6
An Automated Decision Maker with User-centric Principles

In order to tackle the problems of the existing approaches to reasoning about preferences discussed in the introduction and in the previous chapter, we present in this chapter a novel technique for automated decision making based on preferences and available options. This technique is able to handle qualitative preferences expressed in a high-level language and incorporates psychological processes to simulate human decision to resolve trade-offs. Our technique thus chooses one option from a finite set available, based on user preferences that have natural-language-like expressions, such as expressive speech acts (e.g. *like*, *accept* or *need*).

Section 6.1 describes the scope of decisions we are addressing and the assumptions of this work. Section 6.2 presents the high-level preference language that our technique is able to process, which is based on our preference metamodel. We also introduce a running example that exemplifies the use of this language and that will be used throughout the chapter. Next, Section 6.3 describes our technique, whose steps are detailed from Section 6.4 to 6.7. We compare the technique with existing work and present its evaluation in Section 6.8, concluding in Section 6.9.

6.1
Scope and Assumptions

As introduced earlier, our goal is to provide users with an automated decision making technique, which is able to make decisions on their behalf so as to automate their tasks. As decisions may be characterised in many different ways, we define in this section the *scope* of the decision problems we are dealing with. A decision problem consists of choosing one option o based on preferences from a finite set of available options, Opt, where all $o' \in Opt$ are of the same concept, for example a set of *laptops* or a set of *hotels*. Each concept is associated with a finite set of attributes, Att, and each $a_i \in Att$ is associated with a domain D_i, which establishes the values allowed for that attribute. Each domain D_i: (i) consists of a set of values x_{ij}; (ii) can be *discrete* or *continuous*; and (iii) can be *ordered* or *unordered*. For example, real numbers are an ordered continuous domain, integers are an ordered discrete domain, and colours are a unordered discrete domain. We refer to domains

composed of numbers as *numeric*. This way of describing options is a *restricted view* of our ontology model presented in Chapter 3. We left out of the scope of our technique the ability of handling: (i) *adjectives*; (ii) *scales*; and (iii) *proxy attributes*.

Our focus is on decisions that users are able to make themselves, but as these decisions may involve a large number of options, are tedious, and possibly repetitive for users, they prefer to delegate them instead of demanding time and effort in their execution — and they want the system to choose the option they would choose themselves. It is important to highlight that this is different from the goal of MAUT (Keeney and Raiffa 1976), which is a classical approach that was proposed to help people to make *critical* decisions with conflicting preferences, and requires user interaction to identify a function that represent preferences quantitatively.

Given the scope we are addressing, we now detail our *assumptions*. First, users have a set of preferences over the problem that the decision has to be made, i.e. their known preferences. Second, we consider a *unitary decision maker*, that is, provided preferences are given by a single user, and the goal is to increase the satisfaction of this user with the choice. Third, we assume a *consistent* set of preferences. User may provide conflicting preferences, e.g. "I prefer higher quality and lower price," but they are not inconsistent, such as "I prefer A to B, and I prefer B to A." Finally, available options, attributes and their domains are *given* and are, therefore, inputs of our technique, together with user preferences.

6.2 Preference Language and Running Example

As discussed in our study of how humans express preferences (Chapter 2), there are different forms in which they do so, and our goal is to provide them with a language in which they can inform their preferences in a way as close as possible to natural language. Based on our preference metamodel (Chapter 3), we derived a high-level preference language, whose EBNF is presented in Table 6.1, which includes seven types of preferences and means of specifying priorities among attributes and preferences. While constraints, qualifying and rating are monadic preferences, goals, orders and indifferences are dyadic.

The different types of preferences in our language have the same meaning of the corresponding types in our preference metamodel. However, there are some restrictions to the metamodel, which are constructions left unaddressed by our technique. These restrictions are associated with preference targets that cannot be used. We next present them, and also discuss how these limitations could be addressed. We highlight that most of these restrictions, and also those related to preference targets mentioned above, can be held at a higher level of abstraction: features in a user interface can allow users to express preferences that are translated

$$
\begin{aligned}
\mathit{preference} &::= [\mathit{condition}]\,(\mathit{constraint} \mid \mathit{qualifying} \mid \mathit{rating} \mid \mathit{goal} \\
&\quad \mid \mathit{order} \mid \mathit{indifference} \mid \mathit{dontCare}) \\
\mathit{condition} &::= \textbf{if}\ \mathit{formula}\ \textbf{than} \\
\mathit{formula} &:: \mathit{expression} \mid \mathit{formula}\ \textbf{and}\ \mathit{formula} \\
&\quad \mid \mathit{formula}\ \textbf{or}\ \mathit{formula} \mid \textbf{not}\ \mathit{formula} \\
\mathit{expression} &::= \mathit{attribute}\ (= \mid \neq \mid > \mid \geq \mid < \mid \leq)\ \mathit{value} \\
\mathit{constraint} &::= \mathit{formula} \\
\mathit{qualifying} &::= \mathit{expressive\_speech}_a\ \mathit{ct}\ \mathit{formula} \\
\mathit{rating} &::= \mathit{formula}\ \mathit{rate} \\
\mathit{goal} &::= (\textbf{minimise} \mid \textbf{maximise})\ \mathit{attribute} \\
\mathit{order} &::= \mathit{attribute} = \mathit{value} \succ \mathit{attribute} = \mathit{value} \\
\mathit{indifference} &::= \textbf{indifferent}\ \mathit{formula}\ \{\mathit{formula}\} \\
\mathit{dontCare} &::= \textbf{dont\_care}\ \mathit{attribute} \\
\mathit{expressive}_s\mathit{peech}_a\,\mathit{ct} &::= [\textbf{don't}]\,(\textbf{prefer} \mid \textbf{need} \mid \textbf{desire} \mid \textbf{avoid} \mid \textbf{like} \mid \textbf{want} \\
&\quad \mid \textbf{accept} \mid \textbf{require} \mid \textbf{love} \mid \textbf{hate}) \\
\mathit{rate} &::= \textbf{best} \mid \textbf{very\_good} \mid \textbf{good} \mid \textbf{neutral} \mid \textbf{bad} \\
&\quad \mid \textbf{very\_bad} \mid \textbf{worst} \\
\mathit{priority} &::= [\mathit{condition}]\,(\mathit{attribute\_priority} \\
&\quad \mid \mathit{attribute\_indifference} \mid \mathit{preference\_priority}) \\
\mathit{attribute\_priority} &::= \mathit{attribute} \rhd \mathit{attribute} \\
\mathit{attribute\_indifference} &::= \mathit{attribute} \sim \mathit{attribute} \\
\mathit{preference\_priority} &::= \mathbb{Z}.\ \mathit{preference}
\end{aligned}
$$

Table 6.1: Preference language.

to a set of preferences in our language.

- *Preferences over enumeration values.* Referring to an enumeration value, such as *I prefer red*, is to establish preferences over values of an enumeration in a generic way (e.g. the *colour* preference indicates that I prefer *red cars*, *red t-shirts* or *red coats*), and this is not allowed. However, our restricted language still allows users to express preferences over enumeration values, but preferences should be expressed in the context of each specific attribute, as they cannot be expressed generically.
- *Preferences over concepts*. Preferences over concepts, such as "*I prefer LCD to CRT monitors,*" are also not allowed. Preferences over concepts indicate a *is-a* relationship ("*LCD is-a monitor*"). Thus, in order to support concepts, the language would have to be extended with a new comparison operator (*is-a*) to be used in expressions.
- *Preferences over instances.* An instance represents a particular entity, which can be defined by assigning a value to each attribute of the entity class. Therefore, an instance can be represented in our restricted language with a

constraint preference, which makes a value assignment for an attribute that is used as an identifier.

– *Each preference is associated with a single attribute.* We restrict preferences to refer to only one attribute (this restriction is not extended to conditions). As a consequence, propositional formulae of constraints cannot refer to different attributes, e.g. "*I prefer a laptop with a 15" screen and integrated camera.*"

– *Order statements refer only to equality expressions.* Even though order preferences can refer only to equality expressions, some of the forbidden targets can be expressed in a different allowed manner, e.g. "*I prefer a laptop with a 14" or 15" screen to one with a 17" screen*" can be expressed with two preferences: "*I prefer a laptop with a 14" screen to one with a 17" screen*" and "*I prefer a laptop with a 15" screen to one with a 17" screen*."

In order to illustrate our high-level preference language, we introduce an example in this section. Throughout this chapter, this example will also be used to illustrate different parts of our decision making technique. Suppose *Bob* is visiting a university, and needs to choose an apartment at which to stay. Each apartment is described in terms of *seven* attributes, described below, each associated with a domain.

1. *uni*: $\{x \mid x \in \mathbb{R}, 0 < x \leq 15\}$ — the distance, in kilometres, from the apartment to the university.
2. *station*: $\{x \mid x \in \mathbb{R}, 0 < x \leq 1.2\}$ — the distance, in kilometres, from the apartment to the closest underground station.
3. *market*: $\{x \mid x \in \mathbb{R}, 0 < x \leq 0.7\}$ — the distance, in kilometres, from the apartment to the closest supermarket.
4. *zone*: $\{x \mid x \in \mathbb{Z}, 1 \leq x \leq 6\}$ — the zone where the apartment is located. The underground coverage in the city in which the university is located is split into six zones. Zone 1 is the city centre, and the higher the zone number is, the farther it is from the centre.
5. *brand*: { A, B, C, D } — each apartment has a brand, associated with the company it belongs to.
6. *stars*: $\{x \mid x \in \mathbb{Z}, 1 \leq x \leq 5\}$ — a number that represents the apartment quality; the higher, the better.
7. *price*: $\{x \mid x \in \mathbb{R}, 95 \leq x \leq 125\}$ — the price of renting the apartment (per week).

Bob's preferences are shown as follows, using our preference language, numbered by their priority (1...15), and with the final line being an attribute priority.

Table 6.2: Available apartments.

Apartment	brand	market	price	stars	station	uni	zone
Ap_A	A	0.45 Km	£100	2	0.3 Km	5.0 Km	2
Ap_B	D	0.40 Km	£115	3	0.6 Km	2.2 Km	1
Ap_C	B	0.20 Km	£95	2	0.3 Km	10.0 Km	3
Ap_D	B	0.60 Km	£105	2	0.5 Km	6.0 Km	2
Ap_E	B	0.30 Km	£100	3	0.5 Km	3.5 Km	2
Ap_F	C	0.40 Km	£125	4	0.9 Km	2.0 Km	1

1. **don't accept** $zone > 2$
2. **prefer** $uni \leq 2.5Km$
3. **if** $uni \leq 2.5Km$ **then need** $station \leq 1Km$
4. **if** $uni \leq 2.5Km$ **then prefer** $station \leq 0.7Km$
5. **if** $uni \leq 2.5Km$ **then require** $price \leq$ £125
6. **if** $uni > 2.5Km$ **then need** $station \leq 0.7Km$
7. **if** $uni > 2.5Km$ **then require** $price \leq$ £105
8. **minimise** $station$
9. **minimise** $market$
10. **minimise** $price$
11. **prefer** $brand = A$ **or** $brand = B$ **or** $brand = C$
12. $brand = A \succ brand = B$
13. $brand = B \succ brand = C$
14. $stars = 2$ **good**
15. **maximise** $stars$

    – **if** $uni > 2.5Km$ **then** $station \rhd uni$

These preferences and priorities are used to make a choice on Bob's behalf. The decision problem is to choose one apartment from the available options, shown in Table 6.2.

6.3 Technique Overview

The goal of our decision maker is to simulate human reasoning in making decisions, allowing us to exploit natural user expressiveness of preferences (without the need for elicitation methods) and resolve trade-offs (that cannot be resolved with the provided preferences) in a way humans would do. As a result, we propose a decision making technique that is based on heuristics investigated in psychology that explain how people make decisions. More specifically, the overall process is inspired by *reason-based choice* (Shafir et al. 1998), which discusses that people make decisions by identifying reasons to accept and reject options, and

incorporates two principles (Simonson and Tversky 1992): *extremeness aversion* (avoiding extreme options, which are those that compromise too much an attribute because of another that provides a gain), and *trade-off contrast* (influencing the preference between two options with the cost-benefit relationship of all options).

We introduce our technique by showing the different steps, process and data that comprise it, presented in Figure 6.1. We make three main observations on this figure. First, it can be seen that monadic preferences, which indicate preferences with expressive speech acts or rates, are used in many processes of our technique, showing that the technique is *driven by these natural-language expressions*. Second, the technique has *variable parts*: as our technique use a particular interpretation of natural language expressions, this interpretation can differ in different applications. Moreover, as we discuss later, during the decision making process our technique calculates quantitative costs of options based on qualitative preferences, and different functions associated with this calculation can be adopted. Based on experimentation, we selected particular instances (adopted in this thesis) for these parts. Third, our technique is composed of *four main steps*, explained next.

Pre-processing. Our preference language allows the expression of heterogeneous types of preferences, and an integrated view that shows how they interact and how they evaluate individual option attributes is helpful to make a choice. The pre-processing step involves building computational models that compile information given by different preferences provided by users and represent options in a way that their positive and negative aspects are made explicit (according to those preferences). The first is the Preference Satisfaction Model (PSM), which combines the information given by monadic preferences, and the second, the Options-Attribute Preference Model (OAPM), indicates which attribute value is better considering two options.

Explication. Sometimes a preference provided by a user implies a further preference in addition to its literal meaning. For example, preference 14 may indicate that not only the preference is for 2-star apartments, but the closer the number of stars of the apartment is to 2 the better. So, in the explication step we consider *implicit* preferences that we can extract from the preferences explicitly given by users, and based on this information we update the previously produced OAPM.

Elimination. When people make a choice from a set, they first eliminate options that have no advantage when compared to another, i.e. a *dominated* option, or attributes with unacceptable values, i.e. non-compensatory attributes. In the elimination step, we discard these two kinds of options. While the OAPM

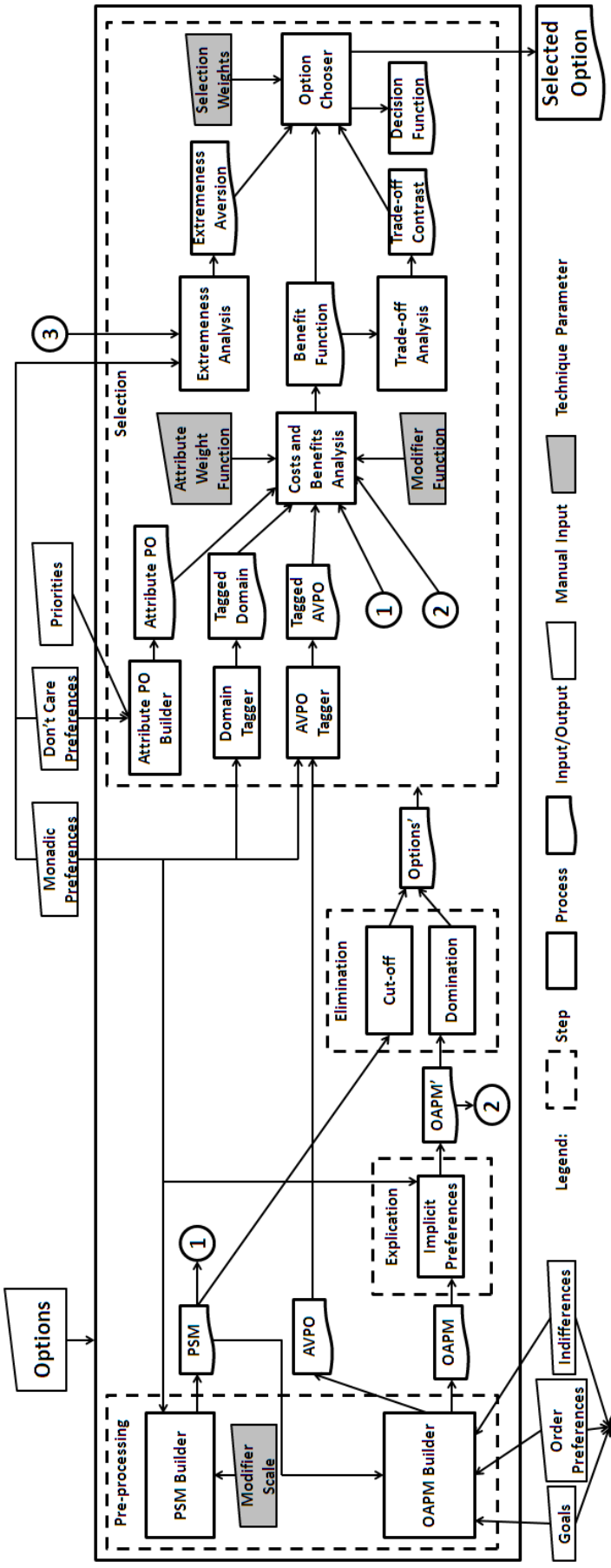

Figure 6.1: Technique Overview.

allows detecting dominated options in an easy way, as it shows the positive and negative aspects of each option compared to another, the PSM exposes options that do not satisfy hard constraints. Hard constraints in our technique are expressed using specific modifiers: *require*, *need*, *hate* and *don't accept* (this is subject to particular interpretations).

