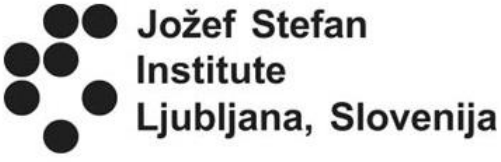
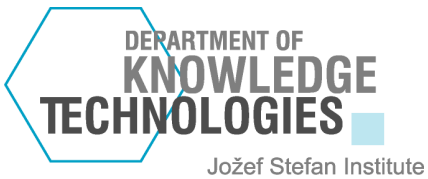
IJS delovno poročilo
DP-15041
2025

Marko Bohanec

Ranking of Alternatives in Qualitative Multi-Criteria Decision Modeling Method DEX





Abstract

We investigate methods for ranking decision alternatives in the context of multi-criteria decision modeling method *DEX* (Decision EXpert). DEX is a qualitative, hierarchical and rule-based method, particularly suitable for *sorting*: assigning alternatives into predefined categories based on their performance. Here, we study methods that extend DEX's capability to *ranking*: ordering alternatives from best to worst. This is achieved by introducing a dual evaluation of alternatives, which run in parallel: *qualitative* evaluation, which assigns alternatives to qualitative classes, and *quantitative* evaluation, which ranks alternatives within each class. The approach employs the *principle of dominance*, which allows identifying "better-than-or-equal-to" preferential relations between decision rules that map to the same qualitative class. The evaluation model is *constructed automatically* using only information that is already contained in the qualitative DEX model; no further information from the decision maker is requested. Other important aspects of the approach are:

1. *Consistency of qualitative and quantitative evaluations*: For each alternative, assigned to class C, the corresponding numerical evaluation lies in the interval $[c - 0.5, c + 0.5]$, where c is the ordinal number of C. The values $c + 0.5$ and $c - 0.5$ are interpreted as "ideal" and "anti-ideal" evaluations within C, respectively.
2. *Compatibility of inputs and outputs*: DEX models are hierarchical, therefore evaluations obtained as outputs at model subtrees enter as inputs to the higher levels of the hierarchy. Thus, the quantitative scales of input and output attributes must match and should obey the $c \pm 0.5$ principle.
3. *Quantitative evaluations preserve the preferential dominance between alternatives*.

The main contributions of this study are two novel ranking algorithms, called QQ2 and QL, which improve over a previous Qualitative-Quantitative (QQ) method. Both algorithms employ optimization models to assign numeric values to decision rules of some class in order to separate them as much as possible. QL and QQ2 are formulated in terms of a linear and quadratic optimization problem, respectively, with linear constraints. They do not require calculating attributes' weights, which was a weak point of QQ. They also better cover the class range and allow for proper handling of preferentially unordered attributes. The three algorithms were experimentally evaluated and compared on 3322 DEX real-life decision tables. Both QL and QQ2 significantly outperformed QQ in terms of within-class separation of decision rules. QL and QQ2 themselves turned out very similar, with no clear winner. Because of the small size and elegance of the quadratic model, and its time efficiency, QQ2 was eventually chosen for implementation in DEXiWin modelling software.

Keywords

Multi-criteria model, method DEX, decision rules, decision alternatives, sorting, ranking, linear optimization, quadratic optimization.

Contents

1 Introduction

Multi-criteria decision modeling (MCDM) is an approach used to make decisions when there are multiple, often conflicting, criteria or factors to consider (Greco, et al., 2016). By using MCDM, a decision maker can break down complex decisions into smaller, manageable parts, making it easier to compare options and choose the one that best meets their needs. Typically, MCDM proceeds (Belton, Stewart, 2002) by defining a set of *criteria* that are relevant to the decision at hand. These criteria are then incorporated into a structured *model* that reflects the decision maker's *preferences* and *priorities*. This model is used to systematically *evaluate* and *compare* different decision options, often referred to as *alternatives*. The process may also include a detailed *analysis* of these alternatives, which helps in exploring the decision space more thoroughly. Such analysis can reveal important insights into the characteristics of each alternative and the relationships between them. Ultimately, the outcomes of this process provide valuable information to *support* the decision maker in making a well-informed final decision.

According to Roy (1996; 2016), there are three main types of decision problems:

1. *Choosing* (Selection Problematic): To select the best alternative (or a small subset of alternatives) from a set of available ones. Example: Choosing the best car to buy based on criteria like price, fuel consumption, and safety.
2. *Ranking* (Ordering Problematic)*:* To rank all alternatives from best to worst based on their performance across multiple criteria. Example: Ranking universities based on criteria like academic reputation, cost, and student satisfaction.
3. *Sorting* (Classification Problematic): To assign alternatives into predefined categories or groups based on their performance. Example: Sorting job applicants into categories like "highly qualified," "qualified," and "not qualified" based on their skills and experience.

In MCDM, predefined categories in sorting are assumed to be *preferentially ordered*, i.e., some categories are assumed better or more preferred than others. Sorting is similar to *classification*, where categories are not necessarily ordered. Example: Classify medical images into diagnostic categories, such as "no findings", "pneumonia", "edema", or "tumor".

In this report, we focus on a multi-criteria modelling method *DEX* (Decision EXpert). DEX (Bohanec, 2022) is a decision-modeling method that combines multi-criteria models with some elements of expert systems. The essential characteristics of DEX are:

- DEX is *hierarchical*: A DEX model consists of hierarchically structured variables, called attributes;
- DEX is *qualitative*: All attributes in a DEX model are symbolic, taking values that are generally words, such as "bad", "medium", "excellent", "low";
- DEX is *rule-based*: Decision alternatives are evaluated according to decision rules, acquired from the decision maker and represented in the form of decision tables.

DEX is primarily a *sorting* or *classification* MCDM method: the evaluation process assigns each alternative to some distinct final evaluation *class*. This is because all components of a DEX model are qualitative: attributes are qualitative and the evaluation process (aggregation of values) is determined by qualitative decision rules (see examples in Figure 1). Furthermore, decision alternatives are described by qualitative input values and all evaluation results are qualitative, too (Figure 2).

In practice, however, it is often necessary to simultaneously address other tasks than just sorting, i.e., choosing and/or ranking. For example, when there are several alternatives assigned to the same evaluation category, we may still want to distinguish between them and choose the best one. Actually,

this can be done in DEX, but may require additional steps and additional effort by the decision maker, for instance by exploring full evaluation results or using decision-analytic tools such as *selective explanation* and *deep comparison* (Bohanec, 2024).

Nevertheless, it is still very tempting to think about a method that would *rank alternatives* using DEX models and would require very little additional effort from the decision maker. In this report we show that this is possible, provided that we accept some assumptions about preferential ordering of decision rules and their numerical interpretation. We present two novel methods for ranking of alternatives using DEX models, called QQ2 and QL, which were designed as an improvement over an older method (Bohanec, et al., 1992), which eventually became known as QQ (Qualitative-Quantitative). We experimentally evaluate QQ2 and QL on a large collection of DEX decision tables and compare it with QQ.

In what follows, we first present related work about MCDM sorting in general and specifically related to DEX. In section 3, we define the concepts and components of DEX models, and introduce formal notation. The aims and goals of alternatives' ranking are defined in section 4. There, we gradually introduce requirements for such a DEX method, providing grounds for actually proposing QQ2 in section 5. In section 6, we present QQ and QL. QQ is the predecessor of QQ2, and QL is similar to QQ2, but uses a linear instead of quadratic optimization model. Results of extensive testing of QQ, QL and QQ2 on 3322 DEX models are presented in section 7. Section 8 discusses some additional extensions needed for a practical implementation of QQ2 (section 8.1), and presents an actual implementation of QQ2 in software called DEXiWin (section 8.2). Section 9 concludes the report.

2 Related Work

Sorting is an important part of MCDM and there is a lot of related literature. However, one should understand that the majority of MCDM methods are of numerical nature: they use numerical attributes, accept numerical input data about alternatives and yield numerical evaluations. Accordingly, they are in some "natural" way more suitable for choosing and ranking than qualitative methods such as DEX. Therefore, it is not surprising that many well-known ranking MCDM methods have their "sorting" variations, which specifically address the sorting problematic. In their textbook, López, et al. (2023) identify and present a number of sorting methods, many as variations of their ranking counterparts. They distinguish between four types of sorting methods:

1. *Full aggregation approach*: Methods where a score is evaluated for each criterion and then synthesized into a global score. A multi-criteria model is explicitly developed to evaluate alternatives. Examples: methods UTADIS (UTilités Additives DIScriminantes), UTADIS GMS, AHPsort as a variation of AHP (Analytic Hierarchy Process).
2. *Outranking approach*: Methods where alternatives are compared between each other (for instance, by pairwise comparison). Preferences and rankings are gradually determined without developing an explicit model. Examples: method PROMSORT as a variation of PROMETHEE (Preference Ranking Organization METHod for Enrichment Evaluations), ELECTREE III and ELECTRE TRI as a variation of ELECTRE (ELimination and Choice TRanslating REality).
3. *Goal, aspiration, or reference-level*: Methods based on providing a reference level (goal) of each criterion and identifying alternatives closest to the goal. Examples: DEA (Data Envelopment Analysis), TOPSIS-SORT as a variation of TOPSIS (Technique for Order of Preference by Similarity to Ideal Solution).
4. *Nonclassical MCDM approaches*: Methods employing decision rules. Example: DRSA (Dominance-Based Rough Set Approach).

While DEX is not mentioned in López, et al. (2023), it belongs to both (1) the full aggregation approach, because it uses an explicit hierarchical model to evaluate alternatives, and (2) non-classical approach, because it uses decision rules and has some similar elements as DRSA.

In addition to an extensive scientific literature reviewed in López, et al. (2023), there are many other recent publications on sorting MCDM methods, which typically address some specific method or extension of an existing method. Kadziński, et al. (2014) propose using ROR (Robust Ordinal Regression) in the context of DRSA. Ru, et al. (2023) investigate using ROR methods with uncertain preferences. Liu, et al. (2020) present a preference learning framework for multiple criteria sorting, based on alternatives being assigned to multiple classes. Sorting extensions of TOPSIS are proposed by Ferreira de Lima Silva, et al. (2020) and Yatsalo, et al. (2024). Wang, et al. (2023) suggest a sorting extension to the method TODIM (Portuguese acronym for Interactive Multi-Criteria Decision Making).

While the issue with the majority of MCDM methods is how to approach sorting from the ranking starting point, the issue with DEX is just the opposite: how to rank alternatives based on qualitative information, suitable for sorting. Also, DEX is somewhat specific and DEX models contain elements different from those in general MCDM methods. Consequently, algorithms for ranking alternatives in DEX are specific and related literature is scarce.

