IJS delovno poročilo
DP-15020
2025

Marko Bohanec

Target Analysis in Qualitative Multi-Criteria Decision Modeling Method DEX

Jožef Stefan Institute
Ljubljana, Slovenija

DEPARTMENT OF KNOWLEDGE TECHNOLOGIES
Jožef Stefan Institute

Abstract

We investigate target analysis algorithms in the context of multi-criteria decision modeling method *DEX* (Decision EXpert). Given some decision alternative, *target analysis* is aimed at finding other decision alternatives (in terms of value assignments to input attributes), that change the overall evaluation of the original alternative in a positive or negative direction, and require the least changes of input values. Target analysis is a combinatorial problem whose search space increases exponentially with the number of input attributes. In this report, we investigate two algorithms, called *BottomUp* and *TopDown*, which use information contained within DEX models to limit the search space: internal structure of attributes, value scales of those attributes, and decision rules that govern the evaluation of alternatives. The two algorithms differ in the direction of constructing candidate solutions. *BottomUp* proceeds recursively from inputs toward the outputs, constructing solutions by combining best solutions found on lower-levels of the model. *TopDown* begins with the target value to be achieved at the top level, and then recursively investigates lower-level decision rules that lead to the corresponding target values. In this study, the two algorithms were experimentally evaluated on 54 DEX models of various sizes and characteristics, and compared with the *Exhaustive* algorithm which searches the whole decision space. *Exhaustive* was found, as expected, applicable to only small models having about 15 to 20 attributes at most. Both *BottomUp* and *TopDown* demonstrated good performance and were in most cases able to generate solutions in the range of seconds. *BottomUp* turned out particularly efficient, but unsuitable for DEX models that use linked attributes. *TopDown* seems to provide a good basis for practical implementation.

Keywords

Multi-criteria model, method DEX, target analysis, option generation, white-box algorithms, attribute tree, attribute hierarchy, decision rules.

Contents

1 Introduction

Mathematical and computational *models* are essential tools for representing, analyzing, and solving complex real-world problems across various domains (Shiflet, Shiflet, 2014; Humi, 2017). A computational model contains variables and other components that characterize the system being studied. Simulations are carried out by adjusting the variables alone or in combination and observing the outcomes[1].

Multi-criteria models are common types of models used In Decision Making and Decision Analysis (Buede, 2013). A *multi-criteria model* (Greco, et al., 2016) is a way of evaluating decision alternatives when there are multiple factors to consider. Instead of looking at just one criterion (like cost or speed), it takes several aspects into account, often balancing trade-offs between them. One of the key features of multi-criteria models is that, in addition to *evaluating* alternatives, they are capable of performing various types of *analyses* that support the decision maker in *exploring* the decision space in order to better understand the situation, reduce uncertainties and justify the decision. Typical types of analyses that are available to a decision maker and/or decision analyst are:

- *Sensitivity Analysis*: Assessing impacts of small changes. Typical approaches include one-factor-at-a-time, multi-factor and gradient-based sensitivity analysis.
- *Stability Analysis*: Identifying thresholds for substantial changes of outcomes, such as changes of alternatives' ranking.
- *Uncertainty Analysis*: Assessing effects of imprecise or uncertain data.
- *Comparative Analysis*: Comparing decision alternatives, and identifying reasons for different evaluations and/or different rankings.

In this report, we focus on a multi-criteria modelling method *DEX* (Decision EXpert) and a specific sensitivity analysis method called *Target Analysis*. 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.

Target Analysis is a kind of multi-factor analysis for DEX models. Given a DEX model and some alternative A_0 together with its evaluation (Figure 1), the task is to find one or more assignments of input attributes A' that change the evaluation value to some desired *target* value. Usually, we want to change the final evaluation to something *better* than before, and achieve this with the *least* changes of input values from A_0 to A'.

Target Analysis has only been recently introduced in DEX modeling. The only related scientific publications that address target analysis algorithms are by Gjoreski, et al. (2020; 2022). There, the authors recognize the target analysis task as a hard combinatorial problem, which may require substantial computational resources. The task is compared with counterfactual explanations (Guidotti, 2022), one of the key approaches of Explainable Artificial Intelligence (XAI) to make decisions more transparent and understandable, helping users to see what factors influenced an outcome and how they could alter it. The approach of Gjoreski, et al., which is in line with the majority of XAI approaches,

[1] https://www.nibib.nih.gov/science-education/science-topics/computational-modeling

considers DEX models as “black boxes”: it considers only model’s inputs and outputs, and disregards any information about its internal structure. The method proposed by Gjoreski, et al. (2022) is called BAG-DSM (Bayesian Alternative Generator for Decision Support Models) and relies on Bayesian Optimization in order to find solutions as quickly as possible. BAG-DSM was extensively evaluated on 42 different benchmark models and one real-life model, and was found “[…] suitable for the task, i.e., it generated at least one appropriate alternative in less than a minute, even for the most complex decision models”. A difficult part of the process turned out to be finding the first solution, particularly when solutions are scarce or may even not exist. Once some solution is found, it usually provides a suitable basis for further combinatorial search. The BAG-DSM approach is fairly general and can be used with qualitative model types other than DEX.

