added to precisely capture the notion of satisfaction degrees in soft constraints.

A c-semiring is a 5-tuple $< E, +_s, \times_s, 0, 1 >$ such that:

- E is a set, $0 \in E$, $1 \in E$;
- $+_s$ is an operator closed in E, associative, commutative and idempotent for which 0 is a neutral element and 1 an annihilator;
- $\times_s$ is an operator closed in E, associative and commutative for which 0 is an annihilator and 1 a neutral element; and
- $\times_s$ distributes over $+_s$.

Compared to a classical semiring structure, the additional properties required by a c-semiring are the idempotency of $+_s$ (to capture a lattice ordering) and the existence of a minimum and a maximum element (to capture hard constraints).

In the SCSP framework, the values specified for the tuples of each constraint are used to compute corresponding values for the tuples of values of the attributes in con (set part of the constraint problem), according to the semiring operations: the multiplicative operation is used to combine the values of the tuples of each constraint to get the value of a tuple for all the attributes, and the additive operation is used to obtain the value of the tuples of the attributes in the type of the problem. More precisely, this is the definition of the operations of *combination* and *projection* over constraints.

The c-semiring is a generic framework for representing soft constraint problem solvers, and by capturing their commonalities in a generic framework one can design generic algorithms and properties instead of several apparently unrelated, but actually similar properties, theorems and algorithms. Different specific frameworks were presented as soft constraint networks (Meseguer et al. 2006, Bistarelli et al. 1997), which are summarised in Table 5.2.

Complexity. Since semiring-based constraints properly generalise classical constraints, this task is NP-hard. As in the classical case, perhaps the most direct

way to solve a soft constraint network is searching in its state space, exploring the set of all possible assignments. Since an optimal solution is an assignment that minimises the violation degree (or equivalently, maximises the satisfaction degree), solving optimally a soft constraint network is an optimisation problem, thus harder than solving classical constraint networks. Different techniques were proposed to optimise optimal solutions, such as defining lower and upper bounds of branch and bound algorithms.

Limitations. As other quantitative approaches, soft constraint-based approaches have the limitation of extracting from users numerical values for preferences, what is not intuitive for them. In addition, solving SCSPs is a NP-hard task. Finally, these approaches do not deal with multiple objectives, the only objective is to maximise the constraints that are satisfied (considering their weight), but additional objectives such as minimise price cannot be represented. Only intervals for acceptable prices can be provided as constraints.

Extensions

There are extensions of the c-semiring framework, whose goal is to represent other kinds of preferences. Next, we briefly describe two of these extensions.

Representing Bipolar Preferences. Bistarelli et al. (Bistarelli et al. 2010) state that preferences on a set of possible choices are often expressed in two forms: negative and positive statements. The former restricts the set of acceptable options, indicating what users do not want, and the latter indicates options that users prefer with respect to other acceptable options. These two kind of preferences are referred to as *bipolar* preferences. The approach described for bipolar preferences proposes a tool to represent these two types of preferences in a single framework and provides algorithms that, given as input a problem with these preferences, return its best solutions.

Representing Intervals. Gelain et al. (Gelain et al. 2010) argue that it is difficult to specify precise values associated with constraints to represent preferences, and it is more reasonable to consider intervals instead of specific values. The authors then define several definitions for optimal solutions in such problems, providing algorithms to find optimal solutions and also to test whether a solution is optimal.

5.4.2 Preference-based Problem Solving for Constraint Programming

Junker (Junker 2008) points out that traditional optimisation approaches compile preferences into a single utility function and use it as the optimisation objective when solving the problem, but they do not explain why the resulting solution satisfies the original preferences, and do not indicate the trade-offs made during problem solving. The author then argues that the whole problem solving process becomes more transparent and controllable by the user if it is based on the original preferences.

He tackled the problem of multi-objective optimisation by decomposing it into alternative sequences of single-criterion optimisation problems, which can be solved by standard optimisers. The chosen sequence gives information that explains the optimality of the solution. Based on the explanation, the user can either accept the solution or modify the preferences. The problem solver then modifies the solution correspondingly. Preferences thus allow the user to interact with the problem solver and to control its behaviour.

Goals. A preference representation model (introduced in Section 4.1.2), and an algorithm based on constraint problem solvers to find optimal outcomes.

Questions. Outcome optimisation. In addition, based on an optimal solution, users may ask two questions.

1. Why can't the criteria z have a value better than ω^* (the optimal solution)?
2. Why hasn't the value ω been chosen for the criteria z?

Input. A set of preferences are modelled in the form of a preorder $\succsim$ on criteria and their respective domains, consisting of a strict part $\succ$ and an indifference relation $\sim$ (see Section 4.1.2 for details), and a set of available options, which are described in terms of values for attributes that have specific domains, and constraints over possible values.

Interpretation. Not ceteris paribus.

How it works. As opposed to combinatorial problems with preferences, which are classically solved by compiling all preferences into a single utility function and by determining an option satisfying the constraints that has maximal or nearly maximal utility, Junker uses individual preferences with constraint solvers, in order

to allow the optimiser to give an explanation of optimality in terms of the original preferences.

First, an atomic optimisation step is performed, which finds options satisfying the constraints C that assign a $\succ$-maximal value to a single criterion z. This is possible if the user provides a total order over the domain of z, but if it is not the case, all possible total orders are generated based on the provided partial order, and the constraint solver runs for each possible total order. Maximal options in this case are the union of maximal options of each of them.

In order to address multiple criteria optimisation, importance order is introduced on the preferences and to decide trade-offs in favour of the more important preferences. Lexicographic optimisation is then used to define an ordering on the option space based on this importance principle, which leads to finding a *lexicographically optimal option*. As the importance among the criteria is unknown, permutations of all possible orders are done. If users are unsatisfied with the results, they can establish the importance among the criteria, which is a strict partial order $I \subseteq Z \times Z$. Thus, only permutations that respect this strict partial order will be considered.

Finally, pareto-optimality is adopted to capture all the possible trade-offs, because using a lexicographic approach gets a best value for a more important criterion z_1, and a less important criterion z_2 is completely penalised. In this case, pareto-optimal options are found, which are those that pareto-dominates other options, i.e. optional options are better in at least one criterion, and equally good in the other ones. As there is no direct way to transform a pareto-optimization problem into a solved form of the optimisation problem even if it is based on totally ordered preferences, the author introduces the notion of wishes, which establish penalisation limits for criteria.

Complexity. CSPs are known to be NP-hard, but this solution is based on the use of standard optimisers, such as constraint-based branch-and-bound, which improve performance.

Limitations. There are some steps in the work described by Junker that are not completely detailed, such as the mapping of preferences $\langle z, > \rangle$ to a utility function u. In addition, one of the most challenging problems in preference reasoning is dealing with trade-off. When a solution is optimal for all criteria, it is trivially optimal. However, requiring users to establish penalisation limits for resolving trade-offs has two main drawbacks: (i) for domains that users are not very familiar with, they might have a vague idea of which is a good limit to establish the trade-off; and (ii) this solution may lead to a situation in which the more important criterion is

maximised to a value that sets the maximum penalty to the less important criterion; however users may prefer something in between.