In a very early attempt, Bohanec, et al. (1992) proposed a "combined qualitative and quantitative method", which is nowadays known as QQ (Bohanec, 2022). The authors explored the idea of a "parallel-but-consistent" evaluation of alternatives: in addition to a normal DEX qualitative evaluation, alternatives are additionally evaluated numerically. The two evaluations are kept consistent so that numeric evaluations do not breach the order established by qualitative evaluations. This is achieved by representing alternatives' values by a combination of a qualitative (symbolic) and a quantitative (numeric) value, where the former indicates a class and the latter a numeric offset within the class (see section 4.3 for further details). In QQ, aggregation functions are determined by a local linear approximation of decision rules. Although QQ was successfully used in practical applications, it had some weak points (discussed later in section 6.1). Also, QQ was only implemented in prototype research software and never made it to the popular DEX software called DEXi (Bohanec, 2020); this attributed to its lower exposure in the field. QQ is a direct predecessor of QQ2 and QL, the methods presented in this report and developed with the aim to improve on QQ's weaknesses.

Another attempt to DEX ranking was made by Mileva Boshkoska and Bohanec (2012). The basic approach was similar to QQ, except that copulas were used instead of linear approximations so as to get a better fit between individual decision rules and their numeric approximation. Another approach, called QUANQUAL, has been suggested by Brelih, et al. (2018). There, the authors combine qualitative evaluation of classes with numeric evaluation within classes; numerical evaluation is given in the form of numeric intervals. QUANQUAL's representation of evaluation values is considerably different from QQ's, and results are incomparable.

3 DEX Concepts and Notation

Formally, a DEX model M is defined as a four-tuple $M = (X, D, S, F)$, where X is the set of *attributes*, S is the descendant function that determines the hierarchical structure of attributes, D is the set of value scales of attributes, and F is the set of aggregation functions (Bohanec, 2022). In what follows, we formally define these components to the level necessary for the scope of this report.

3.1 Attributes

Attributes $X = \{x_1, x_2, \ldots, x_n\}$ are variables that represent observable properties of decision alternatives (inputs), and partial and overall results of evaluations (outputs). In DEX models, attributes

are usually given unique and meaningful names, such as *Price*, *Productivity*, etc. In such cases, the notation x_i is conveniently replaced by the corresponding attribute name.

To illustrate formal concepts, we shall use the DEX model called *CAR* (Figure 1). This is a simple model for evaluating personal cars, used for educational purposes, referred to in many publications and distributed with DEX software[1]. The model contains $n = 10$ attributes: $X = \{CAR, PRICE, BUY.PRICE, MAINT.PRICE, TECH.CHAR., COMFORT, \#PERS, \#DOORS, LUGGAGE, SAFETY\}$.

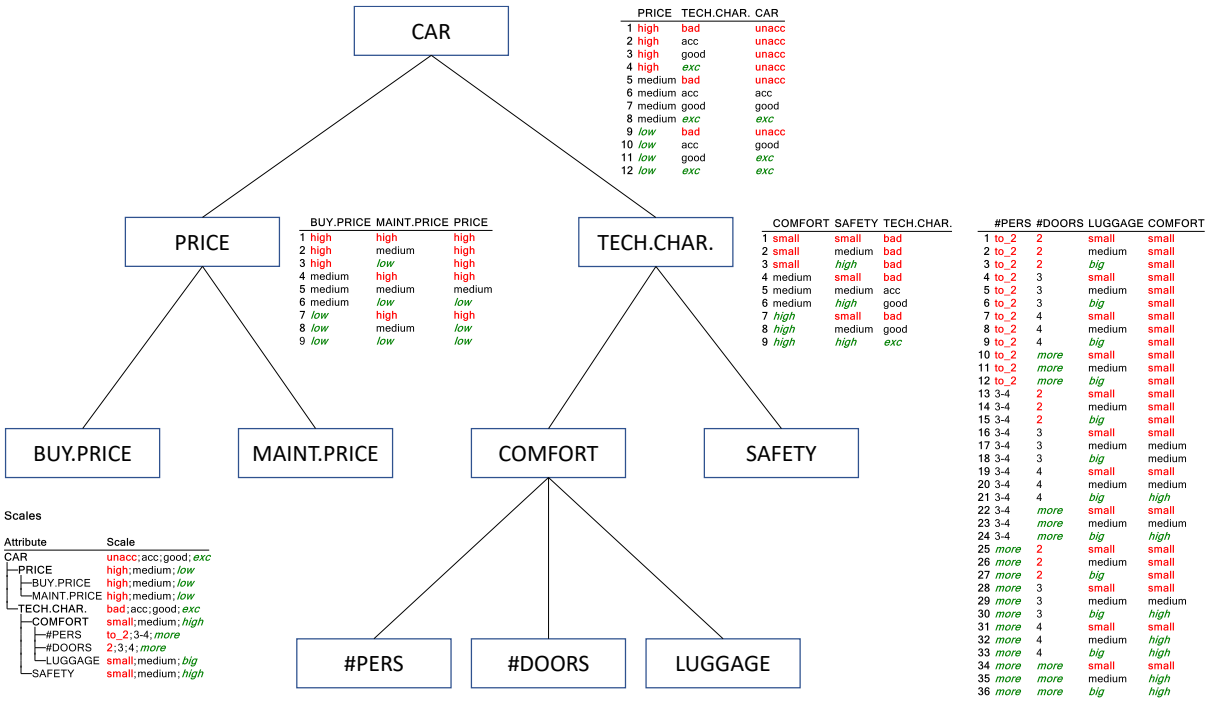


Figure 1: The structure and components of CAR*, a DEX model for evaluating personal cars.*

3.2 Model Structure

Attributes in a DEX model are structured *hierarchically*. The structure is defined by the function $S: X \rightarrow 2^X$, which associates each $x \in X$ with a set of its *descendants* $S(x)$ in the hierarchy. The function S is assumed to be defined so that it forms a *hierarchy*, i.e., a connected directed graph without cycles.

For *CAR* (Figure 1), S is defined as follows: $S(CAR) = \{PRICE, TECH.CHAR.\}$, $S(PRICE) = \{BUY.PRICE, MAINT.PRICE\}$, $S(COMFORT) = \{\#PERS, \#DOORS, LUGGAGE\}$, and $S(x) = \emptyset$ for $x \in \{BUY.PRICE, MAINT.PRICE, \#PERS, \#DOORS, LUGGAGE, SAFETY\}$.

Given S, the set of *parents* of each $x \in X$ is then defined as $P(x) = \{p \in X: x \in S(p)\}$. Attributes without parents are called *roots* and represent main *outputs* of the model. Attributes without descendants, $S(x) = \emptyset$, are called *basic* attributes and represent model *inputs*. Attributes with $S(x) \neq \emptyset$ are referred to as *aggregate* attributes, and are also considered (partial) *outputs* of the model. Normally, DEX models represent *trees*, where each attribute has one parent, except the root which has none. In contrast, a (full) *hierarchy* may contain attributes with multiple parents. DEX allows using hierarchies through the mechanism known as "attribute linking" (Bohanec, 2020; 2022; 2024).

[1] https://dex.ijs.si/

Considering basic and aggregate attributes, X can be partitioned in two distinct subsets:

- $Y = \{x \in X: S(x) \neq \emptyset\}$: the set of all aggregate (output) attributes.
- $Z = \{x \in X: S(x) = \emptyset\}$: the set of all basic (input) attributes.

3.3 Attribute Scales

Each attribute $x \in X$ is associated with a value *scale*, denoted $\text{scale}\, x \in D$, which is defined as an ordered set of symbolic (qualitative) values:

$$\text{scale}\, x = [v_{x,1}, v_{x,2}, \dots, v_{x,m_x}]\,.$$

Here, $m_x \geq 2$ denotes the number of discrete values that can be assigned to x. Usually, value scales are small and rarely consist of more than five values. Scale values are typically represented by words, for instance "low", "high", "unacceptable", "good".

DEX scales can be either *ordered* in an *ascending* or *descending* order, or *unordered*. Values of an *ascending* scale are assumed to be preferentially ordered so that $v_{x,1} \preccurlyeq v_{x,2} \preccurlyeq \cdots \preccurlyeq v_{x,m_x}$, where '$\preccurlyeq$' denotes a weak "worse-or-equal" preference relation. In *descending* scales, the "better-or-equal" relation '$\succcurlyeq$' applies instead. No preference relations can be assumed with *unordered* scales. We denote the order of scale x as

$$\text{order}\, x \in \{\text{unordered}, \text{ascending}, \text{descending}\}.$$

Conventionally, particularly **bad** and ***good*** scale values are printed in color, as shown in Figure 1. All scales in the *CAR* model are ascending. Even though the scale $\#DOORS = \{2,3,4,more\}$ contains number-like values, these numbers are understood as symbols and have no intrinsic numerical value.

However, since scales are ordered sets, we can associate an *ordinal number* with each element of the scale. For $\text{scale}\, x = [v_{x,1}, v_{x,2}, \dots, v_{x,m_x}]$, the corresponding ordinal numbers are $\text{ord}\, v_{x,1} = 1$, $\text{ord}\, v_{x,2} = 2$, ..., $\text{ord}\, v_{x,m_x} = m_x$.

Scales that correspond to some subset of attributes $K \subseteq X$ form a *decision space*, denoted

$$\text{space}\, K = \text{scale}\, x_1 \times \text{scale}\, x_2 \times \dots \times \text{scale}\, x_k \text{ for all } x_i \in K, i = 1,2, \dots, k, k = |K|.$$

3.4 Decision Tables

In order to evaluate decision alternatives, DEX uses aggregation functions $F = \{f_x, x \in Y\}$. Formally, each aggregate attribute $x \in Y$ is associated with a total function that maps:

$$f_x: \text{space}\, S(x) \to \text{scale}\, x.$$

In this context, all descendants of x are called function *arguments* and are denoted $\text{args}\, f_x = S(x)$.

Aggregation functions are represented by *decision tables*, see examples in Figure 1. A decision table T_x, associated with $x \in Y$, consists of r_x *elementary decision rules* that correspond to all possible combinations of values of function arguments. Thus, there are $|\text{space}\, S(x)|$ decision rules: the right-most column of a decision table represents the outcome of each decision rule (also called a *class)*. Further decision-table notation needed for the purpose of ranking is presented later in section 4.