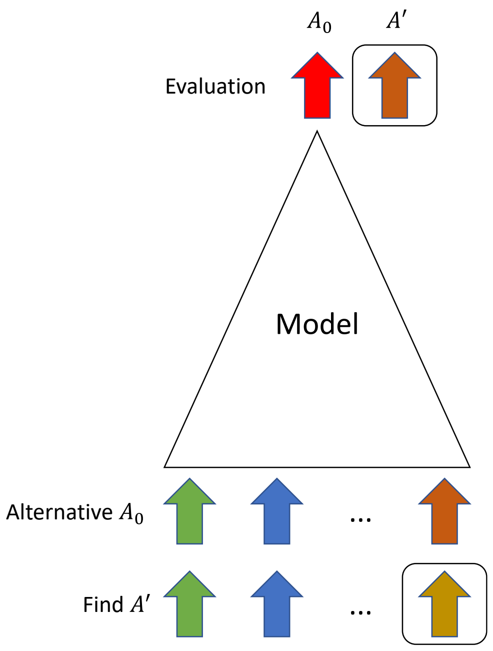


Figure 1: The task of Target Analysis.

In this study, we rather take the “white box” (also called “glass box”, “open box” or “transparent”) approach. We propose two algorithms, called *BotomUp* and *TopDown*, that exploit the inner structure of DEX models, in order to speed-up the search and assure finding all solutions A'. The disadvantage is that the algorithms are DEX-specific and cannot be generalized to other model types.

In what follows, we first present DEX models and their components, and introduce the formal notation. The target analysis is formally defined in section 3. In section 4, the algorithms *BottomUp* and *TopDown* are presented together with the algorithm *Exhaustive*, which is essentially a “naïve brute-force black-box” algorithm, used whenever possible for benchmarking and comparison. Results of extensive testing of the algorithms on 54 models are presented in section 5. Section 6 discusses the current implementations of target analysis and outlines additional requirements based on practical needs. Section 7 concludes the report.

2 Preliminaries and Notation

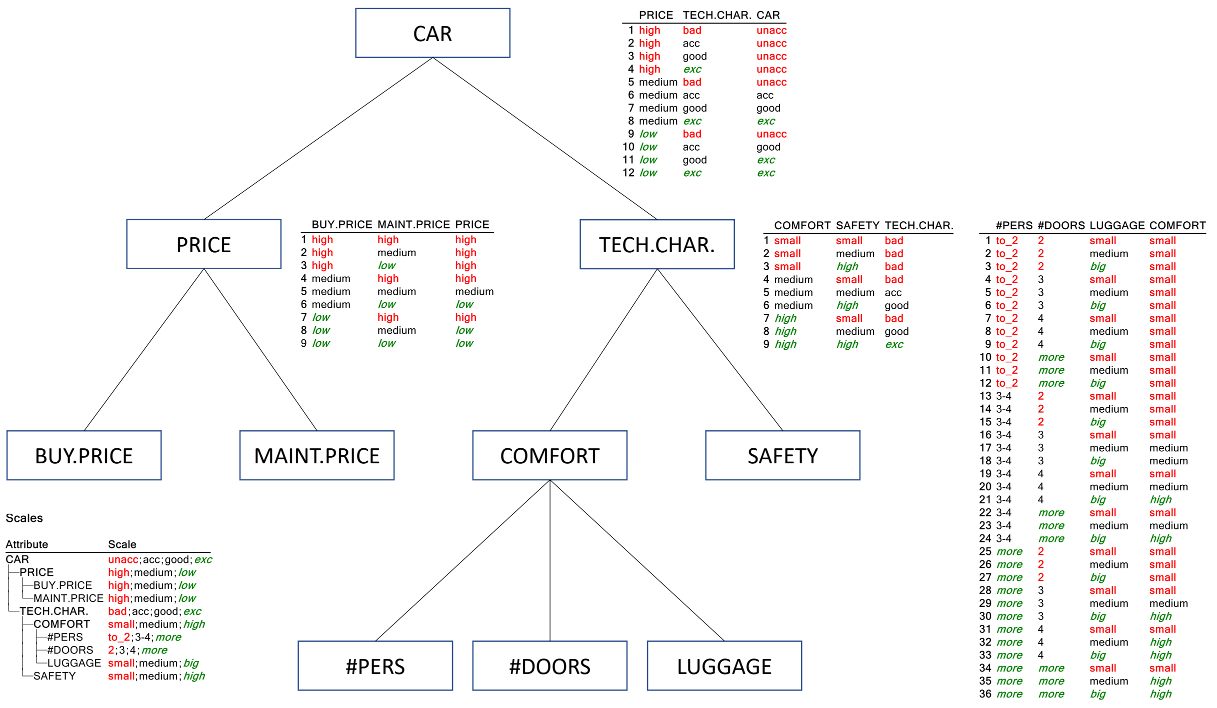
Formally, a DEX model M is defined as a four-tuple $M = (X, D, S, F)$, where X is the set of *attributes*, S is the descendant function that determines the hierarchical structure of attributes, D is the set of value scales of attributes in X, and F is the set of aggregation functions (Bohanec, 2022).