5.5 Graphically-structured Approaches

As seen in previous sections, the use of utility functions and CSPs has drawbacks associated with preference representation as they require not trivial preference elicitation processes. In this section, we present approaches that aim at addressing them by proposing new forms of representing preferences and relationships among attributes. These representations, which rely on graph structures, have the goal of capturing preferences in a intuitive way and also to be used in algorithms to reason about preferences.

5.5.1 CP-nets

CP-nets represent preferences in a compact graphical form and, according to Boutilier et al. (Boutilier et al. 2004), represent preference statements in a natural way. These statements are conditional qualitative preference statements interpreted under the ceteris paribus semantics.

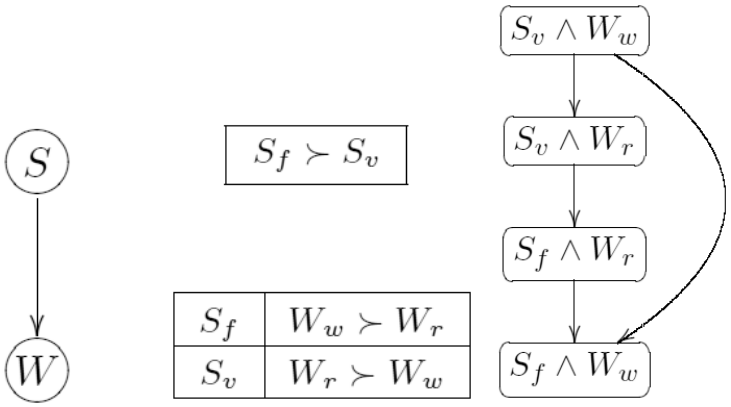
Goals. A preference representation model (CP-nets) and algorithms.

Questions. Outcome optimisation, dominance queries and ordering queries.

Input. A CP-net (introduced in Section 4.2.1) over attributes $V = X_1, ..., X_2$ is a directed graph G over $X_1, ..., X_2$ whose nodes are annotated with conditional preference tables $CPT(X_i)$ for each $X_i \in V$. While the required input for outcome optimisation requires only a CP-net as input, and optionally a partial assignment for attributes, the input for both dominance and ordering queries is a CP-net N and two outcomes o and o'.

Interpretation. Ceteris paribus.

How it works. In order to provide a better understanding of CP-nets, we first introduce a simple example (Boutilier et al. 2004) that expresses preferences over dinner configurations. This network, depicted in Figure 5.1(a), consists of two attributes S and W, standing for soup and wine, respectively. Fish soup (S_f) is strictly preferred to vegetable soup (S_v), while preference between red (W_r) and



5.1(a): Preferences for soup and wine. 5.1(b): Induced graph.

Figure 5.1: Example of a CP-net (Boutilier et al. 2004).

white (W_w) wine is conditioned on the soup to be served: red wine is preferred if served with a vegetable soup, and white wine if served with a fish soup.

The semantics of a CP-net is defined in terms of the set of preference rankings that are consistent with the set of preference constraints imposed by its conditional preference tables (CPTs). Figure 5.1(b) shows the preference graph over outcomes induced by the previously described CP-net. An arc in this graph directed from outcome o_i to o_j indicates that a preference for o_j over o_i can be determined directly from one of the CPTs in the CP-net.

Next we briefly describe how each of the preference-related questions are answered using CP-nets.

Outcome Optimisation. In order to generate the optional outcome of a CP-net, the network is sweep from top to bottom. The *forward sweep procedure* (Boutilier et al. 2004) is provided, which constructs the most preferred outcome. It considers, in order to be more general, a given evidence constraining possible outcomes in the form of an instantiation z of some subset $Z \subseteq V$ of the network attributes. The algorithm sets $Z = z$, and instantiate each $X_i \notin Z$ in turn to its maximal value given the instantiation of its parents.

Ordering queries. For acyclic CP-nets, Boutilier et al. define a corollary that provides an algorithm for answering ordering queries. The corollary states that given an acyclic CP-net N and a pair of outcomes o and o', if there exists an attribute X in N, such that: (i) o and o' assign the same values to all ancestors of X in N; and (ii) given the assignment provided by o (and o') to the parents of X, o assigns a more preferred value to X than that assigned by o', then $N \nvDash o' \succ o$.

Dominance queries. Boutilier et al. demonstrate the equivalence of answering dominance queries with the task of determining the existence of an improving (or worsening) sequence of attribute value flips with respect to the given CP-net. A sequence of improving flips from one outcome to another provides a proof that one outcome is preferred, or dispreferred, to another in all rankings satisfying the network. The authors also showed that this task can be reduced to a special subclass of classical planning problems, and presented several techniques that can be used in a generic search procedure for an improving flipping sequence.

Complexity. CP-nets were investigated to be used with different algorithms related to reasoning about preferences, and these had their complexity analysed with different graph topologies. Outcome optimisation queries can be answered using the forward sweep procedure, taking time linear in the number of attributes. The complexity of two of the comparison queries (dominance and ordering) has been analysed for different graph topologies of CP-nets, which can be seen in Table 5.3. These results are associated with boolean attributes, but ordering results also hold for multi-valued attributes.

Table 5.3: Complexity Analysis of CP-nets (boolean attributes).

Graph topology	Dominance	Ordering
Directed Tree	$O(n^2)$	$O(n)$
Polytree ($indegree \leq k$)	$O(2^{2k}n^{2k+3})$	$O(n)$
Polytree	NP-Complete	$O(n)$
Singly Connected ($indegree \leq k$)	NP-Complete	$O(n)$
DAG	NP-Complete	$O(n)$
General Case	PSPACE-Complete	NP-Hard

Given a CP-net N over n attributes and a set of complete assignments $o_1, ..., o_m$, ordering these assignments consistently with N can be done using ordering queries only, in time $O(nm_2)$.

Limitations. This work leaves different open theoretical questions, mainly related to the complexity analysis of other scenarios in which CP-nets are used to reason about preferences. In addition, CP-nets address representing preferences over discrete domains, and common preferences, such as price minimisation, cannot be directly represented. Moreover, CP-nets, when identifying optimal outcomes, first optimise parent attribute values for later assigning values for their children. This leads to a lexicographic importance among attributes, which may capture trade-off situations in a not appropriate way. This issue is investigated in TCP-nets, presented in Section 5.5.4, which add importance relations and conditional relative importance statements to the the conditional ceteris paribus statements supported by CP-nets.

5.5.2 Combining CP-nets and Soft Constraints

Domshlak et al. (Domshlak et al. 2003) establish a connection between the CP-nets and soft constraints machinery. As dominance queries are hard to be answered using a CP-net structure (Section 5.5.1), the authors propose constructing a semiring structure with a given acyclic CP-net, in order to be able to answer this kind of queries.

Goals. A transformation from *CP-nets* to *SCSPs*.

Questions. Domshlak et al. focus on answering dominance queries *in polynomial time*. This is achieved by using a semiring structure, so the main question is how to transform a CP-net into this structure, so that dominance queries can be answered in polynomial time.

Input. An acyclic CP-net.

Interpretation. Ceteris paribus.