While DEX in general allows different types of output values (intervals, value distributions) (Bohanec, 2022), we shall hereafter assume that each decision rule prescribes exactly one class. Solutions for tables not fulfilling this assumption are discussed later in section 8.1.

3.5 Alternatives

Alternatives (or *decision alternatives*) are objects, actions or other kinds of choices considered in the decision-making process. From the decision modeling viewpoint, $\mathcal{A} = \{A_1, A_2, \dots, A_q\}$ are data vectors processed by M. Each *alternative* $A_i, i = 1,2, \dots, q$, is represented by a vector of values:

$$A_i = [a_{x,i} \in \text{scale}\, x, \forall x \in X],$$

where each $a_{x,i}$ represents the value of A_i that is assigned to attribute x. The value vector can be partitioned to $Y(A_i)$ and $Z(A_i)$, i.e., value vectors corresponding to all aggregate and all basic attributes, respectively.

Given some alternative A, represented by an assignment of values that correspond to input attributes

$$\boldsymbol{z} = [a_1, a_2, \dots, a_b] \in \text{space}\, Z,$$

we can *evaluate* that alternative recursively for each attribute $x \in X$, starting with the root attribute x_1:

$$E_x(A) = \begin{cases} f_x(E_{x_{(1)}}(A), \dots, E_{x_{(k)}}(A)) & \Leftarrow \forall x_{(i)} \in S(x), x \in Y \\ z_x & \Leftarrow x \in Z \end{cases}.$$

Figure 2 shows six personal cars (decision alternatives) represented in terms of (column) vectors of qualitative input values ($Z(\mathcal{A})$, Figure 2, top), and full evaluation ($X(\mathcal{A})$, Figure 2, bottom). The cars are sorted in three categories corresponding to attribute *CAR*: *Car3* and *Car4* are "unacceptable", *Car2* is "good", and *Car1*, *Car5* and *Car6* are "excellent". Lower-level evaluations reveal the main reasons for such results. For instance, both "unacceptable" cars seem too expensive for buying. Still, the question remains: which of the three "excellent" is actually the best? Or, to put it differently: how are *Car1*, *Car5* and *Car6* ranked within the "excellent" class?

Alternatives

Attribute	Car1	Car2	Car3	Car4	Car5	Car6
BUY.PRICE	medium	medium	high	high	*low*	*low*
MAINT.PRICE	*low*	medium	medium	medium	medium	medium
#PERS	*more*	*more*	*more*	*more*	*more*	*more*
#DOORS	4	4	*more*	*more*	*more*	*more*
LUGGAGE	*big*	*big*	*big*	*big*	medium	*big*
SAFETY	*high*	medium	medium	*high*	*high*	*high*

Evaluation results

Attribute	Car1	Car2	Car3	Car4	Car5	Car6
CAR	*exc*	good	unacc	unacc	*exc*	*exc*
├─PRICE	*low*	medium	high	high	*low*	*low*
│ ├─BUY.PRICE	medium	medium	high	high	*low*	*low*
│ └─MAINT.PRICE	*low*	medium	medium	medium	medium	medium
└─TECH.CHAR.	*exc*	good	good	*exc*	*exc*	*exc*
├─COMFORT	*high*	*high*	*high*	*high*	*high*	*high*
│ ├─#PERS	*more*	*more*	*more*	*more*	*more*	*more*
│ ├─#DOORS	4	4	*more*	*more*	*more*	*more*
│ └─LUGGAGE	*big*	*big*	*big*	*big*	medium	*big*
└─SAFETY	*high*	medium	medium	*high*	*high*	*high*

Figure 2: Decision alternatives (personal cars): Input values (top) and evaluation results (bottom).

4 Ranking of Alternatives: Aims, Goals and Requirements

In this section, we gradually develop the concepts, notation and requirements for the new ranking methods.

4.1 Decision Tables: Concepts and Notation

First, let us introduce some concepts and notation, using the decision table T_{CAR} in Figure 3 as an example. The table is associated with attribute *CAR* in Figure 1. Apart from the very first column on the left that displays decision rule indices, T_{CAR} is composed of three columns. The first two, *PRICE* and *TECH.CHAR.*, correspond to descendants of *CAR* in the model structure, which also serve the role of function arguments in this context. It is important to notice that all the possible combinations of values of *PRICE* and *TECH.CHAR.* are listed, giving 12 rows, each called an *elementary decision rule*. Each rule determines the output (*class*) value of *CAR* for that specific arguments.

	PRICE	TECH.CHAR.	CAR
1	high	bad	unacc
2	high	acc	unacc
3	high	good	unacc
4	high	*exc*	unacc
5	medium	bad	unacc
6	medium	acc	acc
7	medium	good	good
8	medium	*exc*	*exc*
9	*low*	bad	unacc
10	*low*	acc	good
11	*low*	good	*exc*
12	*low*	*exc*	*exc*

Figure 3: Decision table associated with the CAR *attribute (from Figure 1).*

In general terms, a decision table $T_x, x \in Y$, represents the mapping

T_x: space $S(x) \rightarrow$ scale x.

T_x consists of *elementary decision rules*

rules $T_x = \{r_{\boldsymbol{x}}, \forall \boldsymbol{x} \in \text{space } S(x)\}$

where each rule

$r_{\boldsymbol{x}}$: $\boldsymbol{x} \in$ space $S(x) \rightarrow c \in$ scale x.

Notice that according to $\forall \boldsymbol{x}$ above, the table is *complete*.

For notational convenience, $x(r) \in$ space $S(x)$ shall denote the conditional part of rule r, and $c(r) \in$ scale x the class assigned by that rule to x. The notation $r_i, i \in \mathbb{N}^+$, denotes the i-th rule in some given decision table.

We also make the following assumptions:

- scale $x = \{C_1, C_2, \dots, C_m\}$;
- all the involved scales (of x and $S(x)$) are preferentially *ordered* and *ascending*;
- the table is preferentially *consistent*, i.e., it obeys the *principle of dominance*: $\forall r, p \in$ rules T: $x(r) \preccurlyeq x(p) \Rightarrow c(r) \preccurlyeq c(p)$.

Here, the *dominance relation* $\boldsymbol{x}_1 \preccurlyeq \boldsymbol{x}_2$, where $\boldsymbol{x}_1, \boldsymbol{x}_2 \in$ space $S(x)$ are conditional parts of two rules, is defined by pairwise argument-by-argument comparison:

$\boldsymbol{x}_1 \preccurlyeq \boldsymbol{x}_2 \Leftrightarrow x_{1,i} \preccurlyeq x_{2,i}, i = 1,2, \dots, |S_x|$

When this relation holds, $\boldsymbol{x}_2$ is said to *dominate* $\boldsymbol{x}_1$, and $\boldsymbol{x}_1$ *is dominated by* $\boldsymbol{x}_2$.

For the purpose of ranking, we shall also add another column to T_x, consisting of a vector of numeric values, individually denoted $q(r) \in \mathbb{R}$. Figure 4 illustrates these notational conventions.

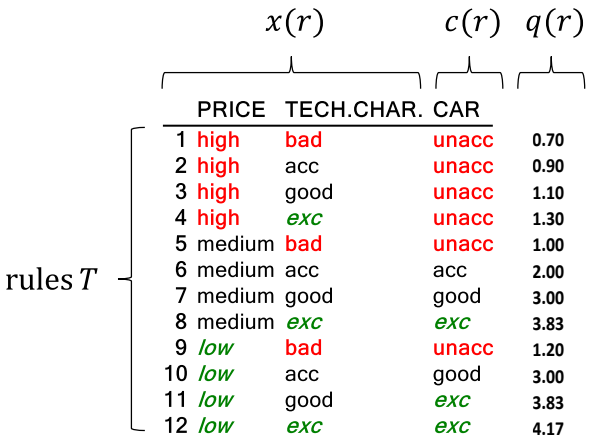


Figure 4: Notational conventions: A decision table with associated numeric values.

4.2 Using Preferential Dominance for Ranking

Now we can define the *ranking* task:

Given:

- A DEX model M,
- a set of alternatives $\mathcal{A} = \{A_1, A_2, \dots, A_q\}$,
- which are already fully evaluated by M so that $A_i = [a_{x,i} \in \text{scale}\, x, \forall x \in X], i = 1,2, \dots, q$.

Let $\mathcal{A}_x(C) \subseteq \mathcal{A}$ denote the subset of alternatives that were sorted to some class $C \in \text{scale}\, x$:

$\mathcal{A}_x(C) = \{A \in \mathcal{A}: E_x(A) = C\}, x \in Y$.

Accordingly, all alternatives in $\mathcal{A}_x(C)$ are evaluated by the same qualitative value C on x. When $\text{scale}\, x = [C_1, C_2, \dots, C_m]$, $\mathcal{A}$ is partitioned to distinct $\mathcal{A}_x(C_i), i = 1,2, \dots, m$.

Usually, x is assumed to be the root of M, but the approach applies equally well to any other aggregate attribute in the model.

Then, for some attribute $x \in Y$ and each of its values $C \in \text{scale}\, x$:

Rank alternatives: Establish an order of alternatives in $\mathcal{A}_x(C)$ with respect to relation '$\preccurlyeq$'.

In other words, the task is to rank alternatives *within* each evaluation class C. A total order is preferred, but partial order is also acceptable. The inter-class ranking established according to preferentially ordered qualitative values of x should be preserved.

Exact requirements for such rankings are defined later in section 4.3. But first let us see how to rank, at least partially, decision rules in a given decision table. Such ranking provides and important building block for ranking of alternatives at a global model level.

Consider decision table *CAR* in Figure 3. There, decision rules map input arguments to qualitative values (classes) of *CAR*: rules 1, 2, 3, 4, and 9 to "unacc", rule 6 to "acc", rules 7 and 10 to "good", and rules 8, 11, 12 to "exc". Consider rules 11 and 12. Even though that they both map to the same "exc" class, they still represent two preferentially different situations. Conditional parts of these rules are

$x(r_{11}) = [low, good]$, $x(r_{12}) = [low, exc]$.

A pairwise comparison of these vectors gives $low = low$ and $good \preccurlyeq exc$, which can be, according to the principle of dominance, generalized to $x(r_{11}) \preccurlyeq x(r_{12})$ and further to $r_{11} \preccurlyeq r_{12}$. Notice that some rules cannot be related in this way, for instance $x(r_4) = [high, exc]$ and $x(r_5) = [medium, bad]$, where pairwise comparison gives the opposite relations: $high \preccurlyeq medium$ and $exc \succcurlyeq bad$; consequently, the relation '$\preccurlyeq$' does not hold (in any direction) for r_4 and r_5. In general, decision rules can be therefore only *partially ranked* according to the principle of dominance.