2.1 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 2). This is a simple model for evaluating personal cars, used for educational purposes, referred to in many publications and distributed with DEX software[2]. The model contains $n = 10$ attributes (presented in the depth-first order):

$X =$
$\{$*CAR, PRICE, BUY.PRICE, MAINT.PRICE, TECH.CHAR., COMFORT, #PERS, #DOORS, LUGGAGE, SAFETY*$\}$.



	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*

	BUY.PRICE	MAINT.PRICE	PRICE
1	high	high	high
2	high	medium	high
3	high	*low*	high
4	medium	high	high
5	medium	medium	medium
6	medium	*low*	*low*
7	*low*	high	high
8	*low*	medium	*low*
9	*low*	*low*	*low*

	COMFORT	SAFETY	TECH.CHAR.
1	small	small	bad
2	small	medium	bad
3	small	*high*	bad
4	medium	small	bad
5	medium	medium	acc
6	medium	*high*	good
7	*high*	small	bad
8	*high*	medium	good
9	*high*	*high*	*exc*

	#PERS	#DOORS	LUGGAGE	COMFORT
1	to_2	2	small	small
2	to_2	2	medium	small
3	to_2	2	*big*	small
4	to_2	3	small	small
5	to_2	3	medium	small
6	to_2	3	*big*	small
7	to_2	4	small	small
8	to_2	4	medium	small
9	to_2	4	*big*	small
10	to_2	*more*	small	small
11	to_2	*more*	medium	small
12	to_2	*more*	*big*	small
13	3-4	2	small	small
14	3-4	2	medium	small
15	3-4	2	*big*	small
16	3-4	3	small	small
17	3-4	3	medium	medium
18	3-4	3	*big*	medium
19	3-4	4	small	small
20	3-4	4	medium	medium
21	3-4	4	*big*	*high*
22	3-4	*more*	small	small
23	3-4	*more*	medium	medium
24	3-4	*more*	*big*	*high*
25	*more*	2	small	small
26	*more*	2	medium	small
27	*more*	2	*big*	small
28	*more*	3	small	small
29	*more*	3	medium	medium
30	*more*	3	*big*	*high*
31	*more*	4	small	small
32	*more*	4	medium	*high*
33	*more*	4	*big*	*high*
34	*more*	*more*	small	small
35	*more*	*more*	medium	*high*
36	*more*	*more*	*big*	*high*

Scales

Attribute	Scale
CAR	unacc; acc; good; *exc*
├PRICE	high; medium; *low*
│├BUY.PRICE	high; medium; *low*
│└MAINT.PRICE	high; medium; *low*
└TECH.CHAR.	bad; acc; good; *exc*
├COMFORT	small; medium; *high*
│├#PERS	to_2; 3-4; *more*
│├#DOORS	2; 3; 4; *more*
│└LUGGAGE	small; medium; *big*
└SAFETY	small; medium; *high*

Figure 2: The structure and components of CAR, a DEX model for evaluating personal cars.

2.2 Model Structure

Attributes in a DEX model are structured *hierarchically*. The structure is defined by the function $S: X \to 2^X$, 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, 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 $x \in \{BUY.PRICE, MAINT.PRICE, \#PERS, \#DOORS, LUGGAGE, SAFETY\}$.

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

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). As we shall see later in section 4, full hierarchies pose a challenge for target analysis algorithms.

For the purpose of this study, we extend the notation and partition X 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.

Let b denote the number of basic attributes: $b = |Z|$.

Furthermore, we shall assume that X, Y and Z are not just normal sets, but *topologically sorted* vectors (Trdin, Bohanec, 2018). For each $x_i \in Y$ this means that all $x \in S(x_i)$ must be positioned to the right of x_i in the vector. Since there are no cycles in the model, such order always exists. For Z we assume the depth-first-search order of basic attributes, and X is then just the concatenation of the two: $X = Y \circ Z$. This representation assures that attributes can be evaluated in a linear order. For any vector of values $\boldsymbol{a}$ assigned to X in this order, we can assume that the right-most b values represent assignments to basic attributes, and the left-most element represents the overall evaluation result.