Selection. After eliminating the options above, we obtain a *consideration set*, which contains options that require trade-off resolution to make a choice. In order to make this selection, we first analyse option costs and benefits, by using the information compiled in our computational models to calculate an option costs with respect to another for each attribute. The overall costs of an option (w.r.t. another) is then a weighted sum of these individual attribute costs, by considering the provided priorities. Next, the trade-off between options and how these options compensate advantages with disadvantages are analysed (which are related to the trade-off contrast and extremeness aversion principles), and these factors are them combined with the previously calculated option costs.

It is important to highlight that, as opposed to many existing approaches, our technique does not focus on isolating two options and comparing them, as we use the whole set of options to evaluate preference between any two options. Our technique results in a partially ordered set, organised in four different levels, as described below.

1. **chosen option**, which is considered the optimal option;
2. **acceptable options**, which are in the consideration set, but were not chosen;
3. **eliminated options**, which were discarded because of non-compensatory attribute(s); and
4. **dominated options**.

We further make an observation on how we interpret provided user preferences. As it can be seen in our systematic review (Chapter 5), the most common interpretation of preferences adopted by approaches for reasoning about preferences is *ceteris paribus* (all other things being equal or held constant) (Hansson 1996). Under this interpretation, a preference that compares values of one or more attributes is taken into account only for comparing two options whose attribute values that are not targets of the preferences are equal. For example, the preference "*I prefer silver cars to white cars*" is considered only for comparing two cars that have the same brand, power, etc., i.e. everything else but colour has to be equal. This interpretation is very limited, because the common scenario is that options differ in more than one, if not several, attributes.

We do not adopt a ceteris paribus interpretation, but we are careful while using preferences provided by users. When users provide preferences, they consider attributes in isolation, and when they are used for comparing two options, the typical scenario is that preferences conflict with each other. In these cases, a trade-off must be resolved, which is typically done by users only while facing concrete decision making situation (Lichtenstein and Slovic 2006). And for resolving these trade-offs that emerge from conflicting preferences, our technique uses a psychology-inspired approach, as discussed above.

We describe next the steps of our technique. We first describe the computational models that are built in the pre-processing step of our technique (Section 6.4), which are later refined with implicit preferences (Section 6.5). These computational models are used to eliminate options (Section 6.6) and to choose an option (Section 6.7).

6.4 Pre-processing

In our approach, the first step to make a decision is to pre-process the available options and analyse them according to the preferences provided by users. As previously introduced, there are monadic and dyadic preferences, and as the former have only a single referent and the latter allow making a pairwise comparison, we process them separately, building two models based on them — the Preference Satisfaction Model (PSM) (Section 6.4.1) and the Options-Attribute Preference Model (OAPM) (Section 6.4.2). These computational models do not allow to conclude which available option is the "best" or decide which of two options is better, but they expose positive and negative aspects of available options, integrating the information provided by heterogeneous types of preferences.

6.4.1 Preference Satisfaction Model

The first model, named *Preference Satisfaction Model (PSM)*, consists of a table that captures how options satisfy preferences in terms of each attribute according to monadic preferences. This table associates option attributes with an expressive speech act or a rate (or their negation), meaning that the preference for an attribute value of a particular option is represented by a specific expressive speech acts or rate (e.g. the $price$ of option Ap_A satisfies ***require***). $e \in ExpressiveSpeechAct$ is an expressive speech act that comes from qualifying preferences, while $r \in Rates$ is a rate (e.g. *love* and *hate*) that comes from rating preferences. Expressive speech acts and rates are collectively referred to as ***modifiers*** ($M = ExpressiveSpeechActs \cup Rates$). Constraint preferences, which

are associated with no modifier, are considered to be associated with an implicit modifier, namely *want*. All these preferences are referred to as monadic preferences ($MP = Constraints \cup QualifyingPreferences \cup RatingPreferences$). The PSM is defined as follows.

Definition 6.1 *The **Preference Satisfaction Model (PSM)** is a partial map from a pair $\langle option, attribute \rangle$ to a modifier or its negation, indicating the most representative modifier that indicates preference for an attribute value of an option.*

$$PSM : Opt \times Att \mapsto \{\epsilon, \neg\} \times M$$

Before describing the PSM construction in detail, we introduce auxiliary functions. Each constraint, qualifying or rating preference p is characterised by (i) a modifier, $mod(p)$, e.g. ***need***; (ii) a formula, $form(p)$, e.g. $station \leq 1$; and (iii) optionally a condition, $cond(p)$, e.g. $uni \leq 2.5$ (examples are given considering preference 3). As we restrict preferences to refer to a single attribute, $att(p)$ is the attribute that is the referent of the preference, e.g. $station$. An option may or may not satisfy a constraint, and $sat(formula, o)$ replaces variables from the provided $formula$ with attribute values from option o and evaluates the formula for a boolean value, e.g. $sat(station \leq 1, Ap\_B) = true$. It is used to check either whether a preference is applicable to an option, i.e. the preference condition is satisfied by the option, or whether the option satisfies the preference itself. Given this notation, we define when a preference is applicable to an option.

Definition 6.2 *A preference p is applicable to an option o, $App(p, o)$, if and only if*

$$\nexists cond(p) \vee (\exists\, cond(p) \wedge sat(cond(p), o))$$

For example, preferences 3, 4 and 5 are applicable only to options Ap_B and Ap_F. Therefore, for each option, there is a subset of monadic preferences that is applicable to it, and each of them is related to a particular attribute. We can thus associate a set of modifiers (or their negation) with each option attribute — $AttMod(MP, o, a)$ — as shown below.

$$\begin{aligned} &AttMod(MP, o, a) ::= \\ &\quad \{\langle \epsilon, m \rangle \mid m = mod(p) \wedge p \in MP \wedge a = att(p) \wedge App(p, o) \wedge sat(form(p), o)\} \\ &\quad \cup \{\langle \neg, m \rangle \mid m = mod(p) \wedge p \in MP \wedge a = att(p) \wedge App(p, o) \wedge \neg sat(form(p), o)\} \end{aligned}$$

$AttMod(MP, o, a)$ may be empty. In the case of Ap_F and attribute $station$, for example, we thus have the following attribute modifiers.

$$AttMod(MP, Ap\_F, station) = \{\langle \epsilon, \textbf{need} \rangle, \langle \neg, \textbf{prefer} \rangle\}$$

Now that we have available all modifiers that indicate preference for a particular attribute of each of the options, we wish to select the most representative one. Expressive speech acts and rates, widely used by people, have an interpretation that is subjective and specific for each individual. Although they may have different meanings, such as expressing requirement or acceptance, all modifiers also express a degree of preference. Modifiers are categorised as *positive*, indicating a preference for an attribute value; *neutral*, indicating indifference and acceptance for an attribute value; and *negative*, indicating a preference against an attribute value. In addition, modifiers of each category can be stronger relative to each other. For example, consider the following two preferences: "*I need an apartment whose price is lower than £125*" and "*I prefer an apartment whose price is lower than £100.*" *Need* is stronger in the sense that it tells what *has* to be satisfied. However, when we have two options, the first satisfying only what is needed (e.g. an apartment that costs £110) and the second satisfying what it is needed and preferred (e.g. an apartment that costs £90), *the degree of preference* of the second is stronger than the degree of preference of the first.

We adopt a particular ranking that captures this notion to indicate the degree of preference of modifiers, namely *modifier scale*, presented in Figure 6.2, and each subset of modifiers that represents (according to this particular scale) the same degree of preference is associated with an index, which will be used later. The interpretation of modifiers is subjective, and therefore the modifier scale is one of the variation points of our technique, as indicated in Figure 6.1 — it can be instantiated in different ways for individual applications, or even customised to individual users. As discussed before, some of the modifiers indicate hard constraints, and those that are positive modifiers (*require* and *need*) are considered less strong than the other positive modifiers for the reasons above. Moreover, the modifiers that indicate hard constraints are in the same modifier scale than the others for modularity reasons: hard constraints are relevant only for eliminating options, so distinguishing modifiers that indicate them from the others is irrelevant in the other steps of the decision making process.

If more than one monadic preference applies to an option attribute, we use the adopted modifier scale to choose among them as the most representative. There are two possibilities. The first case is when there is at least one monadic preference whose formula is satisfied. According to the modifier scale, from the

Category	Modifier	Index
positive	want	9
	love, best	8
	desire, very_good	7
	prefer, like, good	6
	need	5
	require	4
neutral	accept, don't require, don't avoid	3
	neutral, don't love, don't hate	2
	don't need, don't desire	1
negative	don't prefer	-4
	avoid,	-5
	don't like, bad	-6
	don't want, very_bad	-7
	hate, worst	-8
	don't accept	-9

Figure 6.2: Modifier scale.

neutral modifiers (*don't need*, *don't desire*) to the most positive (*want*) there is a stronger preference for an attribute value; and from the neutral modifiers (*accept*, *don't require*, *don't avoid*) to the most negative (*don't accept*) there is a stronger preference against an attribute value. In case there are both positive and negative modifiers, there is inconsistency, which is not taken into account by our technique. Therefore, the strongest modifier from those satisfied (i.e. $\langle \epsilon, m \rangle$) is chosen.

The second case is when there is no monadic preference whose formula is satisfied. Again, there are two possibilities. If there is at least one (not satisfied) monadic preference whose modifier is positive, the weakest one is chosen. For example, if one option attribute is associated with $\langle \neg, \textbf{prefer} \rangle$ and $\langle \neg, \textbf{require} \rangle$, the second is selected, meaning that not even the weakest preference is satisfied. In case there is no monadic preference whose modifier is positive, then the weakest one is chosen (where the weakest is *accept*, *don't require* and *don't avoid*, and the strongest is *don't accept*). Given this informal description of how modifiers are selected, we present Algorithm 1, which describes how the PSM is built using the modifier indices. The PSM of the presented apartment decision problem built according to the proposed algorithm is shown in Table 6.3.

With this model one can see pros and cons against each available option. Even though one of the options might have only positive values, such as *like* and *require*, it does not mean it is the best option, because it may have only minimum acceptable values, and the trade-off with other options might indicate that another option is better. Moreover, other preferences, not processed yet, provide additional information, and this is what we will consider next.

Algorithm 1: PSM Builder

Input: MP: monadic preferences; Opt: options; Att: attributes; $scale$: modifier scale
Output: PSM: preference satisfaction model

foreach *Option* $o \in O$ **do**
  foreach *Attribute* $a \in A$ **do**
    $x \leftarrow null$;
    $m^* \leftarrow null$;
    foreach $\langle \epsilon, m \rangle \in AttMod(MP, o, a)$ **do**
      if $m^* == null \vee$ |IndexOf($m, scale$)|>|IndexOf($m^*, scale$)| **then**
        $x = \epsilon$;
        $m^* \leftarrow m$;
    if $m^* == null$ **then**
      foreach $\langle \neg, m \rangle \in AttMod(MP, o, a) \wedge Positive(m)$ **do**
        if $m^* == null \vee$ IndexOf($m, scale$) < IndexOf($m^*, scale$) **then**
          $x = \neg$;
          $m^* \leftarrow m$;
    if $m^* == null$ **then**
      foreach $\langle \neg, m \rangle \in AttMod(MP, o, a)$ **do**
        if $m^* == null \vee$ IndexOf($m, scale$) > IndexOf($m^*, scale$) **then**
          $x = \neg$;
          $m^* \leftarrow m$;
    $PSM[o, a] \leftarrow \langle x, m^* \rangle$;
return PSM;

Table 6.3: PSM of the Apartment Decision Problem.

	Ap_A	**Ap_B**	**Ap_C**	**Ap_D**	**Ap_E**	**Ap_F**
uni	$\langle \neg, prefer \rangle$	$\langle \epsilon, prefer \rangle$	$\langle \neg, prefer \rangle$	$\langle \neg, prefer \rangle$	$\langle \neg, prefer \rangle$	$\langle \epsilon, prefer \rangle$
station	$\langle \epsilon, need \rangle$	$\langle \epsilon, prefer \rangle$	$\langle \epsilon, need \rangle$	$\langle \epsilon, need \rangle$	$\langle \epsilon, need \rangle$	$\langle \epsilon, need \rangle$
market						
zone	$\langle \neg, don't\ accept \rangle$	$\langle \neg, don't\ accept \rangle$	$\langle \epsilon, don't\ accept \rangle$	$\langle \neg, don't\ accept \rangle$	$\langle \neg, don't\ accept \rangle$	$\langle \neg, don't\ accept \rangle$
brand	$\langle \epsilon, prefer \rangle$	$\langle \neg, prefer \rangle$	$\langle \epsilon, prefer \rangle$	$\langle \epsilon, prefer \rangle$	$\langle \epsilon, prefer \rangle$	$\langle \epsilon, prefer \rangle$
stars	$\langle \epsilon, good \rangle$	$\langle \neg, good \rangle$	$\langle \epsilon, good \rangle$	$\langle \epsilon, good \rangle$	$\langle \neg, good \rangle$	$\langle \neg, good \rangle$
price	$\langle \epsilon, require \rangle$	$\langle \epsilon, require \rangle$	$\langle \epsilon, require \rangle$	$\langle \epsilon, require \rangle$	$\langle \epsilon, require \rangle$	$\langle \epsilon, require \rangle$

6.4.2 Options-Attribute Preference Model

The second model that will aid us in the decision making process, namely Options-Attribute Preference Model (OAPM), is a table that captures comparison relationships between two options, from a perspective of individual attributes. This model shows for which attributes an option is better or similar to another — or no conclusion can be made with the provided preferences. For each OAPM value, which compares an option o to an option o' with respect to an attribute a, there are four possible preference values:

(i) $+$: the attribute value of o is better than o', i.e. $o \succ_a o'$;
(ii) $-$: the attribute value of o is worse than o', i.e. $o' \succ_a o$;
(iii) $\sim$: the attribute value of o is as preferred as o', i.e. $o \sim_a o'$;
(iv) ?: no conclusion can be reached.

The preference values that associate attribute values of two options are derived from provided user preferences. So, besides storing this information, the OAPM also keeps track of the reason for a preferred value. The OAPM is thus as follows.

Definition 6.3 *The **Options-Attribute Preference Model (OAPM)** is a total map from $\langle option_1, option_2, attribute\rangle$ to a preference value, indicating which attribute value of these options is the preferred one, and a reason that indicates the (explicit or implicit) preference that lead to this conclusion.*

$$OAPM : Opt \times Opt \times Att \mapsto \{+, -, \sim, ?\} \times Reason$$

The possible values of *Reason* can be a preference (e.g. maximisation goal), the PSM, or an implicit preference, specifically: *psm*, *max*, *min*, *avpo*, *indiff*, $\langle upper, p\rangle$, $\langle lower, p\rangle$, $\langle around, p\rangle$, and $\langle interval, p\rangle$ (they will be later described). For example, as preference 9 indicates that Ap_B is better than Ap_A with respect to the attribute $market$, the OAPM values are: $OAPM[Ap\_A, Ap\_B, market]$ = $\langle -, min\rangle$ and $OAPM[Ap\_B, Ap\_A, market]$ = $\langle +, min\rangle$ (the reason for the OAPM value is a minimisation goal). The initial OAPM state consists of all values set to ? and therefore, unless there are preferences that compare two attribute values, no conclusion is reached.

Note that the OAPM value $OAPM[o_1, o_2]$ is dual to $OAPM[o_2, o_1]$. The specification of our technique sets and uses both OAPM values to make it easier to understand, but its implementation can be optimised by representing just one of the values.

In the pre-processing step, OAPM values are set based on two kinds of information: (i) goal, order and indifference preferences and (ii) the previously built PSM. This information is processed separately in a specific order — PSM, goals, order preferences and indifference preferences — which makes the relationship between two attribute values established by a subsequently processed preference possibly override the current information present in an OAPM value. This is because a user may have general preferences for an attribute, but also have preferences for specific cases, such as stating that an attribute should be minimised and then providing order preferences for specific attribute values. In addition, preferences may refine other preferences, for instance, according to one preference a set of attribute values are considered equally preferred (e.g. preference 11), and specific preferences establish an order among the preferred values (e.g. preferences 12 and 13).

The next sections describe how the OAPM is constructed, in a declarative way. Presented formulae are representation of rules, which indicate the values to be set in

Table 6.4: Index used to compare PSM values.

PSM value	PSM Index
$\langle \epsilon, modifier \rangle$	$IndexOf(modifier, scale)$
$\langle \neg, negative\ modifier \rangle$	0
$\langle \neg, neutral\ modifier \rangle$	-1
$\langle \neg, positive\ modifier \rangle$	-2

the model. Rules are applied sequentially (following the order they are presented) for all pairs of options and attributes, and a subsequent rule may override the values set by a previous applied rule.