How it works. This work consists of approximating CP-nets via soft constraints, i.e. defining a SCSP based on a CP-net. This provides a uniform framework to combine user preferences with both hard and soft constraints. So, given an acyclic CP-net, a corresponding SCSP is constructed in two steps. First, a constraint graph, named SC-net, is built. Second, preferences and weights for the constraints in the SC-net are computed, and this computation depends on the actual semiring framework being used. Basically, the SC-net constructed consists of one node for each node of the CP-net plus new nodes for nodes from the CP-net that have more than one parent. Arcs correspond to hard and soft constraints, and the latter is associated with weights and penalties.

For calculating these values, two alternative semiring frameworks are presented, based on (i) min+ and (ii) Soft-constraint Lexicographic Ordering (SLO) semirings. In both cases, the computation of preferences and weights ensures information preserving and satisfies the cp-condition, i.e. approximations preserve the ceteris paribus property.

Complexity. Given an acyclic CP-net N with the node in-degree bounded by a constant, the construction of the corresponding SC-net based on weighted SCSP

N_c is polynomial in size of N. No complexity analysis was presented for SLO soft constraints.

Limitations. This approach aims at solving an issue of CP-nets — tractability of dominance testing — by approximating a CP-net ordering via soft constraints. Even though this goal is achieved, precision is compromised to some degree. Different approximations (such as min+ and SLO) can be characterised by how much of the original ordering they preserve, the time complexity of generating the approximation, and the time complexity of comparing outcomes in the approximation. Incomparable outcomes with min+ are considered equal or ordered in either ways, and with SLO, an strict order will always be defined.

5.5.3 UCP-networks

A directed network representation for utility functions that combines certain aspects of quantitative and qualitative approaches for preferences was proposed by Boutilier et al. (Boutilier et al. 2001). The UCP-network formalism is an extension of the CP-network model (Boutilier et al. 2004) that allows one to represent quantitative utility information rather than only preference orderings.

Goals. A preference representation model (UCP-networks) and algorithms.

Questions. Outcome optimisation and dominance queries.

Input. The required input of this approach is a network representation, namely UCP-networks (or UCP-nets), which combine aspects of CP-nets and generalised additive independence (GAI) models. Sets of attributes X_i are said to *generalised additive independent* if the expected value of the utility function u is not affected by correlations between the X_i, but it depends only on the marginal distributions over each X_i.

UCP-nets are similar to CP-nets in that they are also a graph whose nodes are attributes, and an arc from one node to another indicates that preferences over an attribute are conditioned to values of a parent attribute. However, the graph structure is restricted to a DAG and instead of annotating nodes with CPTs, a table associates values of an attribute and its parents with a utility value. In other words, a UCP-net extends a CP-net with conditional utility information. For purposes of elicitation and computation, it is often convenient to normalise utilities over the range [0, 1].

Every UCP-net specifies a GAI decomposition of its underlying utility function, i.e. the utility of an outcome is calculated by summing the different

factors $f_i(X_i, U_i)$, where X_i is each individual attribute, and U_i is the parents of X_i. In addition, the GAI decomposition must respect the conditional preference independence imposed by the network.

Interpretation. Ceteris paribus.

How it works. For dominance queries, the utilities of two outcomes are compared, and they are obtained by extracting and summing the values of each factor in the network. On the other hand, for outcome optimisation queries, the CP-net structure is exploited and the forward sweep procedure (Boutilier et al. 2004), introduced in Section 5.5.1, is used.

One of the main reasons to move from qualitative to quantitative preference models is to support decisions under (quantified) uncertainty about outcomes of options. So, when the distributions induced by options can be structured in a Bayes net (Jensen 2001), UCP-nets can be used to help structure the decision problem. The approach presented (Boutilier et al. 2001) for this purpose uses GAI factorisation of utilities afforded by the UCP-net.

Complexity. Both outcome optimisation and dominance queries can be done in time linear in the size of the network, due to the use of CP-nets with utility functions.

Limitations. An initial problem of this approach is that determining if a quantified network is in fact a UCP-net requires a case-by-case analysis for each "extended family" in the network involving a number of tests exponential in the size of the extended families (each consisting of an attribute, its parents, its children, and its children's parents). However, it is not so problematic, given that families are not expected to have a large size. The major issue is related to the adoption of a quantitative approach, which brings problems to the elicitation process. The authors assume that both structure and local value functions are something that users will often be able to provide without too much difficulty.

5.5.4 TCP-nets

CP-nets represent the class of conditional qualitative preference statements. Brafman et al. (Brafman et al. 2006) address a limitation of this approach by extending CP-nets to capture another class of preference statements: those with conditional relative importance. These statements have the following form: "it is more important to me that the value of X be high than that the value of Y be high."

CP-nets were extended with new types of arcs, and questions to be answered based on this new graphical representation of preferences were investigated.

Goals. A preference representation model (TCP-nets) and algorithms.

Questions. Outcome optimisation, dominance and ordering queries. However, little attention has been given to ordering queries, as they are weaker than dominance queries.

Input. A TCP-net (which is a CP-net that also allows to represent attribute importance, and was presented in Section 4.2.2) is the only required input for outcome optimisation, and it can also be provided with an optional partial assignment for attributes. The required input for both dominance and ordering queries is a TCP-net N and two outcomes o and o'.

Interpretation. Ceteris paribus.

How it works. Informally, the semantics of TCP-nets is as follows. A strict partial order $\succ$ satisfies:

- *the conditional preferences for attribute* X, if any two complete assignments that differ only on the value of X are ordered by $\succ$ consistently with the ordering on X values in the CPT of X. This ordering can depend on the parent of X in the graph;
- *the assertion that* X *is more important than* Y, if given any two complete assignments that differ on the value of X and Y only, $\succ$ prefers that assignment which provides X with a better value; and
- *the assertion that* X *is more important than* Y *given some assignment* z *to attribute set* Z, if given any two complete assignments that differ on the value X and Y only, and in (both of) which Z is assigned z, $\succ$ prefers that assignment which provides X with a better value.

A particular class of TCP-nets is investigated by Brafman et al., because it is a class of networks that are proven satisfiable. This class of TCP-nets consists of conditionally acyclic TCP-nets, whose key property is that they induce an "ordering" over the nodes of the network. A TCP-net is conditionally acyclic if its induced dependency graph is acyclic and for every assignment w to the union of all selector sets (i.e. sets of the attributes associated with conditional importance arcs) of the network, the induced w-directed graphs are acyclic. Therefore, verifying if

a TCP-net is conditionally acyclic becomes a relevant issue. This is performed in polynomial time in certain situations, which are identified and detailed elsewhere (Brafman et al. 2006); however, this verification is generally hard.

Outcome Optimisation. For computing the most preferred outcome of an acyclic TCP-net and a (possible empty) partial assignment x on its attributes, the forward sweep procedure introduced in Section 5.5.1 can be used. Nevertheless, it is not possible if a set of hard constraints is provided. This happens because the key difference between processing an acyclic CP-net and a conditionally acyclic TCP-net is that, while the former induces a single partial order of importance over attributes, the latter induces a hierarchically structured set of such partial orders. So, a branch-and-bound algorithm is presented for computing the optimal outcome in this scenario, which consists of two parts: *Search-TCP* and *Reduce*. The *Search-TCP* algorithm is guided by the underlying TCP-net N. It proceeds by assigning values to the attributes in a top-down manner, assuring that outcomes are generated according to the preferential ordering induced by N. The *Reduce* algorithm, in turn, refines a TCP-net N with respect to a partial assignment K': it reduces both the CPTs and the CITs involving this attribute, and removes this attribute from the network.