Following these principles, the decision rules in Figure 3 can be ranked as follows:

For $C =$ "unacc": $r_1 \preccurlyeq r_2 \preccurlyeq r_3 \preccurlyeq r_4$ and $r_1 \preccurlyeq r_5 \preccurlyeq r_9$.

For $C =$ "acc": there is only r_6.

For $C =$ "good": there are only r_7 and r_{10}, unrelated.

For $C =$ "exc": $r_8 \preccurlyeq r_{12}$ and $r_{11} \preccurlyeq r_{12}$.

Why are these rankings important? When two alternatives A_a and A_b are evaluated to the same class C by two different rules of some decision table, say r_a and r_b, so that $r_a \preccurlyeq r_b$, we can assume that $A_a \preccurlyeq A_b$ in the context of that decision table. In other words: despite that the alternatives have been assigned to the same qualitative class C, we may still be able to rank them within C.

Instead of using the relation '$\preccurlyeq$' directly, we rather introduce the *numeric evaluation* of decision rules, which assigns some numeric value $q_r \in \mathbb{R}$ to each rule $r \in T$. Collectively, the vector of such assignments is denoted $q(T)$ and added as a new column to T. The difference between using relation '$\preccurlyeq$' directly on rules and relation '$\leq$' on $q(T)$ is that '$\leq$' defines a total rather than partial order. This introduces an implicit assumption that decision rules that do not preferentially dominate one another can still be compared. Notice that the $q(r)$ values in Figure 4 already reflect the partial rule orders identified above.

4.3 Requirements for the Ranking Method

Following the observations from the previous section, let us gradually define requirements for the ranking method and values $q(r)$ assigned to decision rules.

Requirement 1: **Automatic construction** $q(T)$ should be determined automatically from information already available in T; no additional input is required from the decision maker.

In principle, it is possible to imagine an approach that would enable users to fully define $q(T)$ on their own. In addition to formulating the column $c(r)$, this would require formulating another column $q(r)$. Knowing that the first task is already quite demanding in DEX, it is unlikely that decision makers would like to attempt the second one. Considering that decision tables already contain some preferential ordering information, it seems more feasible to use that information "for free", provided that we accept some assumptions, as detailed below.

Requirement 2: **Preserving dominance** $q(T)$ should preserve the preferential order of rules established according to the principle of dominance:

$$\forall r, p \in \text{rules}\, T : r \preccurlyeq p \Rightarrow q(r) \leq q(p).$$

Requirement 3: **Consistency of qualitative and numeric evaluations**

$$\forall r \in \text{rules}\, T : c = \text{ord}\, c(r) \Rightarrow q(r) \in [c - 0.5, c + 0.5].$$

This requirement is based on the idea already proposed with QQ (Bohanec, et al, 1992): to present evaluations in the form $C \pm \omega$, where C represents the qualitative value, and $\omega \in [-0.5, +0.5]$ is a numerical *offset* to that value. The offsets -0.5 and $+0.5$ are in the context of C interpreted as "particularly bad" and "particularly good", respectively.

Several notational variations are possible:

- Keeping C in its original qualitative form, i.e., using the notation $C \pm \omega$. Examples: *CAR*="exc"-0.3 would have been considered excellent, but worse than *CAR*="exc"$+0.2$.
- Using ordinal numbers in place of C, but explicitly displaying both components: the above examples are transformed to $4 - 0.3$ and $4 + 0.2$, considering that $\mathrm{ord}\, exc = 4$.
- Adding up the ordinal number and offset: 3.7 and 4.2. This gives a very compact notation which indicates both the ordinal number and the $[-0.5, +0.5]$ offset around it.

The only drawback of the latter notation arises when dealing with numbers that fall exactly midway between two integers, such as 3.5. For those, we still need to explicitly indicate the class, or assure that all the offsets stay in the range $(-0.5, +0.5)$. Class information is not needed in decision tables that display both columns $c(T)$ and $q(T)$ side by side, such as in Figure 14.

The rationale for using values with offsets is because $c(T)$ is considered a primary evaluation, acquired from the decision maker while defining the table T. Then, $q(T)$ is secondary and should reflect $c(T)$. Conversion between $c(T)$ and $q(T)$ is easy, except for rare cases indicated above. $q(T)$ formulated in this way can be used for both ranking of rules or alternatives within some class and across different classes. The interpretation of negative and positive ω values is simple and easy to understand.

Requirement 4: **Compatibility of input and output evaluations in model hierarchy**

So far, we have stayed in the context of a single decision table. However, a DEX model is structured hierarchically and decision tables are stacked one above the other in the hierarchy. This puts very strong requirements on the form and characteristics of $q(T)$. Consider the situation in Figure 5, which shows the top two levels of the *CAR* model together with three corresponding decision tables.

According to the requirements formulated above, the top-level decision table is expected to yield output values that can be assigned to the *CAR* attribute. Those outputs consist of qualitative values $c(T) \in \mathrm{scale}\, CAR$ and numeric values $q(T) \in [\mathrm{ord}\, unac - 0.5, \mathrm{ord}\, exc + 0.5] = [0.5, 4.5]$. However, there are two decision tables below *CAR*, and they are also required to yield a pair of values, qualitative and numeric, within boundaries determined by each table context. Therefore, any evaluation or ranking algorithm on the *CAR* level should accept input values from lower levels in the form $c \pm \omega$. For instance, $q(T_{CAR})$ must be able to accept the inputs *PRICE*=$high - 0.5$ and *TECH.CHAR.*=$bad - 0.5$. These values would trigger the *CAR* rule $r_1 = [high, bad] \rightarrow unacc$, but would also require the evaluation of $q(high - 0.5, bad - 0.5) = q(1 - 0.5, 1 - 0.5) = q(0.5, 0.5)$, which, according to Requirement 3, has to stay in the interval $[unacc - 0.5, unacc + 0.5]$.

This requirement effectively replaces our previous assumption that q maps decision rules to $\mathbb{R}$. In fact, we need a numeric evaluation function

$$Q_T: \mathbb{R}^k \rightarrow \mathbb{R},$$

where k is the number of arguments of T, and Q_T fulfils the requirements and boundaries identified above.

An important side effect of this requirement is that values of alternatives that correspond to basic attributes, can also be generalized to using the $c \pm \omega$ values. For instance, *SAFETY* is a basic attribute

in Figure 1. Instead of using just the qualitative values “small”, “medium” and “high”, as in regular DEX, we can now modify them by offsets, such as *SAFETY*=“medium”+0.1. The function $Q_{TECH.CHAR.}$ is able to accept such input values.

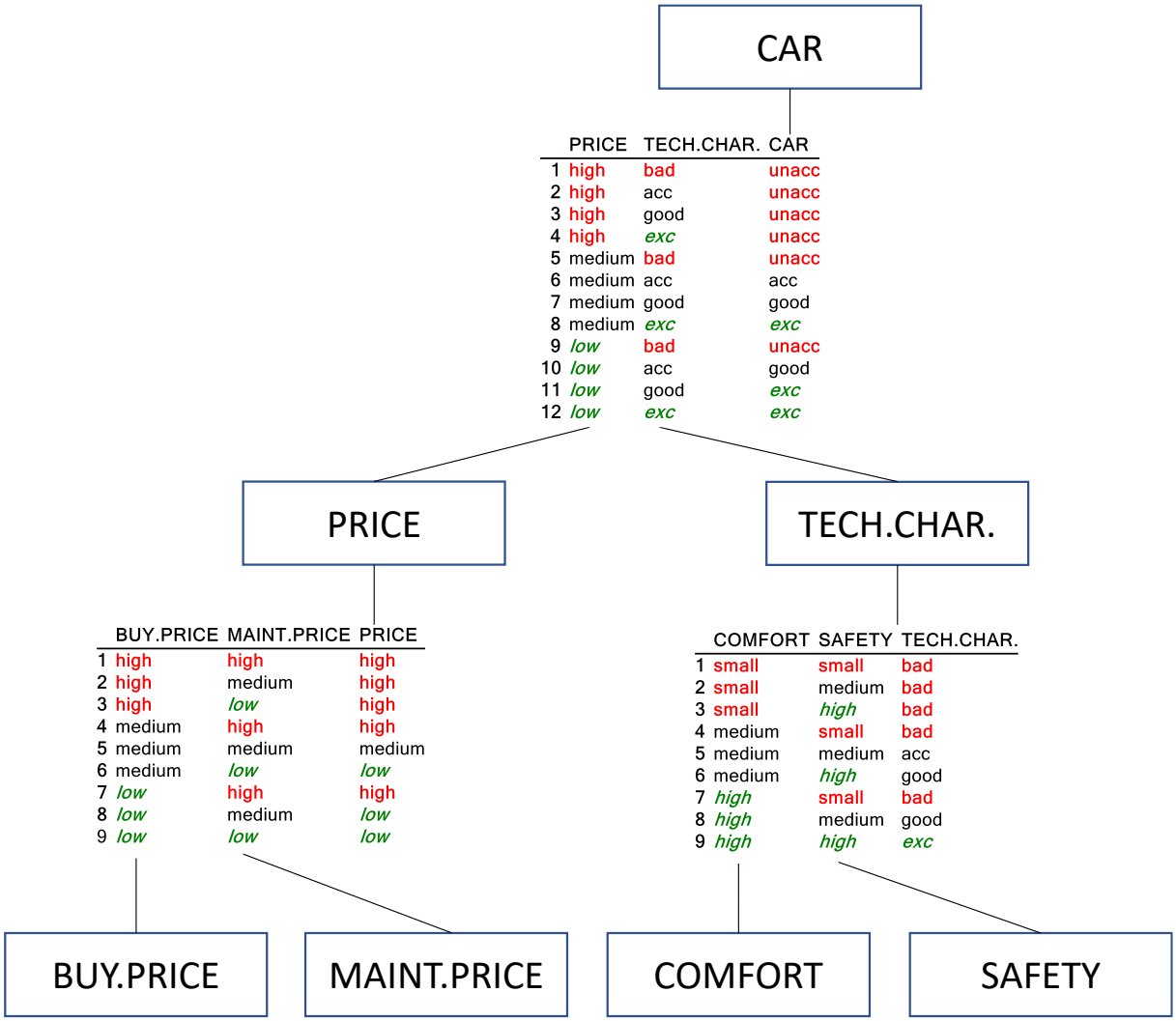


Figure 5: Hierarchical composition of decision tables in the CAR *model.*

With the requirements formulated so far, there are still plenty of possibilities of how to define the function Q_T. The following requirement adds an assumption, which might not be (entirely) in line with decision-maker’s preferences and expectations, but allows to formulate the construction of Q_T as an optimization problem that generates unique solutions.