PSM

Monadic preferences in isolation do not allow us to compare attribute values, but, with the PSM, these preferences are situated in a context, and we can conclude that a value that is considered *best* is better than a value that is *good*, for instance. This idea is investigated by Hansson (Hansson 1990), who discusses the interpretation of "*good*" and "*bad*" in terms of "*better*." Our modifier scale, initially used to select most representative modifiers, is now used to compare modifiers associated with different options.

Note that the indices presented in Figure 6.2 have a gap between negative and neutral modifiers. Besides adjusting indices to give opposite index values to positive and negative modifiers, this gap is used to associate indices with unsatisfied modifiers, i.e. $\langle \neg, m \rangle$. When there is no satisfied modifier for an attribute value, we have three situations: (i) $\langle \neg, positive\ modifier \rangle$, there is one or more positive modifiers to evaluate that attribute value, but it does not satisfy them; (ii) $\langle \neg, neutral\ modifier \rangle$, there is no unsatisfied positive modifier, but there is one neutral that is unsatisfied; (iii) $\langle \neg, negative\ modifier \rangle$, there is no unsatisfied positive or neutral modifier, but the attribute value also does not satisfy a negative one. Situation (iii) is better than (ii), which is better than (i). When there is a satisfied modifier associated with each of two attribute values being compared — i.e. for both, the PSM value is $\langle \epsilon, modifier \rangle$ — the strongest modifier indicates the preferred value (or equally preferred if both options have the same degree of preference according to the scale), i.e. the one with the highest index. We thus use the indices of the modifier scale with additional ones (those shown in Table 6.4) to compare attributes values. Again, this raking is a variable point of our technique, and subject to different interpretations.

With these additional indices, we now choose the PSM value associated with the highest index. This way of stating which attribute value is better causes satisfied positive and neutral modifiers to be better than any other unsatisfied one, and satisfied negative modifiers worse than any other unsatisfied one. This

interpretation is adopted because users typically explicitly state what they want or do not want, being the absence of preference an indifference (weaker than the provided indifference), as people usually remember how the experiences felt when they were at their peak (best or worst) (Schwartz 2005).

Given this approach of establishing a preference relationship between attribute values based on (un)satisfied modifiers, i.e. PSM values, we show the rules used to set the OAPM values, which are applicable only to PSM values that are not null. $PSMIndex(PSM[o, a], scale)$ returns the index of the PSM value according to Table 6.4. Remember that the OAPM value is composed of a preference value (+, −, ∼, ?) *and* a reason.

$$PSMIndex(PSM[o_1, a], scale) = PSMIndex(PSM[o_2, a], scale) \rightarrow OAPM[o_1, o_2, a] = \langle \sim, psm \rangle \tag{6-1}$$

$$PSMIndex(PSM[o_1, a], scale) > PSMIndex(PSM[o_2, a], scale) \rightarrow OAPM[o_1, o_2, a] = \langle +, psm \rangle \wedge OAPM[o_2, o_1, a] = \langle -, psm \rangle \tag{6-2}$$

With respect to $station$, Ap_B is thus considered better than Ap_F, as the $PSMIndex$ associated with $\langle \epsilon, prefer \rangle$ (PSM value of $Ap\_B, station$) is 6, while the $PSMIndex$ of $\langle \epsilon, need \rangle$ is 5 (PSM value of $Ap\_F, station$).

Goals

The next set of preferences that is processed is goals, which is restricted to attributes whose domain is ordered, and indicates that an attribute value is considered better when its value is higher (maximisation goals) or lower (minimisation goals) than another. The rules used to set (or change) OAPM values, which are shown below, use two additional functions: (i) $type(goal)$, which is max when the goal is a maximisation, and min when it is a minimisation; and (ii) $val(o, a)$, which returns value of the attribute a of option o.

$$\exists\, g.(g \in Goal \wedge att(g) = a \wedge App(g, o_1) \wedge App(g, o_2) \wedge val(o_1, a) = val(o_2, a) \rightarrow OAPM[o_1, o_2, a] = \langle \sim, type(g) \rangle) \tag{6-3}$$

$$\begin{aligned} &\exists\, g.(g \in Goal \wedge att(g) = a \wedge App(g, o_1) \wedge App(g, o_2) \\ &\wedge ((type(g) = max \wedge val(o_1, a) > val(o_2, a)) \\ &\vee (type(g) = min \wedge val(o_1, a) < val(o_2, a))) \rightarrow \\ &OAPM[o_1, o_2, a] = \langle +, type(g) \rangle \wedge OAPM[o_2, o_1, a] = \langle -, type(g) \rangle) \end{aligned} \tag{6-4}$$

For example, because of preference 9, the OAPM values associated with

Ap_C and $market$ are set to $\langle +, min \rangle$ with respect to all other options.

Order Preferences

We now proceed to order preferences, which are those that establish an order among two attribute values by explicitly stating the preferred value. Order preferences are transitive, e.g. with preferences 12 and 13 we can derive that brand A is preferred to C. Therefore, from order preferences, we can derive a partial order of attribute values, namely *attribute value partial order (AVPO)*. However, as order preferences may be valid only according to a given condition, two different options may satisfy the conditions of different order preferences. Therefore an AVPO is constructed only with preferences applicable to an option and is specific to a pair of option and attribute. The order preferences of our example have no condition, thus the same partial order (AVPO) is built for all the options with respect to the brand attribute.

An AVPO is a forest (or possibly a tree) $\langle N, A \rangle$, where N is a set of nodes and A is a set of arrows that link nodes. Each node consists of expressions of order preferences, in the form of $attribute = value$ (e.g., $brand = A$), and an arrow from a node to another represents that the source node is preferred to the sink node. Algorithm 2 shows how an AVPO is constructed for a particular option and attribute. If the output is not a forest, there is a case of inconsistency, which is out of scope. In our example, the order preferences lead to the following AVPO for the $brand$ attribute for all options: $brand = A \rightarrow brand = B \rightarrow brand = C$.

Algorithm 2: AVPO Builder

Input: o: option; a: attribute; $Order$: order preferences
Output: $AVPO$: $\langle N, A \rangle$

1 Set⟨AVPONode⟩ $N \leftarrow \emptyset$;
2 Set⟨Arrow⟩ $A \leftarrow \emptyset$;
3 **foreach** *Order preference* $op \in Order$ **do**
4 **if** $\texttt{App}(op, o) \wedge \texttt{att}(op) = a$ **then**
5 $N \leftarrow N \cup \{ \texttt{LHS}(op), \texttt{RHS}(op) \}$;
6 $A \leftarrow A \cup \{\langle \texttt{LHS}(op), \texttt{RHS}(op) \rangle\}$;
7 **return** $\langle N, A \rangle$;

An attribute value of an option is preferred to another according to an AVPO if there is a path from the first to the second (typical tree algorithms are used (Cormen et al. 2001)), for which we use the notation $ExistsPath(AVPO[o, a], val_1, val_2)$, where val_1 matches a node whose expression is $attribute = val_1$. As different AVPOs may establish different preference relationships, we consider an attribute value of option o_1 better than that of option o_2 if there is a path in the AVPOs of both options, as shown in the next rule.

$$ExistsPath(AVPO[o_1, a], val(o_1, a), val(o_2, a)) \\ \wedge ExistsPath(AVPO[o_2, a], val(o_1, a), val(o_2, a)) \rightarrow \\ OAPM[o_1, o_2, a] = \langle +, avpo \rangle \wedge OAPM[o_2, o_1, a] = \langle -, avpo \rangle \quad (6\text{-}5)$$

All options are associated with the same AVPO with respect to $brand$ (presented above), as the same order preferences are applicable to them. According to this AVPO, Ap_A is better than Ap_C, Ap_D, Ap_E and Ap_F, with respect to this attribute.

Indifference Preferences

As opposed to order preferences, indifference is not transitive. A typical example illustrates the reason for this: a person is indifferent to two cups of tea with a difference of 0.1g of sugar on it. If transitivity is adopted, two cups of tea, one with no sugar and another with 1Kg of sugar, by transitivity, are considered equally preferred.

An indifference preference consists of a set of formulae, establishing indifference for two options' attribute values that satisfy formulae of the same indifference preference, but only if the condition (if any) of the preference is satisfied by both options, as detailed as follows.

$$\exists\, i.(i \in Indifference \wedge att(i) = a \wedge App(i, o_1) \wedge App(i, o_2) \\ \wedge \exists f, f'.(f \in form(i) \wedge sat(f, o_1) \wedge f' \in form(i) \wedge sat(f', o_2)) \rightarrow \\ OAPM[o_1, o_2, a] = \langle \sim, indiff \rangle) \quad (6\text{-}6)$$

By applying all the OAPM rules to our apartment decision problem, we produce as result the OAPM presented in Table 6.5.

6.5 Explication

Preferences provided by users always have a literal meaning. For example, the literal meaning of preference 2 of our running example is that apartments that are less than $2.5Km$ away from the university are preferred to those that are farther away than that. This sentence can also provide further information: if a maximum desired value is provided, and no minimum value, one can conclude that lower values are in general preferred to higher values. In addition, as this is a soft-constraint, i.e. it can remain unsatisfied if other attributes compensate this loss, the closer an option is to satisfying the preference, the better. In the case of preference 2, it means that between two apartments, both farther away than $2.5Km$ from the university, the preferred one is the closer. Preferences that can be derived from other explicit preferences are referred to as *implicit preferences*. We

Table 6.5: OAPM of the Apartment Decision Problem.

	Ap_B	**Ap_C**	**Ap_D**	**Ap_E**	**Ap_F**
uni	$\langle -, psm\rangle$	$\langle \sim, psm\rangle$	$\langle \sim, psm\rangle$	$\langle \sim, psm\rangle$	$\langle -, psm\rangle$
station	$\langle +, min\rangle$	$\langle \sim, min\rangle$	$\langle +, min\rangle$	$\langle +, min\rangle$	$\langle +, min\rangle$
market	$\langle -, min\rangle$	$\langle -, min\rangle$	$\langle +, min\rangle$	$\langle -, min\rangle$	$\langle -, min\rangle$
zone	$\langle \sim, psm\rangle$	$\langle +, psm\rangle$	$\langle \sim, psm\rangle$	$\langle \sim, psm\rangle$	$\langle \sim, psm\rangle$
brand	$\langle +, psm\rangle$	$\langle +, avpo\rangle$	$\langle +, avpo\rangle$	$\langle +, avpo\rangle$	$\langle +, avpo\rangle$
stars	$\langle -, max\rangle$	$\langle \sim, max\rangle$	$\langle \sim, max\rangle$	$\langle -, max\rangle$	$\langle -, max\rangle$
price	$\langle +, min\rangle$	$\langle -, min\rangle$	$\langle +, min\rangle$	$\langle \sim, min\rangle$	$\langle +, min\rangle$

(a) Comparison with Ap_A

	Ap_A	**Ap_C**	**Ap_D**	**Ap_E**	**Ap_F**
uni	$\langle +, psm\rangle$	$\langle +, psm\rangle$	$\langle +, psm\rangle$	$\langle +, psm\rangle$	$\langle \sim, psm\rangle$
station	$\langle -, min\rangle$	$\langle -, min\rangle$	$\langle -, min\rangle$	$\langle -, min\rangle$	$\langle +, min\rangle$
market	$\langle +, min\rangle$	$\langle -, min\rangle$	$\langle +, min\rangle$	$\langle -, min\rangle$	$\langle \sim, min\rangle$
zone	$\langle \sim, psm\rangle$	$\langle +, psm\rangle$	$\langle \sim, psm\rangle$	$\langle \sim, psm\rangle$	$\langle \sim, psm\rangle$
brand	$\langle -, psm\rangle$	$\langle -, psm\rangle$	$\langle -, psm\rangle$	$\langle -, psm\rangle$	$\langle -, psm\rangle$
stars	$\langle +, max\rangle$	$\langle +, max\rangle$	$\langle +, max\rangle$	$\langle \sim, max\rangle$	$\langle -, max\rangle$
price	$\langle -, min\rangle$	$\langle -, min\rangle$	$\langle -, min\rangle$	$\langle -, min\rangle$	$\langle +, min\rangle$

(b) Comparison with Ap_B

	Ap_A	**Ap_B**	**Ap_D**	**Ap_E**	**Ap_F**
uni	$\langle \sim, psm\rangle$	$\langle -, psm\rangle$	$\langle \sim, psm\rangle$	$\langle \sim, psm\rangle$	$\langle -, psm\rangle$
station	$\langle \sim, min\rangle$	$\langle +, min\rangle$	$\langle +, min\rangle$	$\langle +, min\rangle$	$\langle +, min\rangle$
market	$\langle +, min\rangle$	$\langle +, min\rangle$	$\langle +, min\rangle$	$\langle +, min\rangle$	$\langle +, min\rangle$
zone	$\langle -, psm\rangle$	$\langle -, psm\rangle$	$\langle -, psm\rangle$	$\langle -, psm\rangle$	$\langle -, psm\rangle$
brand	$\langle -, avpo\rangle$	$\langle +, psm\rangle$	$\langle \sim, psm\rangle$	$\langle \sim, psm\rangle$	$\langle +, avpo\rangle$
stars	$\langle \sim, max\rangle$	$\langle -, max\rangle$	$\langle \sim, max\rangle$	$\langle -, max\rangle$	$\langle -, max\rangle$
price	$\langle +, min\rangle$	$\langle +, min\rangle$	$\langle +, min\rangle$	$\langle +, min\rangle$	$\langle +, min\rangle$

(c) Comparison with Ap_C

	Ap_A	**Ap_B**	**Ap_C**	**Ap_E**	**Ap_F**
uni	$\langle \sim, psm\rangle$	$\langle -, psm\rangle$	$\langle \sim, psm\rangle$	$\langle \sim, psm\rangle$	$\langle -, psm\rangle$
station	$\langle -, min\rangle$	$\langle +, min\rangle$	$\langle -, min\rangle$	$\langle \sim, min\rangle$	$\langle +, min\rangle$
market	$\langle -, min\rangle$	$\langle -, min\rangle$	$\langle -, min\rangle$	$\langle -, min\rangle$	$\langle -, min\rangle$
zone	$\langle \sim, psm\rangle$	$\langle \sim, psm\rangle$	$\langle +, psm\rangle$	$\langle \sim, psm\rangle$	$\langle \sim, psm\rangle$
brand	$\langle -, avpo\rangle$	$\langle +, psm\rangle$	$\langle \sim, psm\rangle$	$\langle \sim, psm\rangle$	$\langle +, avpo\rangle$
stars	$\langle \sim, max\rangle$	$\langle -, max\rangle$	$\langle \sim, max\rangle$	$\langle -, max\rangle$	$\langle -, max\rangle$
price	$\langle -, min\rangle$	$\langle +, min\rangle$	$\langle -, min\rangle$	$\langle -, min\rangle$	$\langle +, min\rangle$

(d) Comparison with Ap_D

	Ap_A	**Ap_B**	**Ap_C**	**Ap_D**	**Ap_F**
uni	$\langle \sim, psm\rangle$	$\langle -, psm\rangle$	$\langle \sim, psm\rangle$	$\langle \sim, psm\rangle$	$\langle -, psm\rangle$
station	$\langle -, min\rangle$	$\langle +, min\rangle$	$\langle -, min\rangle$	$\langle \sim, min\rangle$	$\langle +, min\rangle$
market	$\langle +, min\rangle$	$\langle +, min\rangle$	$\langle -, min\rangle$	$\langle +, min\rangle$	$\langle +, min\rangle$
zone	$\langle \sim, psm\rangle$	$\langle \sim, psm\rangle$	$\langle +, psm\rangle$	$\langle \sim, psm\rangle$	$\langle \sim, psm\rangle$
brand	$\langle -, avpo\rangle$	$\langle +, psm\rangle$	$\langle \sim, psm\rangle$	$\langle \sim, psm\rangle$	$\langle +, avpo\rangle$
stars	$\langle +, max\rangle$	$\langle \sim, max\rangle$	$\langle +, max\rangle$	$\langle +, max\rangle$	$\langle -, max\rangle$
price	$\langle \sim, min\rangle$	$\langle +, min\rangle$	$\langle -, min\rangle$	$\langle +, min\rangle$	$\langle +, min\rangle$

(e) Comparison with Ap_E

	Ap_A	**Ap_B**	**Ap_C**	**Ap_D**	**Ap_E**
uni	$\langle +, psm\rangle$	$\langle \sim, psm\rangle$	$\langle +, psm\rangle$	$\langle +, psm\rangle$	$\langle +, psm\rangle$
station	$\langle -, min\rangle$	$\langle -, min\rangle$	$\langle -, min\rangle$	$\langle -, min\rangle$	$\langle -, min\rangle$
market	$\langle +, min\rangle$	$\langle \sim, min\rangle$	$\langle -, min\rangle$	$\langle +, min\rangle$	$\langle -, min\rangle$
zone	$\langle \sim, psm\rangle$	$\langle \sim, psm\rangle$	$\langle +, psm\rangle$	$\langle \sim, psm\rangle$	$\langle \sim, psm\rangle$
brand	$\langle -, avpo\rangle$	$\langle +, psm\rangle$	$\langle -, avpo\rangle$	$\langle -, avpo\rangle$	$\langle -, avpo\rangle$
stars	$\langle +, max\rangle$	$\langle +, max\rangle$	$\langle +, max\rangle$	$\langle +, max\rangle$	$\langle +, max\rangle$
price	$\langle -, min\rangle$	$\langle -, min\rangle$	$\langle -, min\rangle$	$\langle -, min\rangle$	$\langle -, min\rangle$

(f) Comparison with Ap_F

are aware that there may be exceptions, and an implicit preference may be wrongly considered in certain cases — as it occurs with humans. This problem can be tackled with the addition of knowledge specific to an application area, e.g. by stating if ordered attributes should in general be maximised or minimised, or by learning what individual users usually mean by their provided preferences. Currently, we adopt a set of implicit preferences that in general derives correct preferences from explicit preferences.