Dominance queries. Much like in CP-nets, a dominance query $\langle N, o, o' \rangle$ with respect to a TCP-net can be treated as a search for an improving flipping sequence from the (purported) less preferred outcome o' to the (purported) more preferred outcome o through a sequence of successively more preferred outcomes, such that each flip in this sequence is directly sanctioned by the given TCP-net.

Complexity. Brafman et al. present a theorem that states that every conditionally acyclic TCP-net is satisfiable. Based on this theorem outcome optimisation and dominance queries in the context of this class of TCP-nets are investigated, with the following complexity analyses provided.

- Determining that a TCP-net is conditionally acyclic is co-NP-hard.
- TCP-nets have the optimal outcomes generated using the forward sweep procedure, and therefore the complexity for performing this task also applies for TCP-nets.
  - If a set of hard constraints is provided, determining the set of preferentially non-dominated outcomes is not trivial, and a

branch-and-bound algorithm is used, whose worst case leads to exponential time complexities.

– Dominance testing with respect to CP-nets, and thus TCP-nets, is NP-hard.

Limitations. As TCP-nets are an extension of CP-nets, they inherit some of the limitations of CP-nets, which are related to the complexity of algorithms to reason about preferences and the types of attributes addressed. In addition, even though TCP-nets capture importance relations between attributes, the way that these relations are interpreted lead to the maximisation of the most important attribute and a high penalisation for the least important ones.

5.5.5 Graphically Structured Value Function Compilation

Brafman and Domshlak (Brafman and Domshlak 2008) provide an approach for compiling the information captured by CP-nets and TCP-nets to value functions. As reasoning about preferences represented by value functions is easy as they establish an order among outcomes by ordering computed values, the challenge here is to convert these graphical structures into them. The authors assume that preference statements were already elicited or informed by users, and represented as a TCP-net — or a CP-net, in case there is no (conditional) relative importance between pairs of attributes. The graphical structure plays an important role in analysing and compiling these statements.

Goals. A transformation from *TCP-nets (and CP-nets)* to *value functions*.

Input. The input required is qualitative preferences represented in either a CP-net or a TCP-net, therefore the approach is able to deal with the kinds of statements represented by these graphical structures.

Interpretation. Ceteris paribus.

How it works. This work proposes to use a TCP-net to initially organise qualitative preference statements obtained from the user. Then, this information is compiled to a value function that maintains the qualitative structure and independence assumptions implicit in this TCP-net. This obtained value function is used as the model of user's ordinal preference, and as new information comes from the user, this value function is refined, while still maintaining independence assumptions implied by the original TCP-net, if possible.

A TCP-net can be represented as a value function, as it defines a (partial) order among outcomes. So, the main issue is to investigate whether there exists a generalised additive (GA) value function defined over small factors "implied" by the structure of the network, i.e. find a structured value function that, in some sense, is as *compact* as the original TCP-net.

For CP-nets, the concept of a CP-family is defined, which is the set of an attribute and its parents. That is, the goal is to define a GA value function whose factors are the families of the CP-net. It is shown that every acyclic CP-net is GA-decomposable over its CP-families, by constructing a system of linear inequalities (CP-conditions) and finding a solution for it. A similar approach is presented for TCP-nets — every acyclic TCP-net is GA-decomposable over its TCP-families — a TCP-family is the set of an attribute, its parents and the attributes that impose conditionality to the target attribute.

The approach is also extended to incorporate new item-level rankings provided by users. It is shown that this value function compilation is not only efficient and sound, but also complete, that is, the ability to generate the value function is guaranteed.

Complexity. A system of linear inequalities L, either for CP-nets or TCP-nets, is locally exponential. However, if the maximum cardinality of all extended CP-families (an attribute, its parents, its children, and its children's parents) of a CP-net N is a constant k, a value function consistent with N can be constructed in time polynomial in the size of N. It also holds for TCP-nets.

Limitations. Regarding preferences representation and interpretation, as this work relies on TCP-nets, limitations described for them (and CP-nets), are also applied. Moreover, the authors themselves point out that their work raises numerous open theoretical questions, including (Brafman and Domshlak 2008): (i) when (if at all) GA-decomposition is complete for cyclic TCP-nets, or even just cyclic CP-nets? (ii) what is the most compact form of GA-decomposition that is complete for all consistent TCP-nets?

5.6 Query-based Approaches

Most of the presented approaches were investigated in the context of AI, but more recently preferences have also been taken into account in research related to databases. This was mainly motivated by a scenario that emerged from the web, as users search online store databases and their specified preferences often result in a query over constrained, which has no result. In addition, web search engines provide

a huge amount of results for a set of keywords (a query), which can be reduced and ranked if preferences are taken into account. This section presents work that aims at integrating preferences into a query language.

5.6.1 Scoring Function

Agrawal and Wimmers (Agrawal and Wimmers 2000) proposed a framework for expressing and combining user preferences, based on the definition of an atomic scoring function, together with combination operations used to compute a score for each result tuple. The framework has two basic components: (i) a preference function that specifies user preferences; and (ii) a single meta-combining form *combine* that is based on value functions. Details of how the framework works are provided below.

Goals. A preference representation model and algorithms.

Questions. Ordering queries: how options are ranked according to preference functions provided by multiple users?

Input. Given a framework (detailed in Section 4.3.1) in which preferences for an entity in this framework are expressed by a numeric score between 0 and 1, vetoing it, or explicitly stating indifference, the required input to answer ordering queries is a database with available options and a set of preference functions, which are functions that map options to a score.

Interpretation. Not ceteris paribus. A preference function defines scores for specific options (or projections), and therefore a score is provided for a fully specified context, or, if the score is given for an option projection, it means that the score is also valid for any combination of values for the remaining option names (or attributes). In addition, when the wild type (*) is used for a name, the specified score will be considered for options with all possible values for that name too.

How it works. Preference functions consist of scores provided by (or elicited from) users for individual combinations of values of option names. Then, a preference function meta-combining form is defined, named *combine*, which takes a "*value function*" that states how to compute a new score based on the original scores and produces a preference function combining form, which takes a finite list of preference functions and produces the new preference function. *Combine* abstracts how different scores are combined: this function receives a function

as parameter, which determines how to combine scores. The example provided where the framework was proposed (Agrawal and Wimmers 2000) is a `FirstVeto` function, which establishes that scores of an specific individual are more important than of another.

Limitations. In terms of preference representation, this approach requires users to provide scores for each combination of values, if no elicitation method is used. In addition, no particular inference is performed to reason about preferences: scores order options in a particular order, and therefore ranking options is straightforward. Moreover, the provided preference function combining form is abstract, in the sense that resolving score combination is still part of specific instantiations of the framework, and the example provided (`FirstVeto`) is trivial.

5.6.2 Winnow

An extension to relational algebra was proposed by Chomicki (Chomicki 2003) as preference query formalisation, in which preferences are expressed as binary relations between tuples from the same database relation, which represent options to be selected. The approach defines a central algebraic operator (*winnow*), which selects the set of dominant options from a database according to provided preferences, and this operator has algebraic properties analysed, such as commutativity, commutation of selection or selection, and distribution of winnow over union and different. Analysing such properties of winnow is important (Chomicki 2003) as they can be exploited for formulating efficient database query execution plans.