Requirement 5: Separate decision rules as much as possible for better discrimination between alternatives

First, this means that the smallest and largest Q_{T_x} value for each class $C \in \text{scale}\, x$ should be in the range $[c-0.5, c+0.5]$. For instance, the smallest value of $Q \equiv Q_{T_{CAR}}$ for *CAR* class “unacc” is

from rule 1: $Q(high-0.5, bad-0.5) = Q(1-0.5, 1-0.5) = Q(0.5, 0.5) = 0.5.$

The largest value for class “exc” is then

from rule 12: $Q(low+0.5, exc+0.5) = Q(3+0.5, 4+0.5) = Q(3.5, 4.5) = 4.5.$

Second, we need to introduce some distance measure $d(r,p)$ between all pairs of rules r,p that map to some C. And third, we need to maximize the cumulative distances between pairs of preferentially related rules. The following section elaborates on these points.

5 Ranking Algorithm QQ2

Inputs:

- Decision table $T_x, x \in Y$
- Class $C \in \text{scale}\, x$
- Rule subset $R_C = \{r \in \text{rules}\, T_x : c(r) = C\}$

Notation:

- Indices of rule in R_C: $1,2,\dots,n$
- Rule conditions from R_C: $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_n\}$
- $q(T_x)$ evaluations, corresponding to individual rules: $q_1, q_2, \dots, q_n \in \mathbb{R}$

Output:

$q_1, q_2, \dots, q_n$ determined according to Requirements 1–5.

Let us also define:

- Adjacent rule pairs: $L = \{(i,j): \boldsymbol{x}_i \preccurlyeq \boldsymbol{x}_j, i,j \in 1,2,\dots,n, j > i, \nexists \boldsymbol{x}_k: \boldsymbol{x}_i \preccurlyeq \boldsymbol{x}_k \preccurlyeq \boldsymbol{x}_j\}$;
- Neighbors of rule i: $N_i = \{j: (i,j) \in L \vee (j,i) \in L\}$.

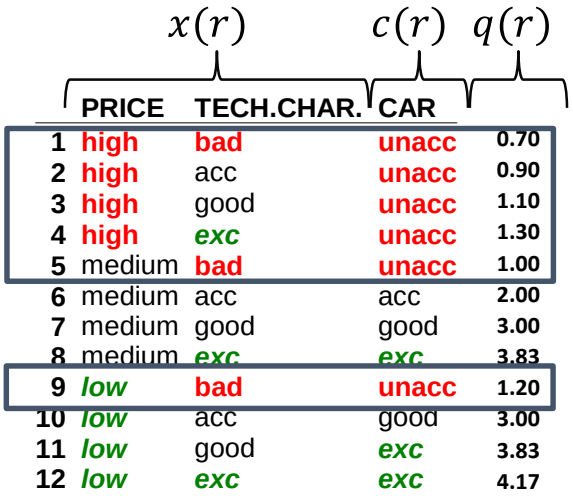


Figure 6: Determining $q(T_{CAR})$ for $c =$ "unacc".

An example situation is shown in Figure 6: we want to determine $q(r)$ for all $r \in R_{unacc}$. The rules that map to *CAR*="unacc" are in Figure 6 indexed 1, 2, 3, 4, 5, and 9, but conveniently renumbered to 1–6 in this example, $n = 6$. Considering the principle of dominance, the six rules form the lattice shown in Figure 7. This is just a different representation of relations already identified in section 4.2. For this example, $L = \{(1,2),(2,3),(3,4),(1,5),(5,6))\}$ and $N_2 = \{1,3\}$.

In order to prepare the lattice for optimization, two elements are added, $q_L = 0$ and $q_U = u$, which determine the lower and upper bounds, respectively, of $q_1, q_2, \dots, q_n$. The value u depends on the algorithm, but should be large enough to assure that we can constrain the separation between any $q_i, q_j, (i,j) \in L$ to $q_j - q_i \geq 1$. A good value for u is the length of the longest path in the lattice.

Having set the lattice (Figure 7), it is now clear what to do: place $q_1, \ldots, q_n$ evenly between q_L and q_U, considering all lattice branches.

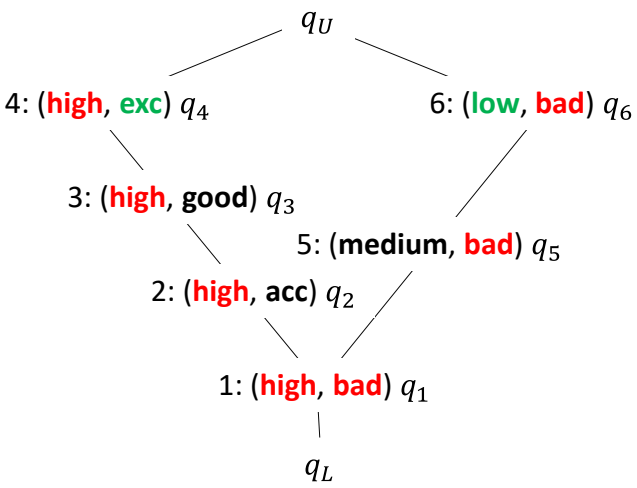


Figure 7: Partial order of CAR *decision rules that map to* $C =$ *"unacc".*

In the context of the lattice, we consider q_L and q_U regular elements and include them in N_i if they are located near element q_i. Therefore, $N_1 = \{2, 5, L\}$, $N_4 = \{3, U\}$, and $N_6 = \{5, U\}$.

On this basis, we can formulate the task as a quadratic optimization problem with constraints: minimize the sum of squares of distances between pairs of q's in L:

Variables: $q_0 = q_L, q_1, q_2, \ldots, q_n, q_{n+1} = q_U$

Minimize $\sum_{(i,j)\in L}(q_j - q_i)^2 = \sum_{i=0}^{n+1}(|N_i| {q_i}^2 - 2\sum_{j=0}^{n+1} q_i q_j)$

with respect to constraints:

$$q_0 = 0$$

$$q_j - q_i \geq 1 \text{ for } \forall (i,j) \in L$$

The value of $q_U = q_{n+1}$ is determined as part of the solution and need not be specified in advance.

After solutions $q_1, q_2, \ldots, q_n, q_{n+1}$ have been obtained, they have to be scaled so as to satisfy Requirements 4 and 5. This is done by placing a ± 0.5 rectangle around each rule point (Figure 8), and assuring, for each class $C \in$ scale x, that the lowest and highest corner of the corresponding surfaces are at $c - 0.5$ and $c + 0.5$, respectively. When the distances between adjacent rules are greater than the minimum distance for that class, the corresponding rectangles can be additionally slanted to increase separation between alternatives; such an example is located in Figure 8 around the point *PRICE*="medium" and *TECH.CHAR.*="bad".

For the *CAR* example, this algorithm, applied to all classes, gives results shown in the $q(r)$ column of Figure 6. The function Q_T is defined as a collection of hyperplanes constructed around the ± 0.5 neighborhood of each rule, as illustrated in Figure 8.

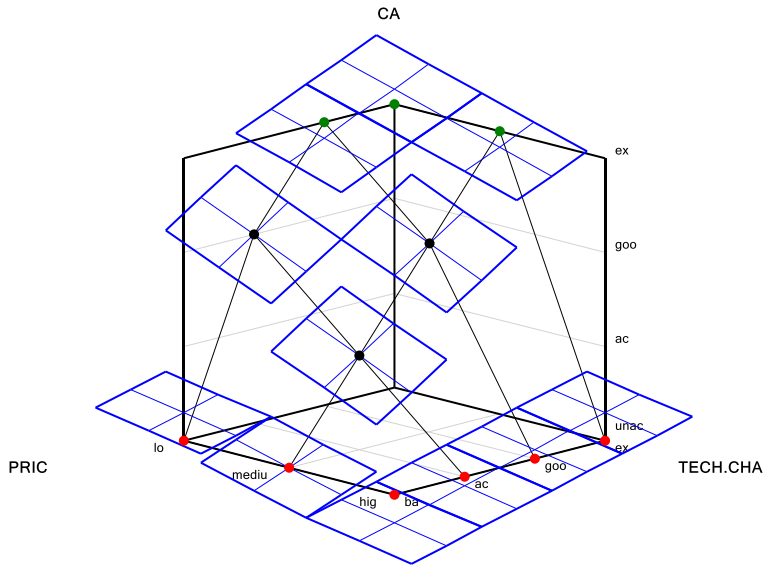


Figure 8: Algorithm QQ2: Graphical representation of $Q_{T_{CAR}}$.

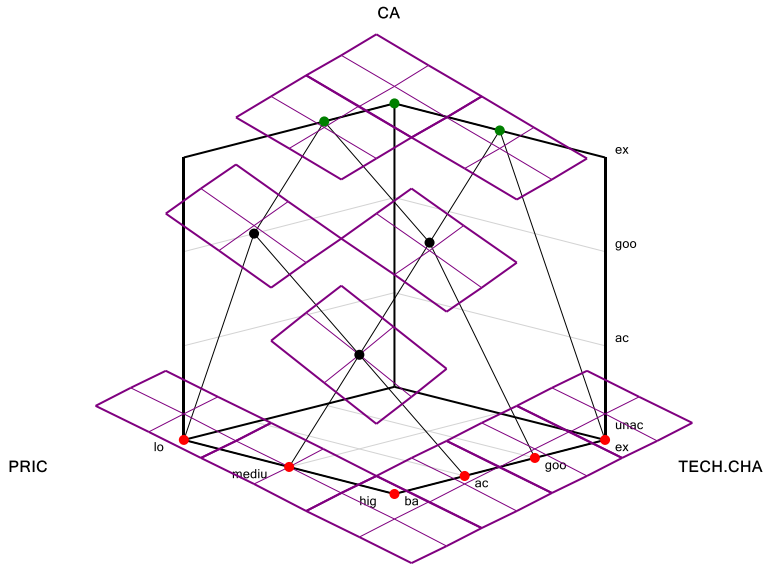


Figure 9: Algorithm QQ: Graphical representation of $Q_{T_{CAR}}$.

6 Other Algorithms

In order to experimentally evaluate QQ2, we used two other algorithms for comparison: the original QQ algorithm and a variation of QQ2 using a linear rather than a quadratic optimization model.