Before introducing our set of implicit preferences, we will describe the context in which they will be considered. Implicit preferences do not override information obtained from explicitly provided preferences, that is, they change OAPM values only when two options are considered similar (w.r.t. an attribute) or no conclusion could be made. Therefore, the OAPM value for these two options must be either $\sim$ or ? and, if the value is $\sim$, it was not set due to an indifference preference, as we formally show in Equation 6-7. Moreover, our current implicit preferences are derived from monadic preferences, and as different monadic preferences may be used to derive different implicit preferences, they are derived when there is *only one monadic preference* that refers to an attribute and is applicable to a pair of options. Equation 6-8 is thus used to verify this restriction.

$$\begin{aligned} &Undecided(o_1, o_2, a) := \\ &\quad OAPM[o_1, o_2, a] = \langle ?, r\rangle \vee (OAPM[o_1, o_2, a] = \langle \sim, r\rangle \wedge r \neq indiff) \end{aligned} \tag{6-7}$$

$$
\begin{aligned}
&UniqueMonadic(p, o_1, o_2, a) := \\
&\quad p \in MP \wedge App(p, o_1, a) \wedge App(p, o_2, a) \wedge p(att) = a \\
&\quad \wedge \nexists p'.(p \neq p' \wedge p' \in MP \wedge App(p', o_1, a) \wedge App(p', o_2, a) \wedge p'(att) = a)
\end{aligned}
\tag{6-8}
$$

Our implicit preferences are also valid only for attributes whose domain is ordered. Finally, when keeping track why the OAPM is being updated, the reason is stored in the form of $\langle type, p \rangle$, where $type$ is the type of the applied implicit preference, and p is the preference that caused the update. This information is used in the selection step (Section 6.7). Now, we can proceed to our set of implicit preferences.

6.5.1 Upper bound

In the upper bound case, an upper limit is provided for an attribute (as in preference 2), which is given in a monadic preference whose formula is an instance of $attribute < value$ or $attribute \leq value$. A preference that satisfies this restriction returns true for $UpperBound(p)$. Due to this upper bound, we infer that there is a minimisation goal for this attribute. Note that monadic preferences may be associated with negative modifiers, and in the case the inference is the opposite: there a maximisation goal.

$$
\begin{aligned}
&Undecided(o_1, o_2, a) \wedge UniqueMonadic(p, o_1, o_2, a) \wedge UpperBound(p) \\
&\quad \wedge ((\neg Negative(mod(p)) \wedge val(o_1, a) < val(o_2, a)) \\
&\quad \vee (Negative(mod(p)) \wedge val(o_1, a) > val(o_2, a))) \rightarrow \\
&\qquad OAPM[o_1, o_2, a] = \langle +, \langle upper, p \rangle \rangle \wedge OAPM[o_2, o_1, a] = \langle -, \langle upper, p \rangle \rangle
\end{aligned}
\tag{6-9}
$$

6.5.2 Lower bound

As opposed to the previous case, a lower bound can be provided for an attribute, with monadic preferences whose formula are instances of $attribute > value$ or $attribute \geq value$, thus indicating that the goal is to maximise the value of this attribute (or to minimise it if the preference modifier is negative). Similar to above, $LowerBound(p)$ is true for preferences that satisfy the formula template. This is the case of preference 1, but as it is associated with a negative modifier, the inferred preference is a minimisation goal.

$$Undecided(o_1, o_2, a) \wedge UniqueMonadic(p, o_1, o_2, a) \wedge LowerBound(p)$$
$$\wedge ((\neg Negative(mod(p)) \wedge val(o_1, a) > val(o_2, a))$$
$$\vee (Negative(mod(p)) \wedge val(o_1, a) < val(o_2, a))) \rightarrow$$
$$OAPM[o_1, o_2, a] = \langle +, \langle lower, p \rangle\rangle \wedge OAPM[o_2, o_1, a] = \langle -, \langle lower, p \rangle\rangle \quad (6\text{-}10)$$

6.5.3 Around

If a desired value is given for an ordered attribute (a *reference value*), and this preference is not a hard constraint (i.e. it may be left unsatisfied), we infer that the closer the attribute value of an option is to the desired value, the better. Therefore, between two options, the preferred one is that whose value for this attribute has a shorter distance from the reference value. The rule that updates the OAPM according to the around implicit preference uses the following notations: (i) $Around(p)$, which is true when the monadic preference formula is an instance of $attribute = value$; (ii) $RefVal(p)$, which returns the reference value of a preference that satisfies $Around(p)$; and (iii) $AroundDist(o, a, p) :=\mid val(o, a) - RefVal(p) \mid$, which gives the distance between the attribute value of an option and the attribute reference value. Negative modifiers, as before, inverts the behaviour of the around preference.

$$Undecided(o_1, o_2, a) \wedge UniqueMonadic(p, o_1, o_2, a) \wedge Around(p)$$
$$\wedge ((\neg Negative(mod(p)) \wedge AroundDist(o_1, a, p) < AroundDist(o_2, a, p))$$
$$\vee (Negative(mod(p)) \wedge AroundDist(o_1, a, p) > AroundDist(o_2, a, p))) \rightarrow$$
$$OAPM[o_1, o_2, a] = \langle +, \langle around, p \rangle\rangle \wedge OAPM[o_2, o_1, a] = \langle -, \langle around, p \rangle\rangle \quad (6\text{-}11)$$

6.5.4 Interval

Instead of providing a single desired attribute value, users may provide an *interval*. In these cases, the provided monadic preference is associated with a formula that is an instance of $attribute > lowerBound$ **and** $attribute < upperBound$, or $\geq$ and $\leq$, instead of $>$ and $<$. We introduce: (i) $Interval(p)$, which is true when preference satisfies the formula template; (ii) $lb(p)$, which gives the interval lower bound; and (iii) $ub(p)$, which gives the interval upper bound.

Similarly to the around rule, the OAPM is updated based on the interval implicit preference using an auxiliary function to calculate the distance from

attribute values to the provided interval, which is depicted next. This function may be modified according to the provided interval, by changing $<$ to $\leq$. Note that the interval distance of attribute values that are in the interval is 0, and this makes the OAPM value between these two attribute values remain the same.

$$IntervalDist(o, a, p) =$$
$$\begin{cases} 0 & \text{if } lb(p) < val(o, a) < ub(p) \\ min(\mid val(o, a) - lb(p) \mid, \mid val(o, a) - ub(p) \mid) & \text{otherwise} \end{cases} \tag{6-12}$$

Given this function, we can now present the rule that updates the OAPM, which sets the attribute value of an option that has the lower distance from the interval than another as preferred; or the opposite, if the preference modifier is negative.

$$\begin{aligned} &Undecided(o_1, o_2, a) \wedge UniqueMonadic(p, o_1, o_2, a) \wedge Interval(p) \\ &\quad \wedge ((\neg Negative(mod(p)) \wedge IntervalDist(o_1, a, p) < IntervalDist(o_2, a, p)) \\ &\quad \vee (Negative(mod(p)) \wedge IntervalDist(o_1, a, p) > IntervalDist(o_2, a, p))) \rightarrow \\ &\qquad OAPM[o_1, o_2, a] = \langle +, \langle interval, p \rangle \rangle \wedge OAPM[o_2, o_1, a] = \langle -, \langle interval, p \rangle \rangle \end{aligned} \tag{6-13}$$

There are three cases to which implicit preferences are applicable in our running example, which are related to the preferences 1, 2 and 14. Now, the relationship between options with respect to the attribute $zone$ is established by the goal of minimising the value of this attribute (upper bound preference, with a negative modifier), and the attribute uni, which has also the goal of minimising the value of this attribute (upper bound preference, with a positive modifier). The unique modifier preference with respect to $stars$ indicates that the closer that the apartment stars are to 2, the better (around preference), but option attribute values are either already decided by the explicitly provided goal, or equal, thus the comparison remains $\sim$. Table 6.6 shows the updated OAPM, only with attributes that changed (for simplicity, we omit the preference that is part of the OAPM reason).

After executing the steps for building our two computational models that support the decision making process, and updating them by considering implicit preferences, we still have preferences provided by users that we have not taken into account, which are don't care preferences and priorities over preferences and over attributes. These will be used later, when resolving trade-off situations for choosing an option but, before, we use the constructed PSM and the OAPM to eliminate options, as presented next.

Table 6.6: Updated OAPM of the Apartment Decision Problem.

	Ap_B	**Ap_C**	**Ap_D**	**Ap_E**	**Ap_F**
uni	$\langle -, psm \rangle$	$\langle +, upper \rangle$	$\langle +, upper \rangle$	$\langle -, upper \rangle$	$\langle -, psm \rangle$
zone	$\langle -, lower \rangle$	$\langle +, psm \rangle$	$\langle \sim, lower \rangle$	$\langle \sim, lower \rangle$	$\langle -, lower \rangle$
stars	$\langle -, max \rangle$	$\langle \sim, around \rangle$	$\langle \sim, around \rangle$	$\langle -, max \rangle$	$\langle -, max \rangle$

(a) Comparison with Ap_A

	Ap_A	**Ap_C**	**Ap_D**	**Ap_E**	**Ap_F**
uni	$\langle +, psm \rangle$	$\langle +, psm \rangle$	$\langle +, psm \rangle$	$\langle +, psm \rangle$	$\langle -, upper \rangle$
zone	$\langle +, lower \rangle$	$\langle +, psm \rangle$	$\langle +, lower \rangle$	$\langle +, lower \rangle$	$\langle \sim, lower \rangle$
stars	$\langle +, max \rangle$	$\langle +, max \rangle$	$\langle +, max \rangle$	$\langle \sim, around \rangle$	$\langle -, max \rangle$

(b) Comparison with Ap_B

	Ap_A	**Ap_B**	**Ap_D**	**Ap_E**	**Ap_F**
uni	$\langle -, upper \rangle$	$\langle -, psm \rangle$	$\langle -, upper \rangle$	$\langle -, upper \rangle$	$\langle -, psm \rangle$
zone	$\langle -, psm \rangle$	$\langle -, psm \rangle$	$\langle -, psm \rangle$	$\langle -, psm \rangle$	$\langle -, psm \rangle$
stars	$\langle \sim, around \rangle$	$\langle -, max \rangle$	$\langle \sim, max \rangle$	$\langle -, max \rangle$	$\langle -, max \rangle$

(c) Comparison with Ap_C

	Ap_A	**Ap_B**	**Ap_C**	**Ap_E**	**Ap_F**
uni	$\langle -, upper \rangle$	$\langle -, psm \rangle$	$\langle +, upper \rangle$	$\langle -, upper \rangle$	$\langle -, psm \rangle$
zone	$\langle \sim, lower \rangle$	$\langle -, lower \rangle$	$\langle +, psm \rangle$	$\langle \sim, lower \rangle$	$\langle -, lower \rangle$
stars	$\langle \sim, around \rangle$	$\langle -, max \rangle$	$\langle \sim, around \rangle$	$\langle -, max \rangle$	$\langle -, max \rangle$

(d) Comparison with Ap_D

	Ap_A	**Ap_B**	**Ap_C**	**Ap_D**	**Ap_F**
uni	$\langle +, upper \rangle$	$\langle -, psm \rangle$	$\langle +, upper \rangle$	$\langle +, upper \rangle$	$\langle -, psm \rangle$
zone	$\langle \sim, lower \rangle$	$\langle -, lower \rangle$	$\langle +, psm \rangle$	$\langle \sim, lower \rangle$	$\langle -, lower \rangle$
stars	$\langle +, max \rangle$	$\langle \sim, around \rangle$	$\langle +, max \rangle$	$\langle +, max \rangle$	$\langle -, max \rangle$

(e) Comparison with Ap_E

	Ap_A	**Ap_B**	**Ap_C**	**Ap_D**	**Ap_E**
uni	$\langle +, psm \rangle$	$\langle +, upper \rangle$	$\langle +, psm \rangle$	$\langle +, psm \rangle$	$\langle +, psm \rangle$
zone	$\langle +, lower \rangle$	$\langle \sim, lower \rangle$	$\langle +, psm \rangle$	$\langle +, lower \rangle$	$\langle +, lower \rangle$
stars	$\langle +, max \rangle$	$\langle +, max \rangle$	$\langle +, max \rangle$	$\langle +, max \rangle$	$\langle +, max \rangle$

(f) Comparison with Ap_F

6.6 Elimination

One of the typical approaches adopted by users for making a choice is the elimination of options in a stepwise fashion until it remains a set of acceptable options, which ideally contains only one element. This is the main idea of the *Elimination by aspects* (Tversky 1972) and *Satisficing* (Simon 1955) approaches from psychology, and it often consists of an iterative approach of making decisions, which is not a characteristic of our technique. However, we can take an initial step in this direction, by eliminating options that have two properties: (i) dominated options (Section 6.6.1); and (ii) options that do not satisfy hard constraints (Section 6.6.2).

6.6.1 Eliminating Dominated Options

We begin by eliminating options that, for at least one attribute, are worse than another option, and that are not better than it for the remaining attributes. In this situation, we say that the options to be discarded are dominated by another.

Based on the information provided by the OAPM, which was constructed based on provided user preferences, we can define *domination*, which is a binary relation that indicates when an option dominates another, formally presented in Definition 6.4.

Definition 6.4 *Let* o_1 *and* o_2 *be two options in* Opt*, and* a *an attribute in* Att*.* $dominates(o_1, o_2)$ ***holds when:***

$$\exists\, a.(OAPM[o_1, o_2, a] = +) \wedge \forall\, a.(OAPM[o_1, o_2, a] \neq -)$$

For the domination relation to be true, o_1 must have an attribute value that is preferred to the attribute value of o_2. In addition, for all other attribute values, o_1 has to be at least as good as o_2. Therefore, we have a reason to reject o_2, and there is no other positive aspect of this option that can balance its negative aspects, w.r.t. o_1. Domination also holds in the absence of information about the attribute comparison of two options ($OAPM[o_1, o_2, a] =?$), if there is at least one attribute whose OAPM value is +.

Based on the definition of domination, we now define the set of dominated options, $Dominated$. As shown in Equation 6-14, all options dominated by at least one other option are in the $Dominated$ set and are discarded.

$$dominates(o_1, o_2) \rightarrow o_2 \in Dominated \tag{6-14}$$

In Tables 6.5(a) and 6.5(e), it can be seen that the OAPM has the value + or $\sim$, for all attributes of options Ap_A and Ap_E, when compared to Ap_D. Therefore, it can be said that $dominates(Ap\_A, Ap\_D)$ and $dominates(Ap\_E, Ap\_D)$, thus $Ap\_D \in Dominated$.

6.6.2 Applying Cut-off Values

The second set of eliminated options is composed of options that do not satisfy hard constraints of users. We consider hard constraints preferences that are either qualifying or rating statements with one of these four modifiers: (i) *don't accept*; (ii) *hate*; (iii) *require*; and (iv) *need*.

Other modifiers, such as *very good*, *want* and *very bad*, are also strong preferences from users, but they are not considered at this moment, because options have positive and negative aspects w.r.t. each other (otherwise it is a case of domination) and, even though an option has an attribute value that is *very bad*, it may be amortised by other positive aspects of this option. This argument is not valid for the four modifiers considered as hard constraints, because they are

interpreted as associated with *non-compensatory* attribute values, i.e. those that cannot be compensated with any benefit provided by other attributes.