Goals. A preference representation model, an algebra and algorithms.

Questions. Non-dominated outcomes.

Input. As input, it is required a database with available options and a set of preference formulae, defined as introduced in Section 4.3.2.

Interpretation. Not ceteris paribus.

How it works. An algebraic operator called *winnow*, whose definition is presented below, picks from a given relation the set of the most preferred (dominant) options, according to a given preference formula. A preference query is a relational algebra query containing at least one occurrence of the winnow operator.

Winnow. *If R is a relation schema and C a preference formula defining a preference relation $\succ_C$ over R, then the winnow operator is written as $\omega_C(R)$, and for every instance r of R:*

$$\omega_C(R) = \{t \in r \mid \neg \exists\, t' \in r.t' \succ_C t\}$$

There are three possible algorithms (Chomicki 2003) to evaluate winnow. The first is a nested-loops algorithm, which compares every two options and discards dominated ones. This algorithm is correct for any preference relation $\succ$. The second, BNL, is an algorithm proposed by Borzsonyi et al. (Borzsonyi et al. 2001) in the context of a specific class of preference queries, namely skyline queries, but the algorithm is considerably more general than the previous one. And the third (Chomicki et al. 2005), SFS, is a variant of the second, in which a presorting step is used. The BNL and SFS algorithms require the preference relation to be a strict partial order.

Complexity. The evaluation of the winnow operator, i.e. identifying dominant options, is not part of the solution provided by Chomicki. However, the author provides a complexity analysis of properties (irreflexivity, asymmetry, transitivity, negative transitivity and connectivity) of preference formulae, which can be seen elsewhere (Chomicki 2003). The best case complexity of the simple nested-loops algorithm presented is of the order of $O(n)$; n being the number of options in the input. In the worst case, the complexity is of the order of $O(n^2)$.

Limitations. The main issue investigated by this approach is algebraic properties of an operator added to relational algebra; however, the problem of identifying the most preferred options is abstracted. Moreover, the goal is to identify non-dominanted options, and there is no way to tell which is the optimal option from this subset of options. Finally, even though any preference can be expressed with a preference formula, including establishing a full total order among options, it is not compact, which is relevant when preferences are provided by users. The preference formula below, which expresses "*I prefer white wine with fish, and red wine with meat*," illustrates this issue.

$$\begin{aligned}(d, dt, w, wt) \succ_C (d0, dt0, w0, wt0) \equiv (d = d' \wedge dt = \text{`}fish' \wedge wt = \text{`}white' \\ \wedge dt' = \text{`}fish' \wedge wt' = \text{`}red') \\ \vee (d = d' \wedge dt = \text{`}meat' \wedge wt = \text{`}red' \\ \wedge dt' = \text{`}meat' \wedge wt' = \text{`}white')\end{aligned}$$

5.6.3 Best-Matches-Only Query Model

Kießling (Kießling 2002) defines preferences as strict partial orders and proposes several preference constructors in order to support the accumulation of single preferences into more complex ones. The author also provides a collection of algebraic laws to manipulate such preference constructions, as well as defines how to evaluate preference queries under the Best-Matches-Only (BMO) query model and decomposition algorithms for complex preference queries.

Goals. A preference representation model, and an algebra.

Questions. Non-dominated outcomes.

Input. The input needed for querying non-dominated outcomes is a database with available options and a set of preferences, built with preference constructors, presented in Section 4.3.3. In summary, there are two classes of constructors: (i) non-numerical base preferences, which include positive and negative preferences; and (ii) numerical base preferences, which include preferences that specify preferred numerical values of an attribute using, for example, intervals. A preference term (i.e. a preference that is valid according to the provided constructors) can also be composed of other preferences.

Interpretation. Not ceteris paribus.

How it works. The BMO result set contains only the best matches with respect to the strict partial order of a preference P. It is a selection of unordered result of options. All options t, $t' \in BMO$ have equal or incomparable values regarding the preference P.

A preference query is defined by $\sigma[P](R)$ declaratively as follows: $\sigma[P](R) = \{t \in R \mid t[A] \in max(P^R)\}$. $\sigma[P](R)$ evaluates P against a database set R by retrieving all maximal values from P^R. Preference queries behave non-monotonically, in that they are sensitive (holistic) to the quality of a collection of values. In addition, the following manipulations are provided in order to evaluate preference queries:(i) preference hierarchies; (ii) decomposition of '+' and and '♦'-queries; (iii) decomposition of '&'-queries; and (iv) decomposition of '⊕'-queries. These are accumulating and aggregating preference constructors introduced in Section 4.3.3.

Limitations. This approach is similar to the winnow operator (Section 5.6.2), and the same limitations reported for it are applicable to the BMO query model.

5.6.4 Query Personalisation based on Preferences

The preference model proposed by Koutrika and Ioannidis (Koutrika and Ioannidis 2006) is associated with preference selection algorithms, generation of personalised answers and raking functions. These parts work together in order to provide personalised results for a database search query, according to the preferences of a specific user. Users have several preferences, and as only some of them are relevant for a specific query, algorithms are proposed for selecting preferences related to a query according to various criteria. Based on a set of selected preferences and a query to be executed, the approach generates personalised query and results, which are ranked based on the estimated user interest.

Goals. A preference representation model, and algorithms.

Questions. Ordering queries, with a focus on the context of databases: how query results are selected and ranked according to user preferences?

Input. The personalisation process has two main steps, and the output of the first (*preference selection*) is used as input of the second (*generation of personalised results*). Preference selection receives as input a set of user preferences, a query to be executed, and a parameter K, which is a criterion for choice. The second step requires the query to be executed, the set of preferences selected in the previous step of query personalisation, and an explicit or implicit specification for L, which indicates the number of preferences that should at least be satisfied. Preferences, as detailed in Section 4.3.4, may be expressed for values of attributes, and for relationships between entities, which indicate to what degree, if any, entities related depend on each other.

Interpretation. Not ceteris paribus. The degree of interest given for each part of attribute-value is independent of other attributes.

How it works. The first step of the query personalisation process deals with the extraction of the top (most critical) K preferences related to a query. The selection is based on a syntactic level, which means that a preference is related to a query, if it maps to a path attached to the query graph. K is a criterion based on the degree

of criticality of preferences, which may specify for instance the top X preferences, or preferences with a degree of criticality above a threshold c_0.

With the execution of this first step, the top K preferences are integrated into the user query and a personalised answer is generated. This answer should be: (a) *interesting* to the user, in that it should satisfy (at least) L from the top K preferences; (b) *ranked* based on the degree of interest; and (c) *self-explanatory*: for each option returned, the preferences satisfied or not should be provided in order to explain its selection and ranking.