For CAR, possible topological orders of Y and Z are:

$Y = [\mathit{CAR}, \mathit{PRICE}, \mathit{TECH.CHAR.}, \mathit{COMFORT}]$,

$Z = [\mathit{BUY.PRICE}, \mathit{MAINT.PRICE}, \mathit{\#PERS}, \mathit{\#DOORS}, \mathit{LUGGAGE}, \mathit{SAFETY}]$.

2.3 Attribute Scales

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

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

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 rather than numbers, for instance "low", "high", "unacceptable", "good".

DEX scales can be either *ordered* in an *ascending* or *descending* order, or are *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 preference relation. In *descending* scales, the relation $\succcurlyeq$ applies instead. No preferences can be assumed with *unordered* scales. We denote the order of scale x as

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

Traditionally, particularly **bad** and ***good*** scale values are printed in color, as shown in Figure 2. All scales in the CAR model are ascending.

In this study, we are not interested in individual scale values. In order to simplify notation, we shall represent all scale elements with the corresponding ordinal numbers. Therefore, we hereafter assume that

$\text{scale}\, x = [1, 2, \dots, m_x]$ for each $x \in X$.

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

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

2.4 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:

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

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

Aggregation functions are represented by *decision tables*, see examples in Figure 2. 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. While DEX in general allows incompletely defined tables and different types of output values (intervals, value distributions) (Bohanec, 2022), we shall hereafter assume that the tables are complete and that each decision rule prescribes exactly one output value from $\text{scale}\, x$. Possible complications arising from not fulfilling these assumptions are discussed later in section 6.

2.5 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 further partitioned in $Y(A_i)$ 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 using Algorithm 1. The algorithm iteratively computes output values, yielding a full assignment $A = \boldsymbol{a}$. The first element of $\boldsymbol{a}$ represents the overall evaluation of A, denoted $E(A) = a_1$.

Algorithm 1: Evaluate alternative.

```
Input: Alternative $A$, assignment $\boldsymbol{z} = [a_1, a_2, \dots, a_b] \in \text{space}\, Z$.

$\boldsymbol{a} = [0]^{n-b} \circ \boldsymbol{z}$
**for** $i \in [n-b, \dots, 1]$ **do**
    $a_i \coloneqq f_{x_i}(\boldsymbol{a}[\text{args}\, f_{x_i}])$

Output: Assignment $A = \boldsymbol{a} \in \text{space}\, X$, which includes the overall evaluation $E(A) = a_1$.
```

3 Target Analysis: Aims and Goals

Now we can formally define the target analysis task:

Given: An alternative $A_0 \in \text{space}\, X$,

Find: A set of alternatives $\mathcal{A}' = \{A'_1, A'_2, \dots\}$ that satisfy some predefined *requirements* with the *least* changes of input values from A_0 to each A'.

The *requirements* and *least changes* can be defined in different ways. Most often, we want to find alternatives A' whose evaluations are better than those of A_0: $E(A') \succ E(A_0)$. Or conversely, we might want to explore changes that degrade the evaluation: $E(A') \prec E(A_0)$. Sometimes we look for a particular target value: $E(A') = v \in \text{scale}\, x_1$. Notice that the former two conditions make sense only when the value scale of the root attribute x_1 is ordered.

To model the *least changes*, we introduce the function

$\text{diff}_K(A_1, A_2)\colon \text{space}^2\, K \to \mathbb{R} \cup \{\text{undefined}\};\ K \subseteq X; A_1, A_2 \in \text{space}\, K.$

This is essentially a distance function, extended to consider subsets of attributes $K \subseteq X$ and facilitate an "undefined" comparison as indication of an undesired solution. Unlike distances, we accept negative values of diff_K, but always aim to minimize the function. When comparing A_0 and A', we set the arguments so that $A_1 = A'$ and $A_2 = A_0$.

Again, several different formulations of $\text{diff}_K(A_1, A_2)$ are possible. Generally, diff_K is defined as a sum of differences corresponding to individual attributes $x_i \in K, i = 1,2, \dots, |K|$:

$$\text{diff}_K(A_1, A_2) = \sum_{i=1}^{|K|} \text{diff}(x_i, a_{i,1}, a_{i,2})$$

Here, $a_{i,j}$ denotes the value of A_j corresponding to x_i, i.e., the i-th element of A_j. Whenever any $\text{diff}(x_i, a_{i,2}, a_{i,1})$ term yields an undefined result, the whole sum is considered undefined, too.