In order to identify the options that are discarded due to cut-off values, we use the information captured by the PSM. First, we show how to select options associated with hard negative modifiers, and then those associated with unsatisfied hard positive modifiers. In the first case, the options selected to be part of the $CutOff$ set, i.e. the set of options discarded due to a cut-off value, are those that have at least one attribute associated with $\langle \epsilon, don't\ accept \rangle$ or $\langle \epsilon, hate \rangle$ in the PSM.

$$\begin{aligned} \exists\, a.(PSM[o, a] = \langle \epsilon, \text{“}don't\ accept\text{”} \rangle \\ \vee\ PSM[o, a] = \langle \epsilon, \text{“}hate\text{”} \rangle) \rightarrow o \in CutOff \end{aligned} \tag{6-15}$$

By construction, every option that satisfies a preference with either the “*don't accept*” or the “*hate*” modifier has the PSM value evaluated for these modifiers for the respective attribute. Therefore, these constraints are always respected.

Differently from the negative modifiers above, *require* and *need* are considered hard constraints unless another positive experience is provided. For example, assume these two preferences: “*I require an apartment at zone 1*”, and “*I accept one in zone 2*.” In this case, the second preference changes the first one to a soft preference, as it indicates an exception to the requirements. So, even though an apartment in zone 2 does not satisfy a requirement, it is not eliminated due to a cut-off. Therefore, *require* and *need* are usually hard constraints, but users may add acceptable exceptions, as in this example. So, we exclude options that satisfy neither a requirement or need, nor any other monadic preference whose modifier is neutral or positive, w.r.t. a particular attribute. However, if an option does not satisfy the requirement or need, but satisfies a monadic preference whose modifier is negative, then it is discarded when there is at least one option that satisfies the requirement or need. This interpretation is adopted because if users provide other preferences related to an attribute besides a requirement or need, it is an indication that they are a soft constraint, but still have a strong preference for their satisfaction. According to this informal explanation, we show below the rule used to select options to be rejected due to positive hard constraints.

$$\begin{aligned} \exists\, a.(PSM[o, a] = \langle \neg, require \rangle \vee PSM[o, a] = \langle \neg, need \rangle \\ \vee\ \exists\, o'.((PSM[o', a] = \langle \epsilon, require \rangle \vee PSM[o', a] = \langle \epsilon, need \rangle) \\ \wedge\ PSM(o, a) = \langle \epsilon, Negative(m) \rangle)) \rightarrow o \in CutOff \end{aligned} \tag{6-16}$$

By construction, if no monadic preference is satisfied by an option, and one of them is associated with a requirement (or need), the PSM value of this option will indicate an unsatisfied requirement (or need).

The PSM presented in Table 6.3 shows that $Ap\_C \in CutOff$, as $PSM[Ap\_C, zone] = \langle \epsilon, don't\ accept \rangle$. Note that an option eliminated because of domination may also be discarded due to a cut-off value, i.e. it is possible that $Dominated \cap CutOff \neq \emptyset$.

6.7 Selection

After performing the elimination step of our process, our set of available options is now reduced to a subset of options (which we refer to as $Acceptable$, but it is also referred to as *consideration set* in the literature), which requires us to resolve trade-offs to make a choice. It is important to highlight that, except for preference and attribute priorities (which will be taken into account in this step), we have already used the information provided by users to reject options, thus in order to make a decision we have to go beyond user preferences. This is the common situation that happens in the process of decision making, as users typically have preferences for individual attributes and resolve trade-offs in light of available options to establish a preference order among them. Our technique, inspired by human decision making, analyses cost and benefits of options and additional factors that humans typically adopt while making decisions.

Humans may also adopt other heuristics to make decisions (Payne et al. 1988), each requiring different amounts of cognitive effort. Heuristics are chosen based on the amount of effort required and the relevance of the decision to be made. Our approach does not aim to reproduce this particular behaviour, because it may be suboptimal when users are not willing to invest effort in the process. Our goal is to understand how people reason to resolve trade-offs, when demanding adequate time and effort to make a decision and this is the reasoning process we adopt to make automated choices. Avoiding extreme options (best for some attributes and worst for others) and analysing the trade-off contrast (influenced by other options), are the two main principles that humans adopt that our technique incorporates (Simonson and Tversky 1992).

The following sections describe the selection step of our technique in four parts. First, we describe how the benefits and costs of each option in the $Acceptable$ set, where $Acceptable = Options \backslash (Dominated \cup CutOff)$, are evaluated with respect to each individual attribute. Then, we show how we assess the overall benefits and costs of options with respect to each other based on the previous evaluation of each individual attribute. Later, we consider the two main principles that are typically adopted by people when making decisions. Finally, we show how all these evaluations are put together and used to make a final decision.

6.7.1 Cost-benefit Analysis

The first part of the process of selecting an option consists of evaluating each pair of options and assessing their relative benefits and costs, using the information provided by the OAPM. The costs of option o_1 w.r.t. o_2 are the benefits of option o_2 w.r.t. o_1, and vice-versa. We compute the cost of o_1 compared to o_2 w.r.t. an attribute a to a real value ranging from 0 to 1, captured by a function we build, represented as shown below.

$$AttCost : Opt \times Opt \times Att \rightarrow \{c \mid c \in \mathbb{R} \wedge 0 \leq c \leq 1\}$$

This value indicates *how much* one option is better than another, w.r.t. to each attribute, which will be used to evaluate the overall option costs. In essence, our cost function transforms qualitative information into quantitative values. One can argue that this quantitative values will comprise a utility function (Keeney and Raiffa 1976), as it is a weighted sum of values given for individual attributes, which represent how much an individual prefers each attribute value, but it is not, mainly because of two main differences. First, the cost function consists of *differences* between values as the cost is obtained by means of a *pairwise* comparison between options (as people usually do), and uses the *specific preferences* for each option pair, as there are many types of preferences applicable for pairs of options (note that using qualitative preferences to quantitatively evaluate preference for options is also a challenge). Second, the way we obtain these cost values is novel: we exploit natural-language-like expressions instead of submitting users to iterative processes, which demands high cognitive effort and time from users. Furthermore, this function is not decisive for making a choice: as introduced before, we will also add two factors that influences the preferences for options, which are associated with the two principles of human decision making. Therefore, option costs are not the unique factors that are needed for making the decision.

The attribute cost is 0, if the $OAPM[o_1, o_2, a] \neq \langle -, r \rangle$; otherwise, i.e. if $OAPM[o_1, o_2, a] = \langle -, r \rangle$, then we use the reason r to compute the $AttCost[o_1, o_2, a]$. The next sections describe this computation for the seven possible reasons r: PSM, order preferences, goal, lower bound, upper bound, around and interval. The remaining one, indifference, always causes OAPM values to be set to $\sim$, and therefore $AttCost$ is 0.

PSM

Our modifier scale (Figure 6.2) allows us to identify whether a modifier is stronger than another, and consequently the preference for a value compared to

another when different modifiers are used to qualify these values. However, as in this step we aim to asses *how much* one attribute value is preferred to another, we have to go beyond the order given by this scale. In order to make this assessment, we associate a numeric value with each index of PSM value, and therefore an option cost for the particular attribute with respect to another is calculated based on the difference between the values associated with the PSM values.

The association of a numeric value with each PSM value is given by a function that we refer to as f_m, which corresponds to a variation point of our technique. We have considered three different functions for generating a value for modifiers:

(i) **linear**: $f_{m_{li}}(index) = index$;

(ii) **quadratic**: $f_{m_{sq}}(index) = index^2$, if $index \geq 0$ and $f_{m_{sq}}(index) = -(index^2)$, if $index < 0$;

(iii) **log**: $f_{m_{lg}}(index) = \ln(\mid index \mid +1)$, if $index \geq 0$ and $f_{m_{lg}}(index) = -\ln(\mid index \mid +1)$, if $index < 0$.

This way of calculating the option cost for a particular attribute based on the PSM is shown in Equation 6-17. The value related to the modifiers is normalised to a value between 0 and 1, considering the possible modifiers (or their negation) that can be associated with any of the two options being compared. Therefore, for all monadic preferences that either $App(p, o_1)$ or $App(p, o_2)$, we select the $\langle \epsilon, m \rangle$ or $\langle \neg, m \rangle$ that has the maximum and minimum indices, i.e. we obtain which are the maximum and minimum values associated with a PSM value that these options can have, represented in Equation 6-17 as $max(maxIdx(o_1), maxIdx(o_2))$ and $min(minIdx(o_1), minIdx(o_2))$, respectively.

$$AttCost(o_1, o_2, a) = \frac{\mid f_m(PSMIndex([PSM[o_1, a], scale)) - f_m(PSMIndex([PSM[o_2, a], scale)) \mid}{f_m(max(maxIdx(o_1), maxIdx(o_2))) - f_m(min(minIdx(o_1), minIdx(o_2)))} \quad (6\text{-}17)$$

We have adopted the **log** function ($f_{m_{lg}}$) in our approach, chosen based on experimentation. This function makes the difference between strong modifiers, such as $want$ and $best$, smaller than the differences between modifiers in the middle of the scale, such as $neutral$ and $don't\ avoid$, and therefore the preference is stronger when we compare positive modifiers with negative modifiers. For example, the cost of values rated with $neutral$ compared to values qualified with $don't\ prefer$ is higher than the cost of values qualified with $want$ compared to values qualified with $desire$, even though for both the index differences are of two units.

When calculating the cost of the attribute uni with respect to options Ap_A and Ap_B, we obtain as the maximum and minimum possible values for

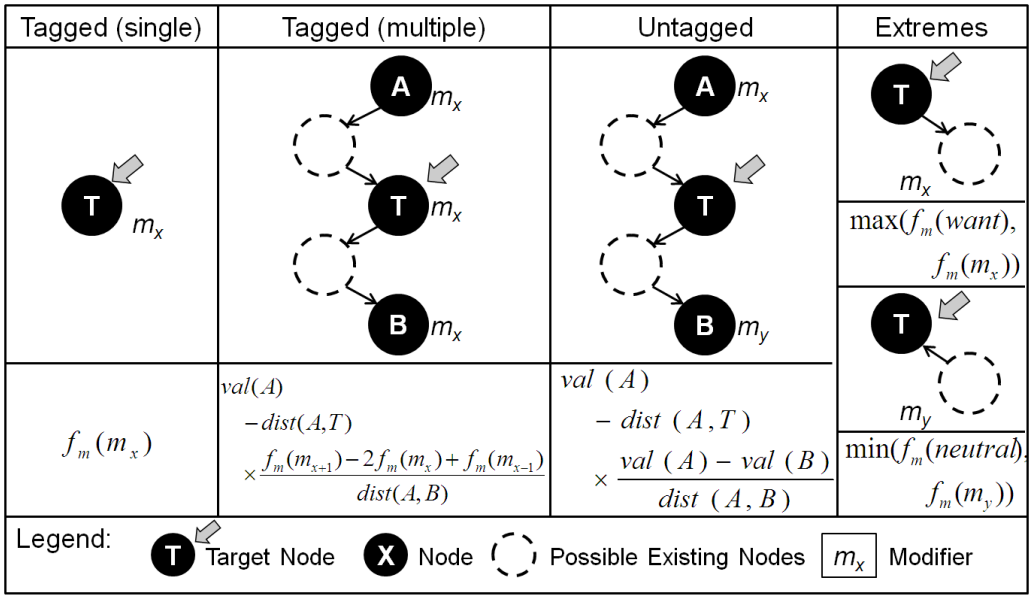

Figure 6.3: Calculating node values.

these options are those associated with $\langle \epsilon, prefer \rangle$ and $\langle \neg, prefer \rangle$, respectively, which are also the PSM values associated with Ap_B and Ap_A. Therefore, we have $\mid f_{m_{lg}}(6) - f_{m_{lg}}(-2) \mid / (f_{m_{lg}}(6) - f_{m_{lg}}(-2))$, which is 1.0 — the value of $AttCost(Ap\_A, Ap\_B, uni)$.

Order Preferences

AVPOs allow comparing attribute values and identifying the preferred one; however, as in our modifier scale, it is not possible to know how much one value is preferred to another. So, in order to obtain this information, we also associate numeric values with AVPO nodes, and for that we use information from the monadic preferences. Each AVPO is associated with an option and an attribute, and the monadic preferences considered are those that have their condition satisfied by that option and attribute. We initiate this process by tagging AVPO nodes with a modifier from the monadic preferences whose formula is satisfied by the domain value associated with the node: for selecting from among multiple modifiers, we follow the same rules used for building the PSM, but here we have only satisfied modifiers. Based on this tagging, the AVPO nodes are associated with a numerical value, as summarised in Figure 6.3 and explained in detail next.

Extreme Nodes For guaranteeing the existence of tagged values in the AVPO, we tag all most preferred values and all least preferred nodes (fourth column of Figure 6.3). The former is tagged with $want$ (in our current modifier scale, it is the strongest modifier) and the latter with $neutral$. If there are values in the partial order

tagged with a stronger modifier than $want$ or $neutral$, for instance $A \succ B \succ C$, and B is tagged with $avoid$, the extreme untagged nodes receive these tags for keeping consistency, i.e. A is tagged with $want$ (as the default) and C with $avoid$. We tag least preferred nodes with $neutral$ by default, because people typically provide an order for preferred or acceptable values, and do not mention not preferred ones — we confirmed this in our previous study (Chapter 2).

Tagged Nodes If the node is tagged with a modifier (first and second columns of Figure 6.3), it receives the value according to the values given for the modifier scale. However, there may be different values tagged with the same modifier and ordered in a sequential way, such as in the running example in which all nodes of the AVPO are tagged with the $prefer$ modifier.

In order to address this issue, Algorithms 3 and 4 are used to calculate the value of tagged AVPO nodes. The first searches for the most distant node that has the same modifier of the target modifier, either searching through the parents or children of nodes, according to a parameter up provided to the algorithm.

Algorithm 3: $LastEqual(tag, node, up, scale)$

Input: tag: modifier to be searched for;$node : AVPONode$; up : $boolean$ (flag to indicate if the search should be in the parents or the children of $node$; $scale$: modifier scale

Output: $\langle firstTagged, dist\rangle : \langle AVPONode, int\rangle$, first tagged node and the distance from it to $node$

```
1  List⟨ AVPONode ⟩ nodeList ← up ? Parents(node) : Children(node);
2  ⟨ AVPONode, int ⟩ lastEqual ← null;
3  double tagValue ← f_m(IndexOf(tag, scale));
4  foreach AVPONode next ∈ nodeList do
5      double nextTagValue ← null;
6      if Tag(next) ≠ null then
7          double nextTagValue ← f_m(IndexOf(Tag(next), scale));
8      if nextTagValue = null ∨ tagValue = nextTagValue then
9          ⟨ AVPONode, int ⟩ temp ← LastEqual(tag, next, up, scale);
10         if temp ≠ null then
11             temp_2 ← π_2(temp) + 1;
12         else if tagValue = nextTagValue then
13             temp ← ⟨next, 0⟩;
14         if lastEqual = null ∨ π_2(lastEqual) < π_2(temp) then
15             lastEqual ← temp;
16 return firstTagged;
```

Then, in Algorithm 4, two situations can happen. If the node tagged with a particular modifier is unique, its node value is given by the f_m function. In the second situation — more than one node is tagged with the same modifier — the most preferred value (brand A, in our running example) is associated with the numeric value related to the modifier, plus half of the difference between the modifier value and the modifiers whose index is increased in one unit, according the modifier scale

(in the example, *desire*). Similarly, the least preferred value (C) is associated with the value related to the modifier, minus half of the difference between the modifier value and the modifiers whose index is decreased in one unit (in the example, *need*). With these values, we divide the difference between the values associated with the most and least preferred values by the their distance (which is two, in the case of A and C) for obtaining the difference between any two values, which we refer to as $step$, and with it we are able to calculate the value of the remaining values. Therefore, the value associated with B is the value of A minus the distance between A and B times the $step$, as shown in lines 23 and 24 of Algorithm 4.

Algorithm 4: $TaggedNodeValue(node, scale)$

Input: $node$: $AVPONode$; $scale$: modifier scale
Output: $value$: $double$

```
1  int dist ← 0;
2  ⟨ AVPONode, int ⟩ above ← LastEqual(Tag(node), node, true, scale);
3  if above ≠ null then
4      dist ← π2(above);
5  ⟨ AVPONode, int ⟩ below ← LastEqual(Tag(node), node, false, scale);
6  if below ≠ null then
7      dist ← π2(below);
8  double tagValue ← fm(IndexOf(Tag(node),scale));
9  if dist = 0 then
10     return tagValue;
11 else
12     double max ← tagValue;
13     double temp ← fm(IndexOf(Tag(node), scale) +1);
14     if temp ≠ null then
15         max = max+ | temp − tagValue | /2;
16     double min ← tagValue;
17     double temp ← fm(IndexOf(Tag(node), scale) −1);
18     if temp ≠ null then
19         min = min− | temp − tagValue | /2;
20 if above = null then
21     return max;
22 else
23     double step ←| max − min | / dist;
24     return max − π2(above) × step;
```

Untagged Nodes If the node is not tagged with a modifier (third column of Figure 6.3), we first obtain the maximum and minimum surrounding node values of the target node, which is done through the execution of Algorithm 5. We find the closest tagged nodes that are preferred to the target node, and the closest tagged nodes less preferred than the target node. Then, we choose from these tagged nodes by selecting those that has the smaller difference from the target node: we consider their unsigned numerical value (as they are tagged, it is calculated in the way we explained above) and divide by the distance between them and the target value ($step$). Next, we calculate the numerical value for the node immediately above (or

below) the target node: we take the numerical value of the tagged node and subtract (or add) from it the step times the distance from the tagged node to the target node minus one, as we are calculating the value of the node immediately above or below. The lower (higher) numerical value calculated for this node is chosen, and therefore we guarantee that all more preferred nodes (than the target node) are associated with a higher numerical value and all less preferred nodes (than the target node) are associated with a lower numerical value.