6.1 Algorithm QQ

Algorithm QQ has been proposed by Bohanec, et al. (1992) as the first DEX approach to ranking of decision alternatives. QQ constructs similar hyperplanes as QQ2 (Figure 9), but in a different way. It considers decision rules as points in a multi-dimensional space. First, it constructs a hyperplane that best, in the least-square sense, approximates all points. The slope of this hyperplane determines weights (importances, priorities) of arguments $S(x)$. Second, the hyperplane is partitioned for each class $C \in \text{scale}\, x$, so that the attribute weights are retained, while assuring the bounds $c - 0.5$ and $c + 0.5$ in the same way as in QQ2.

Consequently, all QQ hyperplane partitions are slanted in the same way; in Figure 9, the weight ratio between *PRICE* and *TECH.CHAR.* is 60:40, which determines the slopes of all hyperplanes in each direction.

QQ is very simple and convenient to implement, but has three drawbacks that eventually motivated the development of QQ2:

- *QQ relies on weights*. Using weights assumes that decision tables, interpreted as points in a multi-dimensional space, can be approximated well by linear functions. This may or may not hold for a given decision table. The ranking of alternatives is expected to work for all kinds of decision tables, and QQ is not suitable for cases other than linear or close-to-linear. In comparison, QQ2 makes no assumptions about the linearity; it considers only preferential ordering of all involved attributes.
- *QQ assumes equal weights for all hyperplane partitions*. This leads to sub-optimal separation of rules. In contrast, QQ2 adapts the orientation of hyperplanes to contexts determined by adjacent rules.
- *QQ does not distinguish between ordered and unordered attributes*, and may thus give wrong results for unordered attributes. In comparison, QQ2 correctly handles unordered attributes, as shown later in section 8.1.

Figure 10 combines Figure 8 and Figure 9, comparing the results of QQ and QQ2. The differences seem small, but are essential. For all classes it is clear that QQ2 covers the vertical dimension (which corresponds to attribute *CAR*) better than QQ. This is well illustrated with the two hyperplanes that correspond to *CAR*="good". There, QQ's hyperplanes are narrower than QQ2's in the vertical direction. This is because QQ's are constrained with the 60:40 weight ratio, while QQ2's successfully adapts to the absence of adjacent rules that may have constrained the position of rectangles. Ultimately, this leads to a better separation of ranked alternatives.

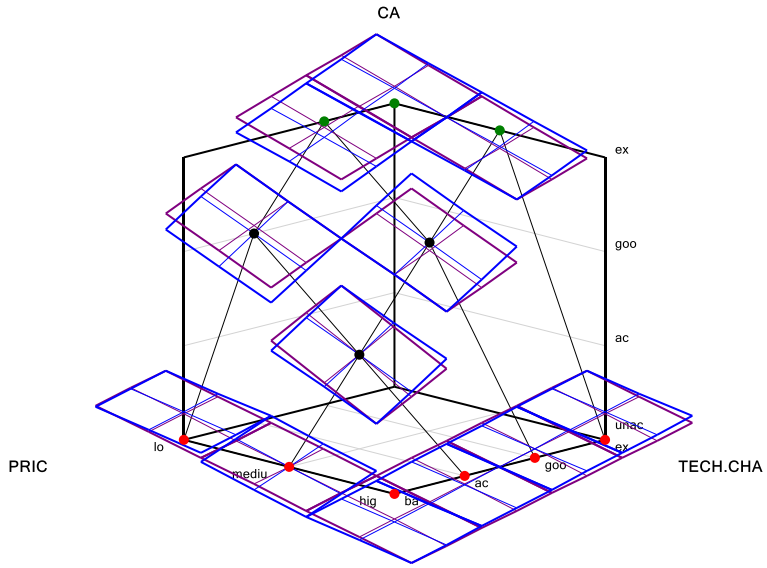


Figure 10: Comparison of QQ (purple) and QQ2 (blue).

6.2 Algorithm QL: Linear Optimization

In section 5, we did not explain why we chose quadratic optimization instead of linear, which is in principle simpler and may require less computation. The answer is that the quadratic model actually turned out simpler and more stable, while run times were comparable. Anyway, we did develop the

linear optimization model QL and compare it with QQ2. The main issue with QL is that, in addition to q variables, we need to introduce additional variables $d_1, d_2, \ldots, d_n$, which represent average distances between some q_i and its neighbors N_i. This increases the size of the optimization problem, as the d values need to be determined in the process, too.

QL is formulated as a minimization of the sum of d values:

Variables: $q_L, q_1, q_2, \ldots, q_n, q_U, d_1, d_2, \ldots, d_n$

Minimize $\sum_{i=1}^{n} d_i$

with respect to constraints:

$$q_0 = 0$$

$$q_u = \text{maxlength}\, L - 1$$

$$q_j - q_i \geq 1 \text{ for } \forall (i,j) \in L$$

$$d_i \geq D_i \text{ for } i = 1, \ldots, n \text{ where } D_i = \left| q_i - \frac{1}{|N_i|} \sum_{j \in N_i} q_j \right|$$

There are two issues with this formulation. First, we need to explicitly specify the upper bound q_U in terms of the maximum path length in L. Otherwise, when the bound is too small, no solutions are found, and solutions are unstable if the bound is too large. Second, the distance D_i requires calculation of an absolute value. This cannot be done directly in a linear program solver. Instead, we have to calculate D_i without using the absolute value and formulate two constraints instead of one for each i: $d_i \geq D_i$ and $d_i \geq -D_i$.

7 Experimental Evaluation

We implemented the three algorithms QQ, QL, and QQ2 in R, using standard R optimization packages lpSolve, lpSolveAPI, and quadprog. We experimentally evaluated them on a set of 3322 DEX decision tables, which were extracted from the database of real-life DEX models (Bohanec, 2017). For each decision table, we observed:

- *Separation*: average distance between dominated adjacent points, defined as $\frac{1}{|L|} \sum_{(i,j) \in L} |q_j - q_i|$;
- *Gaps:* average height of the ± 0.5 hyperplane rectangles, measured as a maximum vertical difference between their corners;
- *Time:* average execution time per decision table.

Table 1: Results of experimental evaluation of QQ, QL, and QQ2.

Measure	QQ	QL	QQ2	Significance
Separation	0.1737442	0.2121548	**0.2122702**	QQ2 > QL > QQ
Gaps	0.2116411	**0.2945648**	0.2943672	QL > QQ2 > QQ
Time [ms]	197.95	12.43	8.34	

We run the algorithms in R Studio on a desktop computer Intel(R) Core(TM) i9-10900K CPU @ 3.70GHz. Results (Table 1) indicate that QQ has been outperformed by QL and QQ2 with respect to both *Separation* and *Gaps*; the differences are statistically significant. Differences between QL and QQ2 are small by absolute values, but are also statistically significant: QQ2 has better separation than QL, and QL is better in terms of gaps. Considering execution times, QQ turned out more than ten times slower than the other two algorithms. This can be explained by QQ being implemented entirely in the high-level R language, while QL and QQ2 used highly optimized machine-level software packages. If QQ had

been implemented in the same way, it would likely run just as fast. Among QL and QQ2, the latter is faster. Anyway, all the algorithms are fast and run in the range of milliseconds per decision table, which is satisfactory for practical applications.

In addition to the *en-masse* evaluation, we also looked at individual Q mappings produced by the algorithms. Typical results achieved on some larger decision tables are shown in Figure 11, where the principle of dominance is obeyed throughout, and Figure 12, where some decision rules violate this principle, resulting in hyperplanes being scattered among the classes.

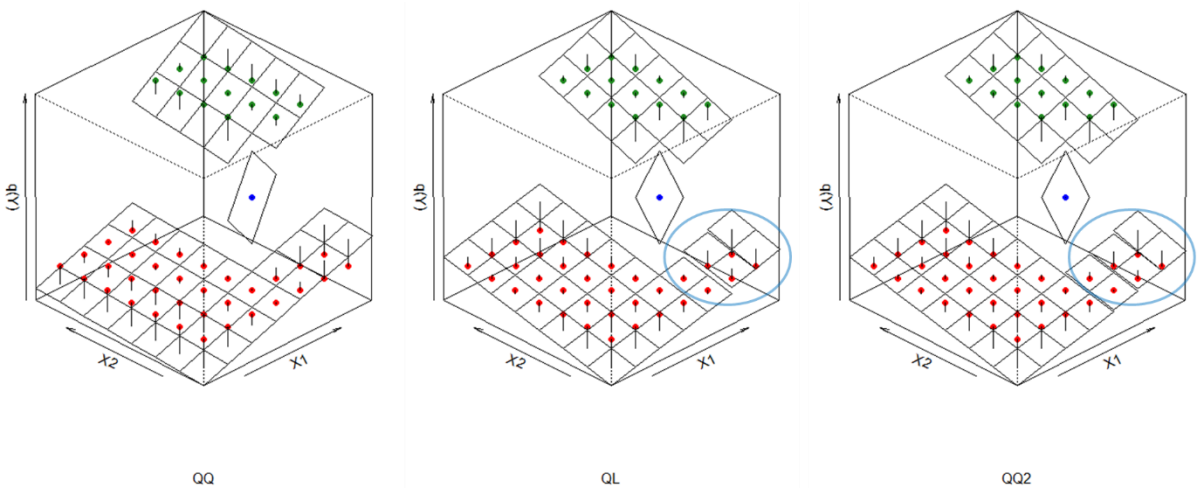


Figure 11: Comparison of QQ, QL and QQ2 on an aggregation function with two arguments and three classes.

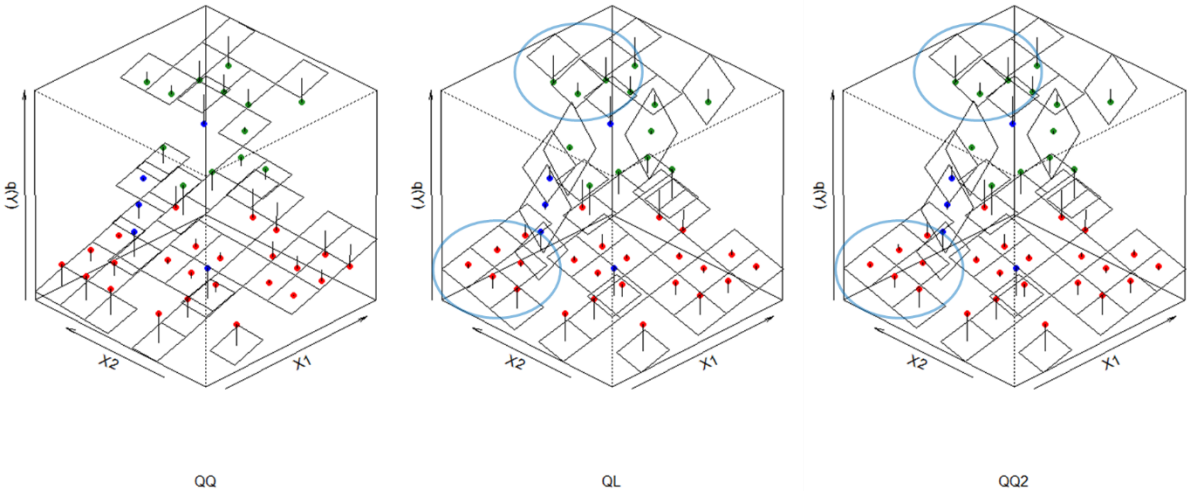


Figure 12: Comparison of QQ, QL and QQ2 on a aggregation function with two arguments and three classes.