Then, when we want to improve the overall evaluation of A_0, we may allow changing input attributes only in the "better" direction, and disallow all changes for the "worse". For unordered attributes, we just notify the difference. This leads to the formulation of *Unidirectional difference "for the better"*: as follows:

$$\text{diff}(x, a_1, a_2) \equiv \text{unidiff}^+(x, a_1, a_2) = \begin{cases} a_2 - a_1 & \Leftarrow \text{order}\, x = \text{ascending} \wedge a_2 \geq a_1 \\ \text{undefined} & \Leftarrow \text{order}\, x = \text{ascending} \wedge a_2 < a_1 \\ a_1 - a_2 & \Leftarrow \text{order}\, x = \text{descending} \wedge a_1 \geq a_2 \\ \text{undefined} & \Leftarrow \text{order}\, x = \text{descending} \wedge a_1 < a_2 \\ 0 & \Leftarrow \text{order}\, x = \text{unordered} \wedge a_1 = a_2 \\ 1 & \Leftarrow \text{order}\, x = \text{unordered} \wedge a_1 \neq a_2 \end{cases}$$

Conversely, when we are interested in changes that degrade the evaluation, we may use the *Unidirectional difference "for the worse"*:

$$\text{diff}(x, a_1, a_2) \equiv \text{unidiff}^-(x, a_1, a_2) = \begin{cases} a_1 - a_2 & \Leftarrow \text{order}\, x = \text{ascending} \wedge a_2 \leq a_1 \\ \text{undefined} & \Leftarrow \text{order}\, x = \text{ascending} \wedge a_2 > a_1 \\ a_1 - a_2 & \Leftarrow \text{order}\, x = \text{descending} \wedge a_1 \leq a_2 \\ \text{undefined} & \Leftarrow \text{order}\, x = \text{descending} \wedge a_1 > a_2 \\ 0 & \Leftarrow \text{order}\, x = \text{unordered} \wedge a_1 = a_2 \\ 1 & \Leftarrow \text{order}\, x = \text{unordered} \wedge a_1 \neq a_2 \end{cases}$$

As a third measure we define *Bidirectional difference*. This time, we consider changes in both directions, “better” and “worse”. Changes in the “better” direction increase the difference, while changes in the “worse” decrease it. This means that we are still looking for overall improvement of alternatives, but accept – and even reward – input assignments that change for the worse. Consequently:

$$\mathrm{diff}(x, a_1, a_2) \equiv \mathrm{bidiff}(x, a_1, a_2) = \begin{cases} a_2 - a_1 & \Leftarrow \text{order}\, x = \text{ascending} \\ a_1 - a_2 & \Leftarrow \text{order}\, x = \text{descending} \\ \quad 0 & \Leftarrow \text{order}\, x = \text{unordered} \wedge a_1 = a_2 \\ \quad 1 & \Leftarrow \text{order}\, x = \text{unordered} \wedge a_1 \neq a_2 \end{cases}$$

4 Target Analysis Algorithms

Before going to the target analysis algorithms themselves, let us make a few remarks. First, while target analysis is relatively new to DEX, a similar analysis called “Plus-Minus-1” or “Plus-Minus” analysis has been there for a long time (Bohanec, 2022). This is essentially a one-attribute-at-a-time analysis: assessing the effects of changing one attribute while keeping the remaining ones constant. We mention this analysis here because it is very simple and can be carried out efficiently: it requires only $\sum_{x \in X}(m_x - 1)$ evaluations. Considering that m_x are small and have a small upper bound, plus-minus analysis is of linear time complexity $O(|X|) = O(n)$.

Second, looking at small DEX models, such as CAR in Figure 2, one might think that target analysis is simple, too. Indeed, the number of all possible CAR input assignments is $|\text{space}\, Z| = \prod_{x \in Z} m_x = 3 \times 3 \times 3 \times 4 \times 3 \times 3 = 972$. It is not difficult at all to make this number of evaluations.

However, the above formula for $|\text{space}\, Z|$ indicates that the number of possible input assignments increases exponentially in $b = |Z|$. As shown in section 5, this number becomes too large even for models of moderate size, i.e., having about 15 criteria. Therefore, there is a real need for a more efficient algorithm. This is exemplified by the fact that target analysis is usually run interactively on laptop or desktop computers, where results are expected within seconds, not in minutes or even hours.

4.1 Algorithm: Exhaustive

The first algorithm used in this study, called *Exhaustive*, is exactly the “inefficient” algorithm mentioned above: it evaluates all possible input assignments to a given model M and records solutions A' that minimize $\mathrm{diff}_Z(A', A_0)$ for each value of x_1.

Algorithm 2: Exhaustive.