Algorithm 5: $FirstTaggedNode(node, up, scale)$

Input: $node : AVPONode$; $up : boolean$ (flag to indicate if the search should be in the parents or the children of $node$; $scale$: modifier scale
Output: $\langle firstTagged, dist\rangle : \langle AVPONode, int\rangle$, first tagged node and the distance from it to $node$

```
1  List⟨ AVPONode ⟩ nodeList ← up ? Parents(node) : Children(node);
2  ⟨ AVPONode, int ⟩ firstTagged ← null;
3  double rate ← 0;
4  foreach AVPONode next ∈ nodeList do
5      ⟨ AVPONode, int ⟩ temp ← null;
6      if Tag(next) = null then
7          temp ← FirstTaggedNode(next, up, scale);
8      else
9          temp ← ⟨next, 0⟩;
10     temp2 ← π2(temp) + 1;
11     double tagValue ← fm(IndexOf(Tag(π1(temp)), scale));
12     double step ←| tagValue | /π2(temp);
13     if up then
14         double rateTemp ← tagValue − (π2(temp) − 1) × step;
15         if rate = null ∨ rateTemp < rate then
16             firstTagged ← temp;
17             rate ← rateTemp;
18     else
19         double rateTemp ← tagValue + (π2(temp) − 1) × step;
20         if rate = null ∨ rateTemp > rate then
21             firstTagged ← temp;
22             rate ← rateTemp;
23 return firstTagged;
```

After choosing the preferred and less preferred nodes, we calculate the difference between their numerical values and divide it by their distance ($step$), and then we calculate the numerical value related to the target node as above, as shown in Algorithm 6.

With this set of algorithms, the value of an AVPO node is given as shown below.

$$NodeVal(node) = \begin{cases} UntaggedNodeValue(node) & \text{if } Tag(node) = null \\ TaggedNodeValue(node) & \text{otherwise} \end{cases}$$

After associating tags and numerical values with the AVPO nodes, we can calculate the costs of an option with respect to an attribute using this information.

Algorithm 6: $UntaggedNodeValue(node, up, scale)$

Input: $node$: $AVPONode$; $scale$: modifier scale
Output: $value$: $double$

```
⟨ AVPONode, int ⟩ above ← FirstTaggedNode(node, true, scale);
double aboveValue ← TaggedNodeValue(π1(above), scale);
⟨ AVPONode, int ⟩ below ← FirstTaggedNode(node, false, scale);
double belowValue ← TaggedNodeValue(π1(below), scale);
double step ←| aboveValue − belowValue | /(π2(above) + π2(below));
return aboveValue − (π2(above) × step);
```

The cost associated with the attribute values of two options according to a given AVPO is shown in Equation 6-18, where $Node(AVPO[o, a], o_1)$ gives the node of the AVPO of option o and attribute a that is associated with the attribute value a of o_1. The cost is normalised in a similar manner as with the PSM. We use as maximum and minimum values those associated with indices of modifiers and their negation that tag any of the AVPO nodes. Finally, as there are two AVPOs, one associated with each option, the final cost is the average of their respective costs as shown in Equation 6-19.

$$AVPOAttCost(o, o_1, o_2, a) = \frac{| NodeVal(Node(AVPO[o, a], o_1)) - NodeVal(Node(AVPO[o, a], o_2)) |}{f_m(maxIdx(AVPO[o, a])) - f_m(minIdx(AVPO[o, a]))} \quad (6\text{-}18)$$

$$AttCost(o_1, o_2, a) = \frac{AVPOAttCost(o_1, o_1, o_2, a) + AVPOAttCost(o_2, o_1, o_2, a)}{2} \quad (6\text{-}19)$$

Goal, Lower Bound and Upper Bound

In the case that an option is considered worse than another (with respect to an attribute) due to a goal, upper or lower bound, we again exploit monadic preferences. Each attribute is associated with a domain, and we tag different domain values of attributes associated with goals (or lower and upper bound preferences) with values of modifiers of monadic preferences satisfied by the domain values. The tagging, according to the formula of monadic preferences, is as follows.

(i) $attribute = value$: the domain value $value$ is tagged with the value associated with the preference modifier.

(ii) $attribute > value_1$ **and** $attribute < value_2$: the domain values $value_1$ and $value_2$ are tagged with the preference modifier, plus and minus the difference from this modifier to the closest modifiers (as with AVPO nodes tagged with

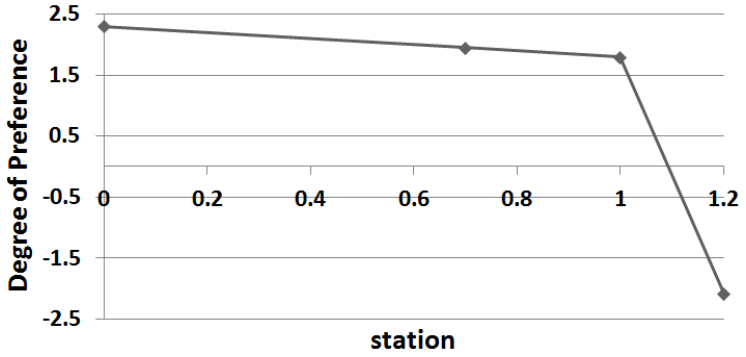

Figure 6.4: Tagging an attribute domain associated with a goal.

the same modifier). Note that it is possible to have ≥ and ≤ instead of > and <, respectively.

(iii) Domain boundaries, if not tagged, are tagged with the numeric values associated with *don't want* (minimum value), which is the minimum modifier that is not a hard constraint, and *want* (maximum value), in case of maximisation goal (or lower bound). If the goal is a minimisation (or upper bound), the boundaries tags are inverted. We do not use preferences with *don't accept* and *hate*, because these modifiers are hard constraints, but as in AVPOs, they are used to tag domain values only to keep consistency if there are monadic preferences that use them.

With this tagging, we keep the maximisation and minimisation goals but also associate a degree of preference (DoP) with specific domain values. For instance, based on preferences 3 and 4, we tag $station = 1.0$ with the value associated with *need*, and $station = 0.7$ with the value associated with *prefer*, obtaining the curve shown in Figure 6.4.

Now, we use these degrees of preference to measure attribute costs. As each domain value is now surrounded by two tagged values (or it is a tagged value), we are able to derive a degree of preference for all domain values, and the cost is the difference between them, normalised according to the maximum and minimum degrees of preference, which are given by the domain boundaries ($min(D_a)$ and $max(D_a)$).

Given an attribute value y_a, whose closest tagged attribute values are x_a, tagged with t_x, and z_a, tagged with t_z, we can calculate the parameters a and b of a linear function $DoP(x) = ax + b$, where $a = (t_x - t_z)/(x_a - z_a)$, and $b = t_x - ax_a$, and then calculate the degree of preference of y_a. If y_a is tagged, it already has an associate degree of preference.

$$AttCost(o_1, o_2, a) = \frac{\mid DoP(val(o_1, a)) - DoP(val(o_2, a)) \mid}{DoP(max(D_a)) - DoP(min(D_a))} \quad (6\text{-}20)$$

Even though goals can be either of maximisation or of minimisation, we use the same equation to calculate the attribute cost, as the difference is the same in both cases, and the cost is associated only with the option whose OAPM value is −.

Around

For assessing the cost using an around preference, we make a similar calculation as above, but many modifiers are not helpful in this case, as there is solely one monadic preference that is applicable to the options being compared — this is a requirement to apply the around implicit preference. We evaluate the attribute cost based on the difference of between attribute values and the reference value, which ranges from 0 (the attribute value is equal to the reference value, thus the distance from this value is 0) to the longest distance from the reference value, considering the attribute domain. As here we cannot use the difference between modifiers (as there is only one modifier, associated with the reference value), we use a function $f_d(dist)$ (currently, instantiated as a linear function), to evaluate cost in terms of the distance from the reference value, as shown in Equation 6-21, where p is the preference associated with the reason of the OAPM value $\langle around, p\rangle$.

$$AttCost(o_1, o_2, a) = \frac{f_d(\mid AroundDist(o_1, a, p) - AroundDist(o_2, a, p) \mid)}{f_d(max(\mid min(D_a) - RefVal(p) \mid, \mid max(D_a) - RefVal(p) \mid))} \tag{6-21}$$

Interval

Similarly to the around preference case, we assess the cost using an interval preference as a basis using the distance from a reference value, which is now an interval. The range of possible distances is from 0 (the attribute value is in the interval) to the longest distance from the interval extremes (considering the attribute domain), which are used by f_d to calculate cost. The difference between the attribute values from the compared options is given by $IntervalDist(o, a, p)$, introduced in Equation 6-12, where p is the preference associated with the reason of the OAPM value $\langle interval, p\rangle$.

$$AttCost(o_1, o_2, a) = \frac{f_d(\mid IntervalDist(o_1, a, p) - IntervalDist(o_2, a, p) \mid)}{f_d(max(\mid min(D_a) - lb(p) \mid, \mid max(D_a) - ub(p) \mid))} \tag{6-22}$$

Given these different ways of calculating the attribute cost $AttCost(o_1, o_2, a)$ based on the reasons for establishing a preference between attribute values of two options, we show the attribute costs for our running example in Table 6.7.

Overall Option Costs

Up to now, we have considered the costs of an option with respect to another by considering attributes in isolation, and now we look at the overall option costs

Table 6.7: Cost-benefit Analysis for the Apartment Decision Problem.

	w_i	**Ap_B**	**Ap_E**	**Ap_F**
uni	0.179	1.000	0.100	1.000
station	0.196			
market	0.132	0.071	0.214	0.071
zone	0.210	0.200		0.200
brand	0.094			
stars	0.030	0.027	0.027	0.054
price	0.158			
Cost		0.231	0.047	0.232

(a) $AttCost(Ap\_A, o', a)$

	w_i	**Ap_A**	**Ap_E**	**Ap_F**
uni	0.196			0.013
station	0.179	0.250	0.083	
market	0.132		0.143	
zone	0.210			
brand	0.094	1.000	1.000	1.000
stars	0.030			0.027
price	0.158	0.500	0.500	
Cost		0.218	0.207	0.098

(b) $AttCost(Ap\_B, o', a)$

	w_i	**Ap_A**	**Ap_B**	**Ap_F**
uni	0.179		1.000	1.000
station	0.196	0.166		
market	0.132			
zone	0.210		0.200	0.200
brand	0.094	0.018		
stars	0.030			0.027
price	0.158			
Cost		0.034	0.221	0.222

(c) $AttCost(Ap\_E, o', a)$

	w_i	**Ap_A**	**Ap_B**	**Ap_E**
uni	0.196			
station	0.179	0.500	0.250	0.333
market	0.132			0.143
zone	0.210			
brand	0.094	0.036		0.018
stars	0.030			
price	0.158	0.833	0.333	0.833
Cost		0.225	0.098	0.212

(d) $AttCost(Ap\_F, o', a)$

(also with respect to another option). This is performed by taking into account the priorities provided — which can be preference priority, attribute priority and attribute indifference — and building an attribute partial order ($attPO$) for each option, as different priorities can be applicable to different options.

As preferences, priorities pri also have a condition $cond(pri)$, and we present a similar applicability definition.

Definition 6.5 *A priority pri is applicable to an option o, $App(pri, o)$, if and only if*

$$\nexists cond(pri) \vee (\exists\, cond(pri) \wedge sat(cond(pri), o))$$

We initially take into consideration preference priorities (applicable to a particular option), which associates a number with preferences, meaning that the lower the number associated with the preference is, the more important it is. Each preference is related to a single attribute (according to our assumptions), and therefore the attribute order follows the order implied by the numbers associated with the preference, as presented in Algorithm 7 — as there may be many preferences associated with an attribute, we consider the lowest number.

Attribute priority and indifference modify the order given by preference priorities. First, if the attribute priority is inconsistent with the order of preference priorities, the attribute that was, before, considered less important becomes the attribute immediately more important than the other attribute referred in the given

Algorithm 7: $ProcessPreferencePriorities(priorities, allAtt)$

Input: $priorities$: preference priorities applicable to an option; $allAtt$: set of attributes
Output: $attPO$: attribute partial order

```
1  Set⟨Attribute⟩ N ← ∅;
2  Set⟨Arrow⟩ A ← ∅;
3  Set⟨Attribute⟩ parents ← ∅;
4  int i ← 1;
5  while priorities ≠ ∅ do
6      Set⟨Attribute⟩ currentAtt ← ∅;
7      while ∃ pri.(pri ∈ priorities∧ Priority(pri) = i) do
8          PreferencePriority pri ← Get(priorities, i);
9          Attribute a ← Attribute(pri);
10         if a ∉ N then
11             N ← N ∪ {a};
12             currentAtt ← currentAtt ∪ {a};
13         priorities = priorities\{pri};
14     if currentAtt ≠ ∅ then
15         foreach a ∈ currentAtt do
16             foreach p ∈ parents do
17                 A ← A ∪ {⟨p, a⟩};
18         parent ← currentAtt;
19     i ← i + 1;
20 foreach a ∈ (allAtt\N) do
21     N ← N ∪ {a};
22     foreach p ∈ parents do
23         A ← A ∪ {⟨p, a⟩};
24 return ⟨N, A⟩;
```

attribute priority. Algorithm 8 shows how this swapping process is performed. Second, if the attribute indifference is inconsistent with the order of preference priorities, the least important attribute becomes as important as the previously more important attribute. Algorithm 9 shows how this change is performed.

Finally, attributes associated with a *don't care* preference are excluded from attribute partial order. It is important to highlight that, as we assume consistency, priorities do not form a cycle.

In our running example, for option Ap_A , preference priorities result initially in the following order.

$$zone \succ \textbf{uni} \succ \textbf{station} \succ price \succ market \succ brand \succ stars$$

By considering the given attribute priority, $station$ and uni are swapped.

$$zone \succ \textbf{station} \succ \textbf{uni} \succ price \succ market \succ brand \succ stars$$

Given the attribute order, we consider the least important attributes as having the level 1 in the order, and the longest path in the order from the least important attributes to the most important ones is referred to as $length(attPO)$. Then, we use

Algorithm 8: $MoveAbove(att_1, att_2, A)$

Input: att_1, att_2: attributes to be swapped; A: attPO arrows

```
1  Set⟨Attribute⟩ oldParents ← Parents(att1) ;
2  Set⟨Attribute⟩ oldChildren ← Children(att1) ;
3  foreach p ∈ oldParents do
4      foreach c ∈ oldChildren do
5          A ← A ∪ {⟨p, c⟩};
6      A ← A\{⟨p, att1⟩};
7  foreach c ∈ oldChildren do
8      A ← A\{⟨att1, c⟩};
9  foreach p ∈ Parents(att2) do
10     A ← A ∪ {⟨p, att1⟩};
11     A ← A\{⟨p, att2⟩};
12 A ← A ∪ {⟨att1, att2⟩};
13 foreach p ∈ oldParents do
14     if ¬ExistsPath(att1, p) ∧ ¬ExistsPath(p, att1) then
15         A ← A ∪ {⟨p, att1⟩};
16 foreach c ∈ oldChildren do
17     if ¬ExistsPath(att1, c) ∧ ¬ExistsPath(c, att1) then
18         A ← A ∪ {⟨att1, c⟩};
```

Algorithm 9: $MoveEqual(att_1, att_2, A)$

Input: att_1, att_2: attributes to be swapped; A: attPO arrows

```
1  Set⟨Attribute⟩ oldParents ← Parents(att1) ;
2  Set⟨Attribute⟩ oldChildren ← Children(att1) ;
3  foreach p ∈ oldParents do
4      foreach c ∈ oldChildren do
5          A ← A ∪ {⟨p, c⟩};
6      A ← A\{⟨p, att1⟩};
7  foreach c ∈ oldChildren do
8      A ← A\{⟨att1, c⟩};
9  foreach p ∈ Parents(att2) do
10     A ← A ∪ {⟨p, att1⟩};
11 foreach c ∈ Children(att2) do
12     A ← A ∪ {⟨att1, c⟩};
13 foreach p ∈ oldParents do
14     if ¬ExistsPath(att1, p) ∧ ¬ExistsPath(p, att1) then
15         A ← A ∪ {⟨p, att1⟩};
16         A ← A ∪ {⟨p, att2⟩};
17 foreach c ∈ oldChildren do
18     if ¬ExistsPath(att1, c) ∧ ¬ExistsPath(c, att1) then
19         A ← A ∪ {⟨att1, c⟩};
20         A ← A ∪ {⟨att2, c⟩};
```