Two approaches are described for the generation of personalised answers: *Simply Personalized Answers (SPA)*; and *Progressive Personalized Answers (PPA)*. The first integrates the top K preferences into the initial query and builds a new one, which is executed. A personalised query is formulated as the union of a set of sub-queries, each one mapping to one or more of the K preferences selected. The last integrates preferences into sub-queries as in the SPA methodology. 1-to-many absence preferences are integrated as if they were presence ones. Hence, two sets of sub-queries are formed: a set of subqueries involving presence and 1-to-1 absence preferences, Q_s, and a set of subqueries involving 1-to-many absence preferences, Q_a. After the execution of each sub-query, a degree of interest d_t is calculated for each result using any *ranking function* for positive, negative or mixed combinations of preferences. A class of ranking functions is provided, and we refer to reader to the provided reference (Koutrika and Ioannidis 2006) for details. The list of options returned, R, is ordered in decreasing degree of interest — those that do not achieve a minimum degree of interest are discarded, and the remaining ones are given as the result of the query.

Limitations. Besides the problem of relying on a quantitative preference model, which are hard to elicit, the approach lets the *ranking function* as user-defined function, which is one of the main issues of the decision making process. Finding a numeric value that represents user preference for an option by combining individual preferences is not trivial, and this is left as a variable point of the approach.

5.6.5 OWLPref

The declarative, domain-independent way of representing preferences in OWL, namely OWLPref (Ayres and Furtado 2007), described in Section 4.4.1, has an associated API, which maps concepts from the OWLPref to queries of the SPARQL Preference (Siberski et al. 2006) query language. The authors support this choice by arguing that SPARQL is currently the most important and used language for querying semantic data, and also because they focused on the

ontology modelling, and not the inference engine for query execution, which is provided by SPARQL. The idea is that each preference defined in OWLPref has a defined mapping to a SPARQL preference statement, nevertheless this mapping is not detailed (Ayres and Furtado 2007), and it is not straightforward — there are kinds of preferences in OWLPref, e.g. `AroundPreference`, that do not have a corresponding constructor in SPARQL Preference.

As the concrete preference reasoning is performed by SPARQL Preference and the mapping process from OWLPref to it is not provided, we dedicate the remainder of this section to detail this SPARQL extension that deals with preferences. SPARQL Preference is extensively based on the winnow operator (Chomicki 2003), which was described in Section 5.6.2.

Goals. A preference representation model (extension of a query language) and formal definitions for each construction (semantics).

Questions. Non-dominated outcomes. More precisely:

1. Which are the solutions non-dominated by any other solutions? (*Skyline queries*)
2. Which are the solutions that satisfy all the (hard and soft) constraints? If none, by relaxing some or all soft constraints, which are best answers? (*Soft constraints*)

Input. A database with available options and a SPARQL Preference query, which is expressed in SPARQL with the `PreferringClause` extension, as shown below.

```
SolutionModifier ::= PreferringClause? OrderClause? LimitClause?
                     OffsetClause?
PreferringClause ::= 'PREFERRING' MultidimensionalPreference
MultidimensionalPreference ::= CascadedPreference
                               ('AND' CascadedPreference)*
CascadedPreference ::= AtomicPreference
                       ('CASCADE' AtomicPreference)*
AtomicPreference ::= BooleanPreference | HighestPreference |
                     LowestPreference
BooleanPreference::= Expression
HighestPreference::= 'HIGHEST' Expression
LowestPreference ::= 'LOWEST' Expression
```

Interpretation. Not ceteris paribus.

How it works. The SPARQL Preference syntax allows modelling two kinds of preferences and two kinds of preference interaction, whose semantics is described informally as follows. For a formal definition, we refer the reader to elsewhere (Siberski et al. 2006).

- **Boolean preferences**. Boolean preferences are specified by a boolean expression over solutions of an ontology defined in OWL DL (Patel-Schneider et al. 2004). An option S_i dominates an option S_j, if the boolean expression is evaluated to true for S_i, and false for S_j.
- **Scoring preferences**. This kind of preference is specified by an expression, which evaluates to a number or a value in other SPARQL domains that have total ordering. They express the preference for maximising or minimising a certain value. Therefore, an option S_i dominates S_j, if S_i has a higher (lower) value than S_j, when a `HighestPreference` (`LowestPreference`) is defined.
- **Multidimensional Preferences**. This kind of preference interaction indicates that two preferences must be addressed in a combined form. For any solutions S_i and S_j, the domination relation to combine independent preferences $\succ_{|C1\ AND\ C2|}$ establishes that S_i is dominated by S_j in neither $C1$ nor $C2$, and that S_i dominates S_j in either $C1$ or $C2$.
- **Cascaded Preference**. A cascaded preference indicates that a preference $C1$ has a higher priority than $C2$. So, for any options S_i and S_j, the domination relation to combine prioritised preferences $\succ_{|C1\ CASCADE\ C2|}$ states that either S_i dominates S_j according to $C1$, or they are incomparable according to $C1$, and S_i dominates S_j according to $C2$.

Based on the definition of dominance among options, a SPARQL Preference query returns as result all non-dominated solutions (*skyline queries*). If a solution S_i dominates S_j according to a preference $C1$, but S_j dominates S_i according to a preference $C2$, both options are presented as the result of the query.

Limitations. The major problem of this approach occurs when a best option is not found, according to stated preferences, which is a scenario that typically occurs, because trade-off among conflicting preferences is very common. For instance, if a user wants to buy a laptop and she states that she prefers to minimise price and maximise performance. The typical case is that price increases as the performance is better and, therefore, according to these preferences no laptop dominates another

(there is none, or only a few laptops, that are more expensive and have worse performance), and consequently the approach cannot select among these laptops.

5.7 Preferences in Argumentation Frameworks

Based on the different presented works, it can be seen that reasoning about preferences has been target of research in different areas of computer science, including AI, databases, and semantic web. Recently, this research topic has been investigated in the context of argumentation (Rahwan and Simari 2009), which can be abstractly defined as the interaction of different arguments for and against some conclusion. Amgoud et al. (Amgoud et al. 2008) discuss how to make decisions through preference-based argumentation, but they basically illustrate an application of the use of argumentation frameworks for deciding between two options, and there is no explicit representation of preferences. As a consequence, this work is not detailed in this section. Modgil (Modgil 2009) extended the classical Dung's argumentation theory (Dung 1995) in order to incorporate arguments that claim preferences between other arguments, thus incorporating metalevel-argumentation-based reasoning about preferences in the object level. This proposed extension is thus detailed next.

Goals. A preference representation model — Extended Argumentation Framework (EAF) — and semantics for identifying which are the sets of justified arguments, i.e. the possible extensions of a given EAF.

Input. The required input of this approach is an Extended Argumentation Framework (EAF), which is an extension of the argumentation framework proposed by Dung (Dung 1995). An argumentation framework models arguments, abstracting from the underlying logic that represents them, as a directed graph in which arguments are represented as nodes and a defeat relation is represented as arrows. This structure is then used for determining which arguments are the justified. EAFs accommodate arguments that claim preferences between other arguments, i.e. they explicit capture the notion of preferences in argumentation frameworks. Formally, an EAF is defined as follows.

An Extended Argumentation Framework (EAF) is a tuple $(Args, \mathcal{R}, \mathcal{D})$ such that $Args$ is a set of arguments, and:

- $\mathcal{R} \subseteq Args \times Args$,
- $\mathcal{D} \subseteq Args \times \mathcal{R}$,
- If $(X, (Y, Z)), (X', (Z, Y)) \in \mathcal{D}$ then $(X, X'), (X', X) \in \mathcal{R}$.