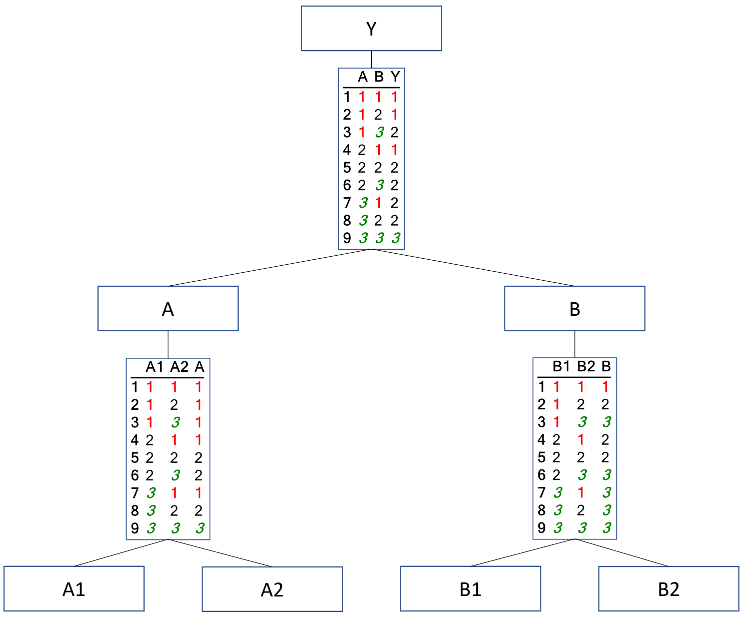
```
Input:
   Alternative $A_0$, assignment $\mathbf{z_0} = [a_1, a_2, \dots, a_b] \in \text{space}\, Z$.
   Difference measure diff (one from section 3)

initialize $R$
for all $\mathbf{z}' \in \text{space}\, Z, \mathbf{z}' \neq \mathbf{z}_0$ do
   $A' := \mathit{Evaluate}(\mathbf{z}')$
   $R \leftarrow \mathit{Record}(E(A'), A', \mathrm{diff}_Z)$

Output: Recorded solutions $R$ for each $E(A') \in \text{scale}\, x_1$.
```

In Algorithm 2, $Evaluate$ refers to Algorithm 1. R denotes a data structure that keeps a record of best assignments A' for each $E(A') \in \text{scale}\, x_1$. "Best" means all assignments that minimize the chosen difference measure diff. Undefined evaluations are not recorded.

We use *Exhaustive* in this study only as a benchmark algorithm: to assess the time complexity of evaluating all possible input assignments and to compare *Exhaustive*'s results with results of the other two algorithms. *Exhaustive* is so simple that it is almost impossible to be in error, while this is not necessarily true for *BottomUp* and *TopDown*.

4.2 Algorithm: BottomUp

As suggested by its name, the idea of the *BottomUp* algorithm is to traverse a DEX model in the bottom-up direction, i.e., from basic attributes to the root, and record best solutions at a given attribute $x \in X$ along the way. As we shall see shortly, solutions for a basic attribute $x \in Z$ are trivial. Solutions for an aggregate attribute $x \in Y$ are then constructed by combining best solutions corresponding to its descendants $S(x)$, considering decision rules in the associated decision table T_x.



	A	B	Y
1	1	1	1
2	1	2	1
3	1	3	2
4	2	1	1
5	2	2	2
6	2	3	2
7	3	1	2
8	3	2	2
9	3	3	3

	A1	A2	A
1	1	1	1
2	1	2	1
3	1	3	1
4	2	1	1
5	2	2	2
6	2	3	2
7	3	1	1
8	3	2	2
9	3	3	3

	B1	B2	B
1	1	1	1
2	1	2	2
3	1	3	3
4	2	1	2
5	2	2	2
6	2	3	3
7	3	1	3
8	3	2	3
9	3	3	3

Figure 3: A simple DEX model that implements $Y = \text{average}(\min(A_1, A_2), \max(B_1, B_2))$.

Let us first sketch the algorithm using a simple model in Figure 3. Let us analyze alternative $A_0 = [1,1,1,1]$ and use the difference measure unidiff^+. Consider that scales of all attributes are [1,2,3].

For basic attributes $x \in Z$, best solutions are just one-element vectors of differences between each value $v \in \text{scale}\, x$ and the element of A_0 that corresponds to x. Therefore, best solutions of attribute *A1* are:

- for $A1 = 1$: solution = [1], difference = 0
- for $A1 = 2$: solution = [2], difference = 1
- for $A1 = 3$: solution = [3], difference = 2

Since all values of A_0 are equal, the same solutions are also the best for the remaining three basic attributes.