The following can be observed from these figures:

- All QQ's slopes are indeed drawn according to the constant $weight(X_2)$:$weight(X_1)$ ratio and form connected per-class hyperplanes whenever possible.
- Consequently, this limits QQ to achieve the full vertical ranges available for each class. For example, the highest point on the very left part of the "red" hyperplane in Figure 11, is much lower with QQ than with QL and QQ2.
- Considering the coverage of the vertical space, QL and QQ2 perform much better than QQ, which is in line with the results shown in Table 1.

Results of QL and QQ2 are very similar. On the one hand, large parts of the corresponding Q's are equal. This happens when q_i points lie on the longest path between q_L and q_U. Consequently, there

are no degrees of freedom to move q_i around the lattice, and the corresponding rectangles are fixed, forming a nicely increasing slope, which is fully connected in Figure 11 and partly disconnected in Figure 12. On the other hand, there are small differences between QL and QQ2, marked with blue ovals in the two figures; they are due to using different (linear versus quadratic) optimization measures.

8 Implementation

8.1 Implementation Considerations for Practice

In section 4.1, we made three assumptions about DEX decision tables:

- they are complete,
- each decision rule maps to exactly one class, and
- all the involved attributes have ascending scales.

While DEX decision tables are always complete, the latter two assumptions are not generally true. Any realistic implementation must address them properly. Here is how we can handle those cases.

Mapping to multiple classes. In general, a DEX decision rule is allowed to map to multiple classes instead of just one:

$$r_{\boldsymbol{x}}\colon \boldsymbol{x} \in \text{space}\, S(x) \to \{C, C \in \text{scale}\, x\}$$

This does not pose a difficult problem, because all the considered algorithms work on a per-class basis. When considering some class C, the algorithms traverse through all rules that map to that class. A rule that maps to multiple classes is just considered several times, once per each class. Figure 13 shows an example where the rule [*BUY.PRICE*="low", *MAINT.PRICE*="medium"] maps to two classes: "medium" and "low". Separate hyperplanes are generated for each class, not disturbing each other.

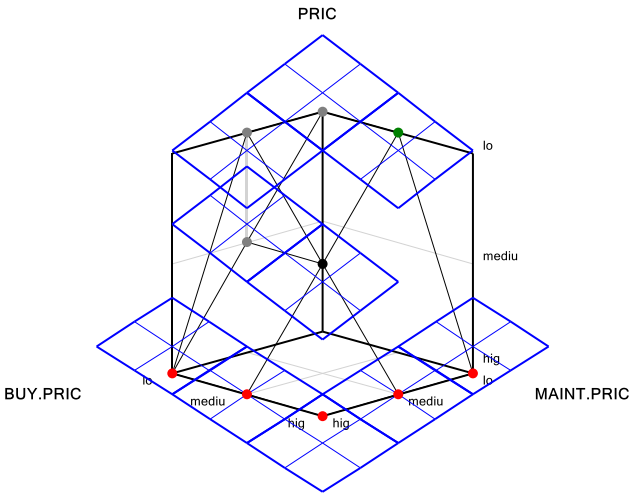


Figure 13: Example of a decision rule mapping to multiple classes.

Evaluating alternatives on this basis globally in the model is more complicated, though. An alternative evaluated to multiple classes receives multiple assignments, where each assignment is of the $c \pm \omega$ type. In the above example, some alternative may be evaluated as *PRICE*= {"medium"+0.17, "low"–0.17}.

Considering descending attribute scales. Any descending attribute scale can be turned to an ascending one by reversing the order of scale values. Although explicitly accounting for descending scales

complicates the mathematical treatment, it is merely a minor inconvenience that demands attention in a practical implementation.

Considering unordered attribute scales. This is a more difficult issue. When the scale of the *class variable* is unordered, then the whole concept of ranking becomes meaningless: we cannot really rank alternatives into unordered classes. One might argue that values $c - 0.5$ and $c + 0.5$ can still be interpreted as "bad" and "good", respectively, in the context of C. But then, how can we compare the numeric values 1.2 and 3.2 when the classes 1 and 3 are just labels, not numbers that can be compared? Consequently, ranking should not be applied with unordered classes.

However, it is still possible to rank alternatives when decision table arguments involve a combination of ordered and unordered attributes. In this case, we may partition the whole decision table into smaller sub-tables that depend only on ordered attributes; we construct such a sub-table for each combination of values that correspond to unordered attributes. For instance, if the original table has three attributes, two ordered and one unordered with three values, then we construct three sub-tables, one for each value of the unordered attribute. Then, we solve the ranking problem separately for each sub-table.

Or alternatively, we may reformulate the principle of dominance (section 4.1) and redefine the relation '$\preccurlyeq$' so that it disregards unordered arguments:

$$\boldsymbol{x}_1 \preccurlyeq \boldsymbol{x}_2 \Leftrightarrow x_{1,a} \preccurlyeq x_{2,a}, \forall a \in S(x): \text{order}\, a \neq \text{unordered}.$$

When all arguments are unordered, then ranking is again impossible, and none of the considered algorithms can handle this situation. Actually, QQ might generate some solution (provided that it can calculate attribute weights from points scattered all over the unordered space), but the result would likely be inappropriate and useless.

8.2 Current Implementation

While ranking has been largely left out from the mainstream DEX software for many years, it has been recently implemented in the software DEXiWin[2] (Bohanec, 2024). There are three functionalities available to the user:

1. *Calculating and presenting q-values for each decision table*. Figure 14 shows an example where the class column (*CAR*) is accompanied with QQ (called QQ1 in DEXiWin) and QQ2 offsets. The column *Linear* shows the results of linear approximation in each rule point; these are used to determine attribute weights in QQ, but are otherwise not relevant in the scope of this report.
2. *Drawing charts of q-values for both QQ and QQ2.* Examples of these are shown in Figures 8, 9, 10 and 13.
3. *Evaluating alternatives using QQ2:* Figure 15 shows the results of such evaluation of the six cars, qualitatively evaluated in Figure 2. While qualitative evaluations are exactly the same as before, the cars are additionally ranked within these classes. Among the excellent cars 1, 5, and 6, the latter appears the best, evaluated with the offset $+0.14$. Among the two bad cars, numbered 3 and 4, Car4 appears better with the offset $+0.32$. Notice that QQ2 evaluations also appear at the lower model levels (*PRICE*, *TECH.CHAR.* and *COMFORT*), providing additional information about the rankings achieved at those levels.

[2] https://dex.ijs.si/dexisuite/dexiwin.html

	PRICE	TECH.CHAR.	CAR	Linear	QQ1	QQ2
1	high	bad	unacc	-0,8333	-0,2727	-0,3000
2	high	acc	unacc	-0,1667	-0,0909	-0,1000
3	high	good	unacc	0,5000	0,0909	0,1000
4	high	*exc*	unacc	1,1667	0,2727	0,3000
5	medium	bad	unacc	0,1667	0,0000	0,0000
6	medium	acc	acc	0,8333	1,0000	1,0000
7	medium	good	good	1,5000	1,9167	2,0000
8	medium	*exc*	*exc*	2,1667	2,8125	2,8333
9	*low*	bad	unacc	1,1667	0,2727	0,3000
10	*low*	acc	good	1,8333	2,0833	2,0000
11	*low*	good	*exc*	2,5000	2,9375	2,8333
12	*low*	*exc*	*exc*	3,1667	3,1875	3,1667

Figure 14: Elementary decision rules with the corresponding value offsets.

Attribute	Car1	Car2	Car3	Car4	Car5	Car6
CAR	*exc*+0.11	good	unacc+0.12	unacc+0.32	*exc*+0.11	*exc*+0.14
├─PRICE	*low*−0.17	medium	high	high	*low*−0.17	*low*−0.17
│ ├─BUY.PRICE	medium	medium	high	high	*low*	*low*
│ └─MAINT.PRICE	*low*	medium	medium	medium	medium	medium
└─TECH.CHAR.	*exc*	good	good+0.10	*exc*+0.10	*exc*	*exc*+0.10
├─COMFORT	*high*	*high*	*high*+0.20	*high*+0.20	*high*	*high*+0.20
│ ├─#PERS	*more*	*more*	*more*	*more*	*more*	*more*
│ ├─#DOORS	4	4	*more*	*more*	*more*	*more*
│ └─LUGGAGE	*big*	*big*	*big*	*big*	medium	*big*
└─SAFETY	*high*	medium	medium	*high*	*high*	*high*

Figure 15: Six cars evaluated in DEXiWin using QQ2.

9 Conclusion

In this study, we investigated ways of extending the capabilities of DEX, which is basically a sorting MCDM method, towards ranking. Specifically, we wanted to improve the old, but still state-of-the-art algorithm QQ (Bohanec, 1992) by addressing its weaknesses: the need to calculate attributes' weights, assuming equal weight ratios for all classes, and inability to properly consider unordered attributes.

The proposed novel algorithms, QQ2 and QL, build on the same idea as QQ: they evaluate decision alternatives in parallel using both a qualitative (class) evaluation and numeric evaluation, which is used to rank alternatives within each class. The two evaluations are kept consistent with each other, employing the $c \pm \omega$ representation of evaluation results, where C is the class, $c = \operatorname{ord} C$, and ω is a numeric ± 0.5 offset within the class.

In contrast with QQ, QQ2 and QL do not rely on the concept of attribute weights. Instead, they employ optimization models to separate decision rules as far apart as possible: QQ2 uses a quadratic and QL a linear optimization model. In this way, they achieve significantly better results in terms of separation and gap size in comparison with QQ. Also, they can properly handle unordered attributes.