Interpretation. The preference relation proposed by Modgil (Modgil 2009) addresses preferences over attacks, and not over (a pair) of outcomes, as it is the case of the other works presented in this chapter. Therefore, the interpretation of preference statements does not apply for this work.

How it works. The main extension of EAFs is the inclusion of a second attack relation $\mathcal{D}$ that ranges from arguments X to attacks $(Y, Z) \in \mathcal{R}$, where $\mathcal{R}$ is the standard binary attack relation in a Dung framework. If $(X, Y), (Y, X) \in \mathcal{R}$, i.e. arguments X and Y attack each other, and there is a $(Z, (Y, X)) \in \mathcal{D}$, we can say that the attack (X, Y) is preferred to (Y, X). Additionally, preference arguments expressing contradictory preferences attack each other, where these attacks can then themselves be attacked by preference arguments. Preferences are not defined by some externally given preference ordering, but are themselves claimed by arguments.

As Dung's argumentation framework was extended, it is essential to redefine some concepts, e.g. when a set is conflict free and the acceptability of arguments (we refer the reader to Dung's work (Dung 1995) for these definitions), i.e. the semantics underlying Dung's framework should also be extended. Modgil starts by defining when an argument A defeats another argument B, which are in $S \subseteq Args$. It occurs when there is an $(A, B) \in \mathcal{R}$ and there is no argument $C \in S$, such that $(C, (A, B)) \in \mathcal{D}$. It is said that $A\ defeat_S\ B$. Conflict free set is intuitively defined as follows. If $A, B \in S$ ($S \subseteq Args$) and A attacks B, then S is *conflict free* only if B does not attack A and there is a C that defence attacks the attack from A to B. Finally, an argument $A \in Args$ is *acceptable* with respect to $S \subseteq Args$, if and only if for all B such that $B\ defeat_S\ A$, there is a $C \in S$ such that $C\ defeat_S\ B$ and there is a reinstatement set for $C\ defeat_S\ B$ (reinstatement set of an argument X is a set of arguments that defeats any argument that attacks X). With these definitions, the admissible, preferred, complete and stable extensions of an EAFs are defined in the same way as for Dung's framework.

Limitations. This work (Modgil 2009) has a different purpose in comparison of other works presented in this section, in that it proposes a general framework for supporting decision making, which is not restricted to choosing a preferred outcome from a set or a pair of outcomes. Therefore, it is not clear how the typical preference statements (discussed in other sections) can be represented and, with them, answering the typical questions related to preference reasoning. In addition, complexity is not a topic discussed, because, with the definition when an argument defeats another, any algorithm to identify extensions of argumentation frameworks can be used. However, only certain kinds of extensions

of argumentation frameworks with particular graph structures can be identified in polynomial time.

5.8 Discussion

In this section, we present a comparison of the presented approaches and discuss relevant points related to them. As stated in the introduction, our goal is not to rate these approaches to choose the best one, but to better understand them and their relationship. We start by summarising the key characteristics of the approaches for reasoning about preferences, which are depicted in Table 5.4. For each approach, we show six different aspects, which are presented in respective columns.

- **Area**: indicates to which research area the approach is mainly related.
- **Qualitative/Quantitative Representation and Reasoning**: classifies the representation and the reasoning method adopted by the approach as quantitative or qualitative.[3] Quantitative approaches represent preferences as a number (or a function that produces a number), such as ratings that capture the value of an option. Qualitative approaches, on the other hand, allow comparing outcomes without specifically stating how much an outcome or attribute value is preferred to another ("*I prefer X to Y*") or qualitatively evaluate them ("*I like X very much*").[4] *Representation* is related to how preferences captured from users are represented. This representation can be directly manipulated or can be transformed into another representation to be used by the algorithm of the approach (*reasoning*). For instance, the approach "Graphically Structured Value Function Compilation" adopts a qualitative representation (CP-net or TCP-net) but converts it to a quantitative representation (value function) for reasoning about the provided preferences.
- **Goals**: the goals shown in this table follow the classification introduced in Section 5.1. The acronyms stand for: *PRM* — Preference Representation Model; *ALGO* — Algorithms; *ALGE* — Algebra; *SEM* — Semantics; *TRANS* — Transformation.
- **Questions**: the questions shown in this table follow the classification introduced in Section 5.1. The acronyms stand for: *OO* — Outcome Optimisation; *DQ* — Dominance Query; *OQ* — Ordering Query; *NDO* — Non-dominated outcomes.

[3]Table 5.4 shows this term in italics in order to make a better visual distinction between the terms qualitative and quantitative.

[4]This distinction between quantitative and qualitative preferences is widely adopted by researchers on preference reasoning, even though an ordinal scale is still quantitative, but imprecise.

– **Input**: provides a general classification for the required input of the approach. It can be utility (or value) functions, a particular structure (e.g. CP-nets), qualitative preference statements (we do not specify here of which kinds), or quantitative preference statements (we do not specify here the domain used to rate outcomes or attribute values).
– **Interpretation**: indicates how preference statements are interpreted by the approach. There are three possible options: (i) *CP* (Ceteris Paribus) — when the approach uses the "all else being equal" interpretation for statements; (ii) *NCP* (Not Ceteris Paribus) — any other possible interpretation for provided statements, e.g. in the approach "Learning Utility Functions with SVM" statements have a marginal contribution for the utility function; and (iii) *N/A* (Not Applicable) — this is only the case of "Preferences in Argumentation Frameworks." As in this approach arguments are abstract and are not necessarily preference statements, it is not possible to state a specific interpretation adopted by the approach.

By analysing these approaches, we can observe that they address three main problems: (i) given a representation of qualitative statements, how can the typical questions (outcome optimisation and dominance queries) related to reasoning about preferences be answered? (ii) given a representation of qualitative statements, how can it be transformed into a utility function? and (iii) how can we represent compact utility functions? In problem (i), preferences are considered easier to elicit than in quantitative approaches, as qualitative statements are considered closer to the user vocabulary, but are difficult to be reasoned about. Problem (iii) is related to approaches that rely on utility functions to reason about preferences, and with them one can easily compare outcomes. However, eliciting utility functions is not a trivial task, and for particular contexts, combinations of attribute values have utility values that cannot be represented in a compact form, such as assuming utility independence among attributes. Therefore, problem (ii) aims at obtaining the main benefits of (i), i.e. easy representation, and (iii), i.e. easy reasoning, but the challenge now is the translation of qualitative to quantitative statements. In addition, considering all possible combinations of attribute values is a combinatorial problem, therefore it is very important to take into account the complexity of reasoning about preferences. We do not discuss complexity in this section, as the worst case takes exponential time to be computed for almost all approaches, and some of them do not provide complexity analysis.

All the approaches aim at helping users to make decisions of choosing from among alternatives, and we can classify this aim into two categories: (a) helping users to make decisions in situations that they are confused and it is hard for them to compare outcomes and identify the best one; and (b) helping users to make

Table 5.4: Comparison among Approaches to Reason about Preferences.