Then, we move to aggregate attribute *A*. There we look at the associated decision table. There are five rules that map to *A*=1 (rules 1, 2, 3, 4, and 7 in Figure 3). The table columns *A1* and *A2* indicate which lower-level solutions need to be combined and possibly recorded as best solutions for *A*=1. There are five such combinations, as shown in Table 1. Among the five candidate solutions, only one, [1,1], has a minimum difference of 0 and is thus recorded as the best solution for value 1 of attribute *A*.

Table 1: Candidate value vectors for finding best solutions for value 1 of attribute A.

Rule	*A1*	*A2*	*Candidate solution*	*Difference with A_0*
1	1	1	[1,1]	0
2	1	2	[1,2]	1
3	1	3	[1,3]	2
4	2	1	[2,1]	1
7	3	1	[3,1]	2

Solutions for other values of *A* and for all values of *B* and *Y* are generated by the same token and are shown in Table 2. Since *BottomUp* generates solutions for all values of *Y*, it can answer any target analysis question formulated in section 3.

Table 2: BottomUp *solutions for* $A_0 = [1,1,1,1]$.

Attribute	*Attribute value*	*Best solutions*	*Difference with A_0*
A	1	[1,1]	0
	2	[2,2]	2
	3	[3,3]	4
B	1	[1,1]	0
	2	[1,2], [2,1]	1
	3	[1,3], [3,1]	2
Y	1	[1,1,1,2], [1,1,2,1]	1
	2	[1,1,1,3], [1,1,3,1]	2
	3	[3,3,1,3], [3,3,3,1]	6

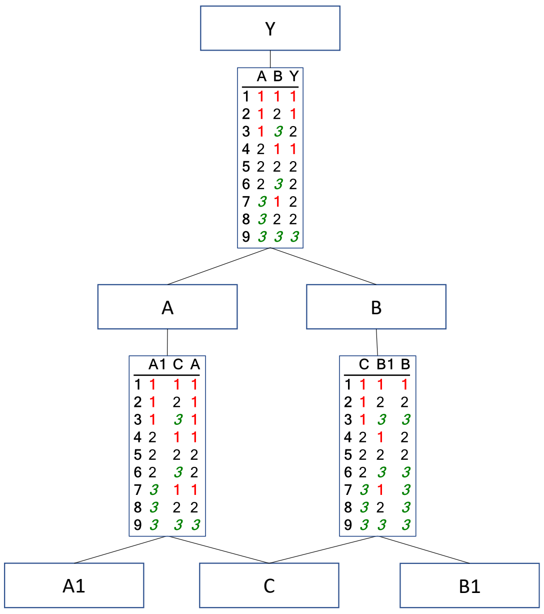
Algorithm 3: BottomUp.

```
Input:
  Alternative $A_0$, assignment $\boldsymbol{z_0} = [a_1, a_2, \dots, a_b] \in \text{space}\, Z$.
  Difference measure diff (one from section 3).

**for** $i \in [n, \dots, 1]$ **do**
  initialize $R_{x_i}$
  **if** $x_i \in Z$ **then**
    **for** each $v \in \text{scale}\, x$ **do**
      $R_{x_i} \leftarrow \mathit{Record}(v, [v], \text{diff}_{\text{space}\, x_i})$
  **else**
    **for** each $\boldsymbol{a} \in \text{space}\, S(x)$ **do**
      $v \coloneqq f_{x_i}(\boldsymbol{a})$
      **for** all $\boldsymbol{s} \in \mathit{CandidateSolutions}(x, \boldsymbol{a})$ **do**
        $R_{x_i} \leftarrow \mathit{Record}(v, \boldsymbol{s}, \text{diff}_{\text{space}\, S(x)})$

Output: Recorded solutions $R_x$ for each attribute $x \in X$ and each value $v \in \text{scale}\, x$.
```

This example generalizes to Algorithm 3. The algorithm loops through all attributes in the backward direction. For each visited attribute x_i, it records best solutions in the corresponding data structure R_{x_1}. For basic attributes, the solutions consist only of single-value vectors $[v]$ and associated differences for each attribute value v. For aggregate attributes, the algorithm considers all decision rules and all combinations of lower-level solutions, arising from each rule, that map to some attribute value v. Given x and some assignment of function arguments $\boldsymbol{a} \in$ space $S(x)$, candidate solutions are defined as a Cartesian product:

$$CandidateSolutions(x, \boldsymbol{a}) = R_{s_1}[a_1] \times R_{s_2}[a_2] \times \ldots \times R_{s_k}[a_k]$$

Here, s_i represent immediate descendants of x so that $S(x) = \{s_1, s_2, \ldots, s_k\}$. The notation $R_s[a]$ represents all solutions corresponding to attribute s and its value $a \in$ scale s.