By any means, QQ2 and QL are very similar to each other: they run fast and yield very similar results. They differ very little only in the "free" areas of the decision-rule space, i.e., decision rules that lie outside the longest paths of rule lattices. Despite that QQ2 has a somewhat more complex formulation of the optimization measure than QL, and employs quadratic rather than linear optimization algorithm, we chose it for implementation in DEXiWin for other advantageous characteristics: simple constraints and elegant formulation using squared distances, which are never negative, and require just a half of optimization variables and constraints than QL. Furthermore, QQ2 does not require setting up the upper bound q_U. In all experiments, it turned out very stable and non-problematic.

Overall, all the considered algorithms effectively bring the ranking functionality to DEX. Furthermore, they require no additional actions on behalf of the user (decision maker), as all information needed for ranking is extracted from existing decision tables and decision rules. However, this is a double-edged

sword. On one hand, it is given "for-free", and this is really convenient in practice. On the other hand, it makes assumptions about the user's preferences; the main assumptions are that preferentially related adjacent rules are separated by the same "preferential" distance all over the decision space, and that this distance needs to be maximized in order to separate evaluated alternatives as much as possible. This may or may not be true, and is not being verified with the user by any of the studied algorithms. The only way to do that is to explicitly ask the user if she or he agrees with offsets assigned to decision rules (such as the column QQ2 in Figure 14). In the case of disagreement, the user should be allowed to change the offsets (possibly overviewed by the software to keep the values within consistent boundaries). Although this might turn out a difficult task for the user, it is a relevant research topic for the future.

10 References

Belton, V., Stewart, T. J. (2002). *Multiple Criteria Decision Analysis: An Integrated Approach*. Springer. https://doi.org/10.1007/978-1-4615-1495-4.

Bohanec, M., Urh, B., Rajkovič (1992), V.: Evaluating options by combined qualitative and quantitative methods. *Acta Phychologhica* 80, 67–89. https://kt.ijs.si/MarkoBohanec/pub/ActaPsych1992.pdf.

Bohanec, M. (2017): Multi-criteria DEX models: An overview and analysis. *SOR-2017: 14th International Symposium on Operational Research in Slovenia*, Bled, Slovenia, September 27-29, 2017 (eds. Zadnik Stirn, L., et al.), Ljubljana: Slovenian Society Informatika, Section for Operational Research, 155–160. https://kt.ijs.si/MarkoBohanec/pub/2017_SOR_DEXmodels.pdf.

Bohanec, M.: DEX (Decision EXpert) (2022): A qualitative hierarchical multi-criteria method. In: *Multiple Criteria Decision Making* (ed. Kulkarni, A.J.), Studies in Systems, Decision and Control 407, Singapore: Springer, 39–78. https://doi.org/10.1007/978-981-16-7414-3.

Bohanec, M. (2024): DEXiWin: *DEX Decision Modeling Software, User's Manual, Version 1.2*. Institut Jožef Stefan, Delovno poročilo IJS DP-14747, 2024.
https://kt.ijs.si/MarkoBohanec/pub/2024_DP14747_DEXiWin.pdf.

Brelih, M., Rajkovič, U., Ružič, T., Rodič, B., Kozelj, D. (2019): Modelling decision knowledge for the evaluation of water management investment projects. *Central European Journal of Operations Research* 27, 759–781. https://link.springer.com/article/10.1007/s10100-018-0600-5.

Ferreira de Lima Silva, D., Ferreira, L., Teixeira de Almeida-Filho, A. (2020): A new preference disaggregation TOPSIS approach applied to sort corporate bonds based on financial statements and expert's assessment, *Expert Systems with Applications*, 152, 2020.
https://doi.org/10.1016/j.eswa.2020.113369.

Greco, S., Ehrgott, M., Figueira, J. (Eds.) (2016): *Multi Criteria Decision Analysis: State of the art Surveys*. New York: Springer. http://dx.doi.org/10.1007/978-1-4939-3094-4.

Kadziński, M., Greco, S., Słowiński, R. (2014): Robust Ordinal Regression for Dominance-based Rough Set Approach to multiple criteria sorting, *Information Sciences*, 283, 2014.
https://doi.org/10.1016/j.ins.2014.06.038.

Liu, J., Kadziński, M., Liao, X., Mao, X., Wang, Y. (2020): A preference learning framework for multiple criteria sorting with diverse additive value models and valued assignment examples, *European Journal of Operational Research*, 286(3), 2020. https://doi.org/10.1016/j.ejor.2020.04.013.

López, L.M., Ishizaka, A., Qin, J., Carrillo, P.A.Á. (2023): *Multi-Criteria Decision-Making Sorting Methods*. London: Academic Press. ISBN: 978-0-323-85231-9.

Mileva Boshkoska, B., Bohanec, M. (2012): A method for ranking non-linear qualitative decision preferences using copulas. *International Journal in Decision Support System Technology* 4(2), 42–58 (2012) http://dx.doi.org/10.4018/jdsst.2012040103.

Roy, B. (1996). *Multicriteria Methodology for Decision Aiding*. Springer. https://doi.org/10.1007/978-1-4757-2500-1.

Roy, B. (2016): Paradigms and challenges. In: Greco, S., Ehrgott, M., Figueira, J. (eds.) *Multi Criteria Decision Analysis: State of the Art Surveys*. Springer, New York. http://dx.doi.org/10.1007/978-1-4939-3094-4.

Ru, Z., Liu, J., Kadziński, M., Liao, X. (2023): Probabilistic ordinal regression methods for multiple criteria sorting admitting certain and uncertain preferences, *European Journal of Operational Research*, 311(2), 2023. https://doi.org/10.1016/j.ejor.2023.05.007.

Yatsalo, B., Radaev, A., Haktanir, E., Skulimowski, A.M.J., Kahraman, C. (2024): A family of fuzzy multi-criteria sorting models FTOPSIS-Sort: Features, case study analysis, and the statistics of distinctions, *Expert Systems with Applications*, 237(B), 2024. https://doi.org/10.1016/j.eswa.2023.121486.

Wang, L., Zhang, Z.-X., Ishizaka, A., Wang, Y.-M., Martínez, L. (2023): TODIMSort: A TODIM based method for sorting problems, *Omega*, 115, 2023. https://doi.org/10.1016/j.omega.2022.102771.

Appendix 1: Notation

DEX Model	$M = (X, D, S, F)$
Attributes	$X = \{x_i, i = 1, \dots, n\}$
Aggregate attributes (outputs)	$Y = \{x \in X: S(x) \neq \emptyset\}$
Basic attributes (inputs)	$Z = \{x \in X: S(x) = \emptyset\}$
Scales	$D = \{\text{scale}\, x_i \,\vert x_i \in X\}$
Scale	$\text{scale}\, x = [v_{x,1}, v_{x,2}, \dots, v_{x,m_x}]$
Scale order	$\text{order}\, x \in \{\text{unordered, ascending, descending}\}$
Decision space	$\text{space}\, K = \text{scale}\, x_1 \times \text{scale}\, x_2 \times \dots \times \text{scale}\, x_k,\ k = \vert K \vert$
Ordinal value	$\text{ord}\, v_{x,i} = i, i = 1,2, \dots, m_x$
Descendant function	$S: X \to 2^x$
Parents of x	$P(x) = \{p \in X: x \in S(p)\}$
Aggregation functions	$F = \{f_x, x \in Y\}$
… associated with $x \in Y$	$f_x: \text{space}\, S(x) \to \text{scale}\, x$
Function arguments	$\text{args}\, f_x = S(x)$
Decision table for $x \in Y$	$T_x: \text{space}\, S(x) \to \text{scale}\, x$ $T_x = \{r_i, i = 1,2, \dots, \vert\text{space}\, S(x)\vert\}$
Decision rule	$r \in T_x: \boldsymbol{x} \in \text{space}\, S(x) \to C \in \text{scale}\, x$
All elementary rules of T_x	$\text{rules}\, T_x = \{r_{\boldsymbol{x}}, \forall \boldsymbol{x} \in \text{space}\, S(x)\}$
Condition of rule r	$x(r) \in \text{space}\, x$
Class assigned by rule r	$c(r) \in \text{scale}\, x$
Class scale in the T_x context	$\text{scale}\, x = \{C_1, C_2, \dots, C_m\}$
Numeric value of rule r	$q(r) \in \mathbb{R}$
Values assigned to rules	$q(T) = [q_1, q_2, \dots, q_n] \in \mathbb{R}^n, n = \vert\text{rules}\, T\vert$
Numeric evaluation function	$Q_{T_x}: \mathbb{R}^k \to \mathbb{R}, k = \vert S(x)\vert$
Decision alternatives	$\mathcal{A} = \{A_1, A_2, \dots, A_q\}$
Alternative	$A_i = [a_{x,i} \in \text{scale}\, x, \forall x \in X]$
Input assignment	$Z(A_i) = [a_{x,i} \in A_i, \forall x \in Z]$
Output assignment	$Y(A_i) = [a_{x,i} \in A_i, \forall x \in Y]$
Evaluation of alternative A on x	$E_x(A)$
Alternatives sorted to some class C	$\mathcal{A}_x(C) = \{A \in \mathcal{A}: E_x(A) = C\}, x \in Y$
Principle of dominance	$\forall r, p \in \text{rules}\, T: x(r) \preccurlyeq x(p) \Rightarrow c(r) \preccurlyeq c(p)$
Dominance of rules w.r.t. '$\preccurlyeq$'	$\boldsymbol{x}_1 \preccurlyeq \boldsymbol{x}_2 \Leftrightarrow x_{1,i} \preccurlyeq x_{2,i}$, for $i = 1, \dots, \vert S_x\vert, \boldsymbol{x}_1, \boldsymbol{x}_2 \in \text{space}\, S(x)$
Combined qualitative-quantitative value	$c \pm \omega$, where C is class, $c = \text{ord}\, C, \omega \in [-0.5, +0.5]$
Adjacent rule pairs	$L = \{(i,j):$ $\boldsymbol{x}_i \preccurlyeq \boldsymbol{x}_j, i, j \in 1, \dots, n, j > i, \nexists \boldsymbol{x}_k: \boldsymbol{x}_i \preccurlyeq \boldsymbol{x}_k \preccurlyeq \boldsymbol{x}_j\}$
Neighbors of rule r_i	$N_i = \{j: (i,j) \in L \vee (j,i) \in L\}$
Lower and upper q bounds for optimization	$q_L, q_U \in \mathbb{R}, q_L < q_U$
Adjacent rules separation measure	$\frac{1}{\vert L\vert} \sum_{(i,j) \in L} \vert q_j - q_i\vert$