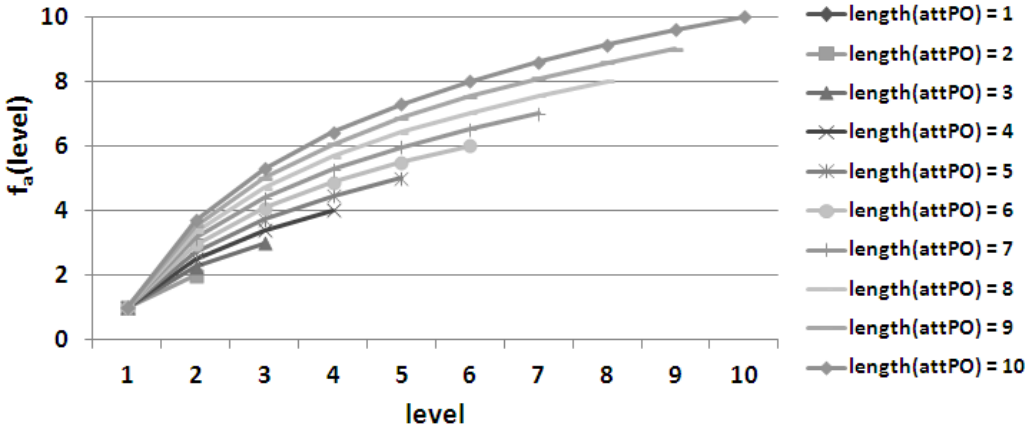

Figure 6.5: Attribute weights calculated with the logarithmic function.

a logarithmic function ($f_a(x) = \alpha \log x + \beta$) for calculating the attribute weights when considering the overall option benefits. We establish the following points for the function.

$$f_a(1) = 1 \tag{6-23}$$

$$f_a(length(attPO)) = length(attPO) \tag{6-24}$$

Point 6-23 indicates that attributes in the first level of the order have the minimum weight, which is 1, and point 6-24 shows that attributes in the last level have the maximum weight, which is $length(attPO)$. The logarithmic function, with the characteristics imposed by the points we established, gives a much higher priority to more important attributes, and these more important attributes have a smaller difference among them (in comparison with a linear function). This behaviour is shown in Figure 6.5, which shows the weight logarithmic function for one to ten levels of attributes. This is a default form we are adopting for calculating attribute weights, which was also selected based on experimentation, and it is a variation point. Next, we present how we calculate the parameters α and β of the logarithmic function for a particular level of attributes.

$$\alpha = \frac{length(attPO) - 1}{\log length(attPO)} \tag{6-25}$$

$$\beta = 1 \tag{6-26}$$

Based on the logarithmic function with the calculated parameters, the weight of each attribute $a_i \in Att$ is as shown below.

$$w_i = \frac{f_a(level(a_i))}{\sum_{a_j \in Att} f_a(level(a_j))}$$

Finally, now that we have the costs of an option o_1 with respect to o_2, for each individual attribute, and we also have the attributes weights, we calculate the overall benefits from o_1 with respect to o_2 using a weighted sum, as presented next. This function, which denotes the costs of all options w.r.t. each other option, calculated for our running example is shown in the last row of Table 6.7, which also details the attribute weights for each option.

$$Cost(o_1, o_2) = \sum_{a_i \in Att} w_i \times AttCost(o_1, o_2, a_i)$$

6.7.2 Trade-off Contrast

The result of not having dominated options in the set of acceptable options is that for any two options, one option is better for one or more attributes and the same applies to the other. As a consequence, a trade-off must be resolved for choosing one of the two options. According to Simonson and Tversky (Simonson and Tversky 1992), when people make choices they do not look only for the two options being compared, but analyse the cost-benefit relationship between two options compared with the cost-benefit relationship between all other options. This reasoning of comparing the trade-offs of the whole set of options is referred to as *trade-off contrast*, and is not in accordance with traditional decision making theory as it states that the preference between two options is independent of the other available options.

Therefore, we incorporate a new factor in the process of choosing an option, which is captured by a function that shows the trade-off between two options.

$$to : Opt \times Opt \rightarrow \mathbb{R}$$

We build the trade-off (to) as a partial function whose domain is every pair of options that satisfies $Cost(o_1, o_2) < Cost(o_2, o_1)$ and is associated with the options' cost-benefit relationship: $Cost(o_1, o_2)/Cost(o_2, o_1)$. Because $Cost(o_1, o_2) < Cost(o_2, o_1)$, to is always value in the interval [0, 1] and $Cost(o_2, o_1)$ cannot be 0. The average of all values of to is referred to as avg_{to}.

The trade-off between two options does not have a meaning in an isolated manner; when we have only two options, all we know is that one option has higher or lower cost than another. When there are other options, and the decision maker observes that the cost-benefit relationship is better for other options, this is seen as a negative aspect of the option and the benefits become smaller. That is, the option requires giving too much for receiving just a little in exchange.

Table 6.8: Trade-off Analysis for the Apartment Decision Problem.

	Ap_B	**Ap_E**	**Ap_F**
to			
ToContrast		0.191	

(a) Trade-off of Ap_A

	Ap_A	**Ap_E**	**Ap_F**
to	0.943	0.937	
ToContrast	0.017	0.017	

(b) Trade-off of Ap_B

	Ap_A	**Ap_B**	**Ap_F**
to	0.731		
ToContrast			

(c) Trade-off of Ap_E

	Ap_A	**Ap_B**	**Ap_E**
to	0.969	0.999	0.956
ToContrast	0.052	0.052	0.052

(d) Trade-off of Ap_F

Given the structure we built to store trade-offs, to, we now calculate the option costs with respect to trade-off, having as a basis the average of the trade-off between a particular option with the others — which is represented by the average of these trade-offs $avg_{to}(o)$ — and the trade-off among all options (which is represented by the average of all trade-offs avg_{to}). If $Cost(o_1, o_2) < Cost(o_2, o_1)$ and the trade-off relationship of o_1 is higher (i.e. worse) than avg_{to}, then we have one more cost of o_1 w.r.t. o_2. If the trade-off is lower (i.e. better) than the average, than it is counted as a benefit, and therefore as a cost for o_2. The function $ToContrast(o_1, o_2)$, which captures this notion of trade-off contrast, is shown below.

$$ToContrast(o_1, o_2) = \begin{cases} avg_{to}(o_1) - avg_{to} & \text{if } to(o_1, o_2) \text{ is defined} \\ & \text{and } avg_{to}(o_1) > avg_{to} \\ avg_{to} - avg_{to}(o_1) & \text{if } to(o_2, o_1) \text{ is defined} \\ & \text{and } avg_{to}(o_2) < avg_{to} \\ 0 & \text{otherwise} \end{cases} \quad (6\text{-}27)$$

The function $Cost$ calculated for our running example allows us to analyse the trade-off among options (to), which is shown in Table 6.8. The to averages are: avg_{to} = 0.923, $avg_{to}(Ap\_B)$ = 0.940, $avg_{to}(Ap\_E)$ = 0.731 and $avg_{to}(Ap\_F)$ = 0.975. Based on that we can calculate the trade-off contrast ($ToConstrast$), which is also presented in this table.

6.7.3 Extremeness Aversion

Another aspect that people take into consideration when making a decision is how extreme options are. Extreme options are those that have a large improvement for one attribute (or set of), e.g. quality, and a high penalisation for another attribute (or set of), e.g. price. In general, people avoid extreme options (Simonson and Tversky 1992), and this is referred to as *extremeness aversion*.

In order to evaluate how extreme options are, we compare option attribute values to the best possible values, measuring the distance between them. As best

values are subjective to each individual, we use the preferences applicable to each option to identify the best value for each attribute. As these values are better or equal to the particular option being analysed o, each attribute value of o can be associated with an attribute cost, which ranges from 0, i.e. the attribute value is equal to the best value, to 1, i.e. the attribute value is the worst possible value. Each of these costs is referred to as distance from best, or $bestDist(o, a)$.

The procedure is similar to making a cost-benefit analysis of the option being analysed with a hypothetic option whose attribute values are the best. Preferences to identify best values are processed in the inverse order of that used to build the OAPM, consequently we will keep the same precedence order, as earlier processed preferences may have their OAPM value overridden. Best values are identified in the following way, when attribute a of option o is being analysed. If there is a *don't care* preference associated with a (and is applicable to o), a is not taken into account.

(i) If $val(o, a)$ has an associated node in the $AVPO(o, a)$, the best value is that related to a source node of the AVPO. If more than one source node exists, we use the one with the highest associated numeric value.

(ii) If there is a goal associated with a (and the goal is applicable to o, we use the domain lower bound ($min(D_a)$) in case of a minimisation goal, and the domain upper bound ($max(D_a)$), otherwise.

(iii) If there are monadic preferences applicable to o, i.e. there is a modifier associated with its PSM value, we do not specify a particular best value but a PSM value that would be associated with the best value, which is $\langle \epsilon, m \rangle$ or $\langle \neg, m \rangle$, whose index is the highest and m is the modifier of a monadic preference applicable to o.

(iv) If none of the above can be applied, or monadic preferences are inconclusive to measure the distance between the attribute value and the best value, i.e. they are considered similar, we use implicit preferences. Then, the best value is: (a) $min(D_a)$, in case of an upper bound, (b) $max(D_a)$, in case of a lower bound; (c) $RefVal(p)$, in case of an around preference; and (d) either $lb(p)$ or $ub(p)$, which are an interval boundaries, in case of an interval preference.

Given this way of identifying best values, we calculate $bestDist(o, a)$ in the same way that attribute costs are calculated, but comparing options with best values instead of other options.

An extreme option has low costs for some attributes ($bestDist(o, a)$ close to 0) and high costs for others ($bestDist(o, a)$ close to 1), therefore we evaluate how extreme the option is by calculating the standard deviation of the function $bestDist$

for a particular option, for all attributes, which is a value between 0 and 1.

$$ext(o) = STDEV(\{bestDist(o, a_i) \mid i = 1...|Att|\})$$

The acceptable options of our apartment example are ordered according to their extremeness in the following way (from the least extreme to the most extreme): $ext(Ap\_B) = 0.344$, $ext(Ap\_E) = 0.346$, $ext(Ap\_A) = 0.361$, and $ext(Ap\_F) = 0.403$.

Finally, as the more extreme the option is, the more people avoid it, it is considered that the more extreme option, between two options, has a cost with respect to the other option. So, in order to capture this aspect, we define $ExtAversion : Options \times Options \rightarrow \mathbb{R}$, which represents the cost of the first option compared to the second, with respect to the extremeness aversion principle. This function, presented below, shows how the extremeness aversion is calculated: the more extreme option has a cost that is the difference between the extremeness values of the two options, and, as the less extreme option has no cost with respect to the other, the value is 0.

$$ExtAversion(o_1, o_2) = \begin{cases} ext(o_1) - ext(o_2) & \text{if } ext(o_1) > ext(o_2) \\ 0 & \text{otherwise} \end{cases} \quad (6\text{-}28)$$

6.7.4 The Decision Function: Comparing Relative Option Values

After executing the previous steps, we have analysed three aspects when comparing options: their costs, the trade-off relative to the set of available options, and how extreme they are. The last two aspects are also seen as costs (or benefits): if the trade-off is higher than the average, it is also considered as a cost, and a more extreme option has a cost when compared to a less extreme. So the final value of an option with respect to another combine these three aspects in a weighted sum of these costs, which can be seen in Equation 6-29, comprising our *decision function* — $d(o_1, o_2)$. We are now considering default weights (the last variation point of our technique), which are 0.25 for trade-off contrast and 0.15 for extremeness aversion. Based on the $d(o_1, o_2)$ function, we identify the chosen option, which is the option that has less or equal disadvantages ($d(o_1, o_2) \leq d(o_2, o_1)$) than every other option of the $Acceptable$ set, i.e. those that are better or equal to the other options. If different options have the same decision value with respect to another ($d(o_1, o_2) = d(o_2, o_1)$), and they are better than every other option, we randomly choose one of them.

$$\begin{aligned} d(o_1, o_2) = {} & (1 - w_{to} - w_{ea}) \times Cost(o_1, o_2) \\ & + w_{to} \times ToContrast(o_1, o_2) + w_{ea} \times ExtAversion(o_1, o_2) \end{aligned} \quad (6\text{-}29)$$

Table 6.9: Options for Illustrating the Impact of the User-centric Principles.

	A	**B**	**C**
uni	0.5 Km	1.75 Km	3.0 Km
price	£150	£100	£50

(a) Extremeness Aversion.

	A	**B**	**C**
uni	0.9 Km	1.8 Km	3.0 Km
price	£125	£95	£90

(b) Trade-off Contrast.

In order to demonstrate the effect of the trade-off contrast and extremeness aversion, we use an example that is smaller than our running example, but also involving a choice among apartments. The apartments are now described only in terms of the distance from the university (uni) and price ($price$), both real numbers, whose domains are [0.5, 3.0] and [£50, £150]. The preferences provided by the user are two goals: minimise the value of both attributes, and these attributes are equally important. For showing the impact of extremeness aversion, let A, B and C be three options, whose attribute values are detailed in Table 6.9(a).

By calculating option costs and benefits, we find out that the option costs are amortised for all the options, when they are compared in a pairwise fashion, i.e.

- $AttCost(A, B, uni)$ = 0.5 and $AttCost(B, A, price)$ = 0.5,
- $AttCost(B, C, uni)$ = 0.5 and $AttCost(C, B, price)$ = 0.5, and
- $AttCost(A, C, uni)$ = 1.0 and $AttCost(C, A, price)$ = 1.0.

Therefore, if we calculate the value of one option with respect to another without taking into account the extremeness aversion principle, we conclude that options are equally good, as $d(o_1, o_2) = d(o_2, o_1)$ for all the options. However, if we calculate the distance from the best attribute values for each available option, we can verify that $bestDist(A, uni) = 0$ and $bestDist(A, price) = 1$, and therefore $ext(A) = 0.5$. B, as it has the intermediate values, has its extremeness evaluated to 0 (the best distance for both attribute values is 0.5), and C has the opposite values of A, having its extremeness also evaluated to 0.5. As a consequence, now A and C have a cost with respect to B, which is chosen as the optimal option.

Given that we showed the impact of the extremeness aversion isolated from the trade-off contrast, we now introduce another set of options to explain the impact of the latter, described in Table 6.9(b), using the same preferences. First, we calculate the $Cost$ function in order to identify the costs of each option, which is composed of the weighted sum of the costs identified for each individual attribute. The first rows of Table 6.10 show these calculated values — weights used are 0.5 for both uni and $price$, as they are equally important. Considering solely these costs, A is the optimal option, as it has less costs than B and C, and B is better than C.

By comparing the trade-off, i.e. the ratio between benefits and cost, when the cost is smaller than the benefit, we can see that the cost paid to choose A

Table 6.10: Options for Illustrating the Impact of Trade-off Contrast.

	A	**B**	**B**	**C**	**A**	**C**
uni	0.0	0.36	0.0	0.48	0.0	0.84
$price$	0.3	0.0	0.05	0.0	0.35	0.0
$Cost(o_1, o_2)$	0.15	0.18	0.025	0.24	0.175	0.42
$to(o_1, o_2)$	0.833		0.104		0.416	
$ToContrast(o_1, o_2)$	0.174	0.0	0.000	0.347	0.174	0.0
$v(opt_1, opt_2)$	0.156	0.135	0.019	0.267	0.175	0.630
Balance	**0.021**			**0.248**		**0.455**

instead of B is very high to get the provided benefits (0.833), in comparison to the trade-off between B and C (0.104), and A and C (0.416). Therefore, as people tend to adopt the trade-off contrast principle (Simonson and Tversky 1992), choosing A seems to be not a "good deal," because the costs are too high to get A's benefits, in comparison with other options.

So, by considering the average (0.451) of the to function as a threshold for considering the trade-off relation as a benefit or cost, we now have the value of $ToContrast$ shown in its respective row, which is incorporated into the option costs using our default weight (0.25). Finally, the optimal and chosen option is now option B. The adoption of this principle is consistent with our previous study (Chapter 2) as many of the participants pointed out in their preferences that their choice is based on a "good cost-benefit relationship."

As it can be seen in our provided examples, both the extremeness aversion and trade-off contrast are incorporated as costs or benefits using weights, in order to evaluate the decision value of an option with respect to another. However, the individual costs associated with extremeness aversion and trade-off contrast that are added to this value may not be high enough to make the order established by the $Costs$ function change. Even together, the extremeness aversion and trade-off contrast may still not be high enough to make this change. Therefore, it is the interplay among the option costs, extremeness aversion and trade-off contrast that specify which option of every two options is preferred, and consequently make it possible to determine the chosen option.