Approach	**Area**	**Quantitative/Qualitative**		**Goals**	**Questions**	**Input**	**Inter-pretation**
		Representation	**Reasoning**				
CUI Networks (Engel and Wellman 2008)	AI	Quantitative	Quantitative	PRM, ALGO	OO	Utility Functions	NCP
Utility Functions for Ceteris Paribus Preferences (McGeachie and Doyle 2008)	AI	*Qualitative*	Quantitative	TRANS		Quantitative Preferences	CP
Learning Utility Functions with SVM (Domshlak and Joachims 2007)	AI	*Qualitative*	Quantitative	TRANS		Quantitative Preferences	NCP
Semiring-based Constraint Satisfaction (Bistarelli et al. 1997)	Constraint Programming	Quantitative	Quantitative	PRM	OO	Quantitative Preferences	NCP
Preference-based Problem Solving for Constraint Satisfaction (Junker 2008)	Constraint Programming	*Qualitative*	Quantitative	PRM, ALGO	OO	Soft Constraint Satisfaction Problem	NCP
CP-nets (Boutilier et al. 2004)	AI	*Qualitative*	*Qualitative*	PRM, ALGO	OO, DQ, OQ	CP-net	CP
Combining CP-nets and Soft Constraints (Domshlak et al. 2003)	AI	*Qualitative*	Quantitative	TRANS	DQ	Acyclic CP-net	CP
UCP-networks (Boutilier et al. 2001)	AI	*Qualitative*	Quantitative	PRM, ALGO	OO, DQ	UCP-net	CP
TCP-nets (Brafman et al. 2006)	AI	*Qualitative*	*Qualitative*	PRM, ALGO	OO, DQ	TCP-net	CP
Graphically Structured Value Function Compilation (Brafman and Domshlak 2008)	AI	*Qualitative*	Quantitative	TRANS		CP-net or TCP-net	CP
Scoring Function (Agrawal and Wimmers 2000)	Databases	Quantitative	Quantitative	PRM, ALGO	OQ	Quantitative Preferences	NCP
Winnow (Chomicki 2003)	Databases	*Qualitative*	*Qualitative*	PRM, ALGO, ALGE	NDO	Qualitative Preferences	NCP
BMO Query Model (Kießling 2002)	Databases	*Qualitative*	*Qualitative*	PRM, ALGE	NDO	Qualitative Preferences	NCP
Query Personalisation based on Preferences (Koutrika and Ioannidis 2006)	Databases	Quantitative	Quantitative	PRM, ALGO	OQ	Quantitative Preferences	NCP
OWLPref (Ayres and Furtado 2007) SPARQL Preference (Siberski et al. 2006)	Semantic Web Databases	*Qualitative*	*Qualitative*	PRM, SEM	NDO	Qualitative Preferences	NCP
Preferences in Argumentation Frameworks (Modgil 2009)	Argumentation (AI)	*Qualitative*	*Qualitative*	PRM, SEM		Extended Argumentation Framework	N/A

decisions in situations that they know (or have an idea of) what is best, but the set of possible outcomes is so large that this task becomes time-consuming and requires too much effort of the user. Even though, in the end, the problem to be solved is the same in these two categories, they differ in a very relevant aspect: engagement of users in providing information about their preferences. A typical scenario for (a) is a company manager that has to decide an action to be taken, and this action has a crucial impact in the profit of the company. In addition, many attributes with uncertain values are associated with this action. Therefore, this manager is willing to spend a significant amount of time to precisely specify preferences among options and related attributes. And a typical scenario for (b) is web users that make searches with keywords and receive in return a large amount of results, which they have to filter and rank.

MAUT has emerged with the goal described in (a), i.e. helping a confused decision-maker to evaluate complex alternatives when their outcomes are uncertain and have several relevant attributes. Therefore, adopting utility functions to represent preference is reasonable (but still difficult), as users are willing to spend time in eliciting them, i.e. when the consequences of making a wrong decision compensates the time and effort spent in eliciting (and building new) preferences. Even though there is work (McGeachie and Doyle 2008, Domshlak and Joachims 2007) that obtains utility functions from qualitative statements, it still has limitations reported in the approaches presented in this chapter.

Query-based approaches address the scenario described in (b), and their aim is not to totally automate the decision making process, but to provide a reduced amount of results to users, possibly ranked. Considering preferences in queries allows users to be more restrictive when providing constraints for queries and still obtaining results that contain useful information — if the query is over constrained it is likely that to return no result. Therefore, this family of approaches aim at identifying non-dominated outcomes, and those dominated are discarded as they have no advantage with respect to the non-dominated outcomes. Choosing among non-dominated outcomes is a task that is still left for the user, but some heuristics, such as considering lexicographic importance among attributes, can be used to rank them, and give guidance to the users. There are two approaches (Agrawal and Wimmers 2000, Koutrika and Ioannidis 2006) that are quantitative, but for defining a total order of outcomes requires making assumptions, such as independence among attributes. Moreover, expecting quantitative values from users may require sophisticated elicitation processes.

In order to deal with over constrained problems, CSPs have been extended to incorporate constraints that can remain unsatisfied, with a no-satisfaction penalty

associated with them. These approaches have the objective to minimise the penalty of unsatisfied constraints (or maximising preferences), and other objectives, e.g. minimise price, cannot be specified. In addition, trade-off situations among conflicting objectives cannot be modelled either. SCSPs can help in solving a significant class of problems, e.g. meeting scheduling problems, but there are other kind of problems that cannot be addressed, such as those that include multi-objectives and trade-off.

Finally, there are the graphically-structured approaches, which structure qualitative preference statements in a graph and adopt the ceteris paribus interpretation. For graphs with certain properties, these approaches can be efficiently used to define optimal outcomes and, with the proposed techniques (Domshlak et al. 2003), can also answer dominance queries efficiently. However, two outcomes can be compared only when "all else is equal," and in practice it is often not the case. When other attribute values differ, outcomes are considered incomparable and no decision can be made regarding dominance. Moreover, these approaches can deal only with discrete attribute domains, and this is also a very restrictive assumption. Furthermore, trade-off is only captured in TCP-nets, but a lexicographic approach is adopted, unless a fully specified CIT is provided.

5.9 Final Considerations

In this chapter, we presented a systematic review of research work that aims at helping and automating decision makers to make choices taking into account their preferences. We were not limited to a specific research area, and included works in the context of decision theory, artificial intelligence, constraint programming, databases and semantic web. Each of these works was presented following an evaluation framework, which facilitates their comparison. Our review allows understanding not only each individual work but also, with our discussion, how they are related, which kind of issues they typically address and their limitations.

This study shows that a main issue related to reasoning about preferences is dealing with trade-off situations. Both utility functions and (conditional) qualitative statements can capture them, but it is difficult to express this kind of preferences in a compact form. For instance, if additive utility independence is not identified among attributes, each possible outcome will have a particular utility that cannot be derived from individual attribute values. And, in the case of qualitative statements, each full assignment for attributes must be compared. Moreover, users typically do not provide these kind of preferences, because they are constructed just when users face a concrete decision making situation (Lichtenstein and Slovic 2006).

Moreover, many of the presented approaches are able to reason only about a

restricted set of preferences, thus constraining users while expressing preferences. Dealing with heterogeneous types of preferences, as those of our preference metamodel, which include natural-language-like expressions, is not a trivial task. In next chapter, we present a decision making technique to address these issues of existing work.