Considering our apartment example, if the $Costs$ function were the only factor taken into account, the apartment Ap_F would have been chosen — note that $Cost(Ap\_B, Ap\_F)$ = 0.09768 and $Cost(Ap\_F, Ap\_B)$ = 0.09760. Nevertheless, by considering the other two principles we are adopting, we have a different result. The costs of Ap_B and Ap_F are almost equal, but Ap_B is the least extreme option, while Ap_F is the most extreme: Ap_F has the best values for some attributes (uni, $zone$ and $stars$), but a high penalisation for others ($station$ and $price$). Moreover, by analysing the cost-benefit relationship between options, we identify that the relationship between Ap_F and Ap_B, which is 0.999, is worse

Table 6.11: Decision Function of the Apartment Decision Problem.

	B	**E**	**F**
Cost	0.231	0.047	0.232
ToConstrast	0.000	0.191	0.000
ExtAversion	0.017	0.015	0.000
d	0.141	0.078	0.139

(a) Ap_A

	A	**E**	**F**
Cost	0.218	0.207	0.098
ToConstrast	0.017	0.017	0.000
ExtAversion	0.000	0.000	0.000
d	0.135	0.129	0.059

(b) Ap_B

	A	**B**	**F**
Cost	0.034	0.221	0.222
ToConstrast	0.000	0.000	0.000
ExtAversion	0.000	0.002	0.000
d	0.021	0.133	0.133

(c) Ap_E

	A	**B**	**E**
Cost	0.225	0.098	0.212
ToConstrast	0.052	0.052	0.052
ExtAversion	0.042	0.060	0.057
d	0.154	0.080	0.149

(d) Ap_F

than the average 0.922. Table 6.11 shows the values of all functions calculated for every pair of options of the $Acceptable$ set, and by considering the weighted sum of the costs, trade-off contrast and extremeness aversion, the **chosen option is** Ap_B.

Our criterion to choose the optimal option is the balance between costs and benefits of each option with respect to another. As this is calculated in a pairwise fashion, there may be situations in which there is a cycle, that is, $d(o_1, o_2) < d(o_2, o_1)$, $d(o_2, o_3) < d(o_3, o_2)$ and $d(o_3, o_1) < d(o_1, o_3)$. This situation arises because we use different criteria to compare the attribute values of each pair of options, for example, the price of o_1 and o_2 is compared based on a goal, and o_1 and o_3 based on a monadic preference. As user preferences are consistent, if we analyse only the price attribute, we will find no cycles; however, as the d function is calculated in a different manner according to different preferences, small differences in the scales may lead to a cycle when considering the overall option costs.

In order to choose one option when this situation occurs, we adopt the following strategy. We identify the set of options that are considered better than the highest number of options, and then, from these, we choose the one with the minimum-maximum balance for every option that is considered better than it. In other words, if option o can be chosen as the optimal option, and for every option o' that $d(o, o') > d(o', o)$, we calculate the maximum value for the difference between $d(o, o')$ and $d(o', o)$. And if this value is the lowest one when compared to the value of every other candidate option (options that are better than the same amount of options of o), than o is chosen.

In our experiment (see next section), which was performed with real user data, there was only 1 (of 113) occurrence of cycle. Even though this number is low, we use a workaround to solve this issue, and it is part of future work to completely eliminate the possibility of cycles.

Finally, we discuss the complexity of our technique. As it can be observed in

Table 6.12: Complexity Analysis of our Technique.

Pre-processing	
PSM	$O(\mid Opt \parallel P_e \mid)$
OAPM	$O(\mid Opt \mid^2 \mid Att \parallel P_e \mid)$
AVPO	$O(\mid Opt \mid^2 + \mid P_e \mid^2)$
Explication	$O(\mid Opt \mid^2 \mid Att \parallel P_e \mid)$
Elimination	$O(\mid Opt \mid^2 + \mid Opt \parallel Att \mid)$
Selection	
Cost	$O(\mid Opt \mid^2 \mid Att \mid + max(PP_i) \mid PP_i \mid + \mid P_i \parallel Att \mid)$
Extremeness Aversion	$O(\mid Opt \parallel Att \mid + \mid Opt \mid^2)$
Trade-off Contrast	$O(\mid Opt \mid^2)$
Decision Function	$O(\mid Opt \mid^2)$

the presented algorithms, our technique runs in polynomial time, and most of the algorithms require comparing each pair of options according to each attribute. We present the complexity of each part of our technique in Table 6.12, whose total is

$$O(\mid Opt \mid^2 \mid Att \parallel P_e \mid + \mid P_e \mid^2 + max(PP_i) \mid PP_i \mid + \mid P_i \parallel Att \mid) \quad (6\text{-}30)$$

where

- Opt is the set of available options,
- Att is the set of attributes,
- P_e is the set of preferences,
- PP_i is the set of preference priorities,
- $max(PP_i)$ is the maximum number associated with the preference priorities, and
- P_i is the set of attribute priorities and attribute indifferences.

6.8 Comparison with Related Work and Evaluation

One of the key advantages of our approach is the ability to handle different types of preferences. In this section, we thus compare our technique with existing approaches to reasoning about preferences in terms of the preference types they can handle. We also evaluate our approach empirically by comparing our choices with those of a human expert. As the input of our technique is high-level preferences, and existing approaches cannot handle all of them, our empirical evaluation does not make side-by-side comparison with existing work.

As discussed in previous chapter, many existing approaches are based on *utility functions (UFs)*. As with UFs it is possible to order available options, thus choosing among them, different approaches (McGeachie and Doyle 2008,

Table 6.13: Reasoning Approaches vs. Preferences.

Approach	Preference								Priority		
	Cd	Ct	G	O	Q	R	I	D	a	i	p
UF-based (McGeachie and Doyle 2008)				X							
SVM-based (Domshlak and Joachims 2007)				X		X	X				
SCSP (Bistarelli et al. 1997)		X									X
Bipolar preferences (Bistarelli et al. 2010)		X									X
Interval-valued SCSP (Gelain et al. 2010)		X									X
CP-Nets (Boutilier et al. 2004)	X			X							
TCP-Nets (Brafman et al. 2006)	X			X					X		
Scoring Function (Agrawal and Wimmers 2000)		X				X		X			X
Winnow (Chomicki 2003)		X		X			X				X
BMO (Kießling 2002)		X		X		X					X
Query Personalisation (Koutrika and Ioannidis 2006)						X					X
SPARQL (Siberski et al. 2006)		X	X								X

Legend — Cd: condition; Ct: constraint; G: goal; O: order; Q: qualifying; R: rating; I: indifference; D: don't care; a: attribute priority; i: attribute indifference; p: preference priority.

Domshlak and Joachims 2007) have been proposed to transform specific models that capture qualitative preferences (which are closer to how users express preferences) into UFs, i.e. quantitative preferences, which are consistent with the constraints established by the qualitative preferences. Some approaches (Bistarelli et al. 1997) extend Constraint Satisfaction Problems (CSPs) to incorporate *soft constraints* (that can remain unsatisfied), namely SCSP, associating a penalty (or preference) with each constraint, and creating an optimisation problem of minimising penalty (or maximising preference). In order to allow the representation of other kinds of preferences, extensions to traditional soft constraints approaches were proposed: (i) bipolar preferences (Bistarelli et al. 2010), which distinguish what users want (indicating preferred options), and what they do not want (restricting the set of acceptable options); and (ii) use of intervals (Gelain et al. 2010) to represent penalties (or preferences), as it may be difficult to specify precise values. A third group of approaches, mainly represented by CP-Nets (Boutilier et al. 2004) and TCP-Nets (Brafman et al. 2006), takes another direction, proposing new *graphical structures* to represent and reason about qualitative preferences. Finally, work in the area of databases proposes *extensions of query languages* (Agrawal and Wimmers 2000, Chomicki 2003, Kießling 2002, Koutrika and Ioannidis 2006, Siberski et al. 2006) to incorporate preferences and algorithms to provide query results according to specified preferences. Even though these approaches propose different solutions, they share the common goal of making a choice based on preferences. However, as shown in Table 6.13, they address limited kinds of preferences, restricting their natural expression by humans. Our technique is the only one that exploits natural language expressions — expressive speech acts — to make decisions on behalf of users.

Besides being able to deal with restricted kinds of preferences in comparison to our technique, these approaches only choose between two options when the preferences provided are sufficient to make the decision, i.e. if the decision involves

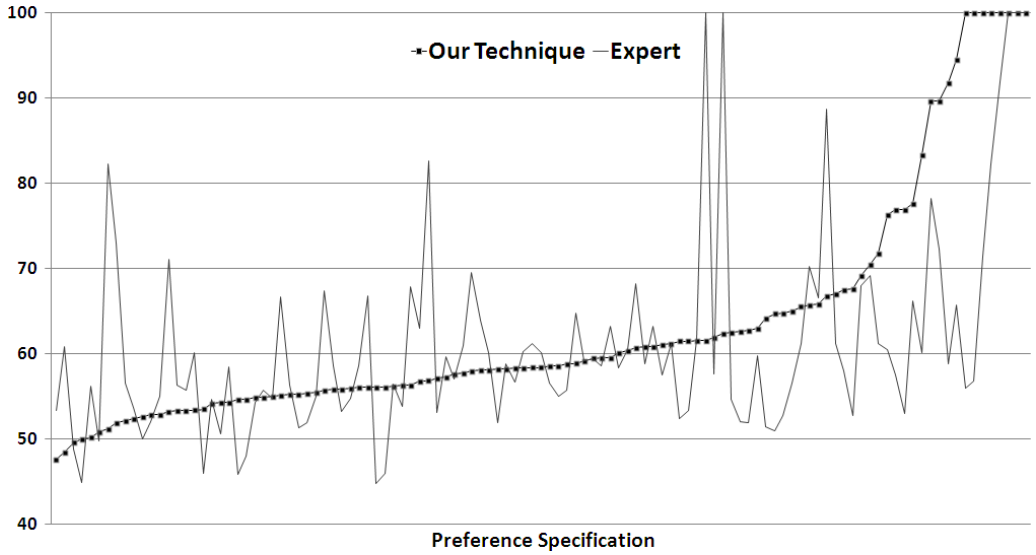

Figure 6.6: Expert vs. our technique: first choices.

trade-off, users must have previously resolved it and specified their preferences. However, as discussed by Tversky (Tversky 1996), people resolve trade-offs in light of available options, and do not provide such preferences. Our technique, on the other hand, resolve trade-offs using (i) preferences over individual attributes; (ii) priorities; and (iii) user-centric principles, with the aim of performing a similar reasoning that humans do.

Finally, note that this thesis is not concerned with preference elicitation methods, as our goal is to provide a preference language as close as possible to natural language, so that users can directly express their preferences, and based on such preferences make a choice. However, we do not exclude the possibility of combining our approach with elicitation methods, which can be simpler if the gap between provided user preferences and the preference model (or language) in which elicited preferences are captured is reduced, as our approach proposes.

The empirical evaluation of our technique is based on the study that also informed the preference language itself (Chapter 2). Participants provided preference specifications (in natural language) for use by an individual to buy a laptop on their behalf. Both these individuals, and domain expert, were given a laptop catalogue (with 144 laptops) from which to choose up to five options. The three relevant parts of the study used for our evaluation are the initial preference specification, the user choices and the domain expert recommendation. We use these to compare our decisions against those of the user and domain expert. Similarly to how the domain expert recommendation was assessed in the study, we calculate a similarity score SS, which compares the recommendation with the user choice, using Equation 2-1 — it takes into account the position of the up to five chosen laptops using a weighted average. SS ranges from 0 to 100, with 100 indicating a

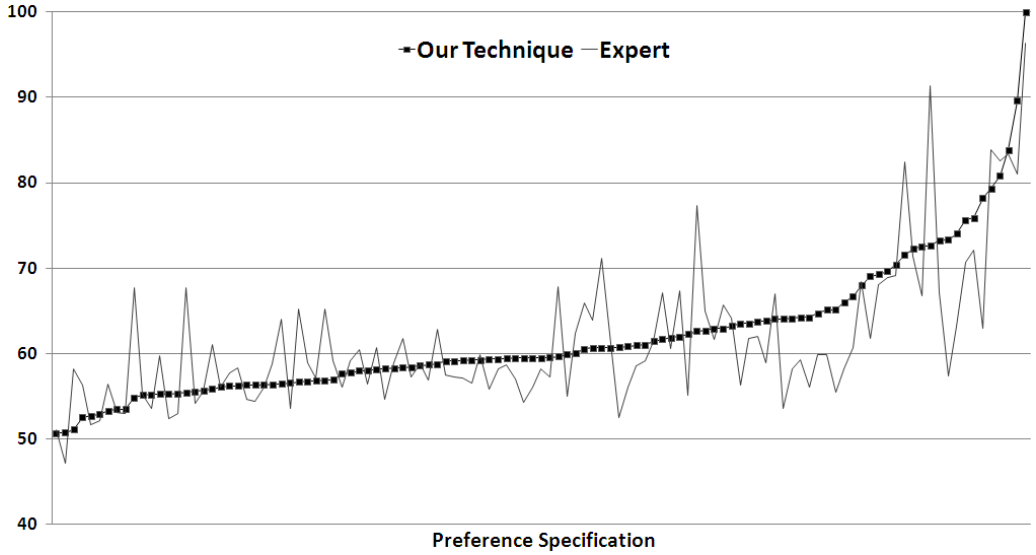

Figure 6.7: Expert vs. our technique: up to five choices.

match to user choice.

Using a graphical user interface developed to input preferences according to our preference language, we were able to store the preferences provided by 113 participants. Of the 192 user specifications, 79 (41%) use subjectivity or purpose, and therefore cannot be expressed in our language. For example, "*I'd like a laptop to carry on my backpack*." Moreover, of these 79, 9 have no expert recommendation, as they are too vague, such as "*I would never delegate this task [buy a laptop] to another person*." For the remaining specifications, we applied our technique (which takes an average of 3.6026 $seconds$ on an Intel Core i5 2.30GHz, 8GB of RAM, with standard deviation 1.4051, to be executed for each request, with 144 laptops, and 61 attributes), and obtained the similarity scores shown in Figures 6.6 and 6.7. The former shows the similarity score considering the first expert choice and the first choices of our technique compared to the first user choice, and the latter shows the comparison between the first up to five choices. If the domain expert recommended x laptops, we use the first x choices of our technique. Even though our technique does not rank acceptable options, as we calculate a numeric value to compare them, we use these numbers to obtain the first up to five choices. Both charts have their *x-axis* ordered according the SS calculated for our technique.

The results show that the values obtained for the (human) domain expert and our technique are not so different — considering only the first choices (labelled as *F*), our technique has M_{SS} = 63.19 and the domain expert has M_{SS} = 61.25; i.e. our technique has a better average SS than the expert. The same occurs for the up to five choices (labelled as *5*). These SS values, as well as standard deviation, minimum and maximum, are summarised in Table 6.14. The difference between the obtained similarity scores is *significant* when comparing only first choices, as determined by

Table 6.14: Analysis of Domain Expert and our Technique choices.

	Our Technique (F)	Expert (F)	Our Technique (5)	Expert (5)
Average	63.19	61.25	61.94	61.44
Standard Deviation	13.36	11.93	8.00	8.32
Minimum	47.68	44.80	50.72	47.12
Maximum	100.00	100.00	100.00	96.39

a Wilcoxon Signed Ranks test — $W(112) = 3711, p = 0.0497$ (F), and thus we reject the null hypothesis that domain expert and our technique choices are equal. However, this is not the case of the up to five choices: $W(112) = 3774, p = 0.1131$ (5). As a consequence, *we can conclude that our technique makes choices at least as good as those of the domain expert*.

6.9 Final Remarks

Previous studies (Lichtenstein and Slovic 2006) state that people have a set of preferences that they are aware of, which are used as a basis to make decisions and, when facing concrete decision making situations (Tversky 1996), they construct new preferences to resolve trade-offs that cannot be resolved with this set of known preferences. In this chapter, we presented an automated decision making technique that uses preferences expressed by users in a high-level language, which is close to how people express their known preferences in natural language. The technique resolves trade-offs based on priorities, which indicate attributes they consider more important, combined with user-centric principles, thus making decisions in a way similar to how humans do, with the aim of automating tasks on their behalf. Our technique makes a decision in a stepwise fashion: first, it evaluates preferences over individual attributes; second, it eliminates dominated options and those that do not satisfy cut-off values, obtaining a consideration set, which contains options that require trade-off resolution; third, it chooses one option from the consideration set evaluating option pros and cons, trade-off contrast and extremeness aversion, being the latter two the main principles from psychology we adopt. The empirical evaluation of our technique showed that it is able to make a choice on behalf of the users at least as good as that made by a human domain expert, considering our experiment.