	A	B	Y
1	1	1	1
2	1	2	1
3	1	3	2
4	2	1	1
5	2	2	2
6	2	3	2
7	3	1	2
8	3	2	2
9	3	3	3

	A1	C	A
1	1	1	1
2	1	2	1
3	1	3	1
4	2	1	1
5	2	2	2
6	2	3	2
7	3	1	1
8	3	2	2
9	3	3	3

	C	B1	B
1	1	1	1
2	1	2	2
3	1	3	3
4	2	1	2
5	2	2	2
6	2	3	3
7	3	1	3
8	3	2	3
9	3	3	3

Figure 4: A simple DEX model that implements $Y = \text{average}(\min(A_1, C), \max(C, B_1))$.

As shown later in section 5, *BottomUp* is very efficient and works well for models structured as ordinary trees. However, it has a major flaw: it does not work well with general hierarchies. While it still finds some solutions, it usually misses some others. The reason is that with general hierarchies, best solutions associated with some aggregate attribute x are not necessarily composed of best solutions associated with its descendants $S(x)$. Let us illustrate this with an example, using the model in Figure 4. This is essentially the same model as in Figure 3, except that attributes *A2* and *B1* have been merged to *C*, and *B2* has been renamed to *B1*. Decision tables remained exactly the same.

Table 3: Exhaustive *solutions for* Y *and* $A_0 = [1,1,1,1]$. *The asterisk * marks solutions found by* BottomUp.

Attribute	*Attribute value*	*Best solutions*	*Difference with* A_0
Y	1	$[1,1,2]^*$, $[1,2,1]$, $[2,1,1]$	1
	2	$[1,1,3]^*$, $[1,3,1]$, $[2,2,1]^*$	2
	3	$[3,3,1]^*$	4

Traversing the hierarchy using the same assignment $A_0 = [1,1,1,1]$ gives the same best solutions for *A* and *B*. However, the best solutions for *Y* are different than before and consist of three rather than four values assigned to basic attributes. All solutions found by *Exhaustive* are shown in Table 3. *BottomUp* does find four of those solutions (marked with * in Table 3), but misses three others. For instance, it misses the solution [1,3,1] for *Y*=2. The reason is that this solution would require concatenating two lower-level solutions [1,3] and [3,1] (with *C*=3 the common element). However, in Table 2, there is no element [1,3] associated with *A* that can be concatenated with [3,1] of *B*.

In general, when an aggregate attribute x has subordinate attributes that form a hierarchy rather than a tree, the best solutions associated with that attribute are not necessarily composed of best solutions associated with its descendants $S(x)$. This is a serious limitation for using *BottomUp* with full hierarchies, and a motivation for introducing another algorithm that properly handles such situations.

4.3 Algorithm: TopDown

The *TopDown* algorithm approaches target analysis in the opposite direction than *BottomUp*: given some target value v_1 of the root attribute x_1, it traverses the associated decision table, looking for decision rules that map to v_1. Arguments of each decision rule provide target values for recursive applications of the algorithm in subtrees below x_1. When reaching a basic attribute, its target value is assigned to the vector of input values, which is gradually built in the process. Once all inputs have been assigned, they are recorded as a possible candidate solution. The algorithm proceeds until all aggregate attributes have been visited and all decision rules that apply have been checked. All these steps are presented with more details in Algorithm 4 – Algorithm 7.

Algorithm 4: TopDown, main part.

```
Input:
  Alternative $A_0$, assignment $\mathbf{z_0} = [a_1, a_2, \dots, a_b] \in \text{space } Z$.
  Target value $v_1 \in \text{scale } x_1$.
  Difference measure diff (one from section 3).

initialize $R$
$\boldsymbol{a} \coloneqq [v_1] \circ [0]^{n-1}$        // initialize the vector of $n$ assignments; $0$ denotes no assignment yet
$VisitAttribute(1, \boldsymbol{a})$        // recursively search for solutions, starting with the first (root) attribute

Output: Recorded solutions $R$ for attribute $x_1$ and value $v_1$.
```

Algorithm 5: TopDown, method VisitAttribute.

```
method $VisitAttribute(i, \boldsymbol{a})$
  Inputs:
    Attribute index $i$
    $\boldsymbol{a}$: Vector of $n$ assignments developed up to the $(i-1)$-th aggregate attribute.

if $i > n - b$ then                  // visited all aggregate attributes, record the candidate solution
    $R \leftarrow Record(v_1, Z(\boldsymbol{a}), \text{diff}_Z)$
else if $CanContinue(\boldsymbol{a})$ then          // check if it makes sense to continue
   for each rule $r \in T_{x_i}$ that maps to target $a_i$ do
      $\mathbf{r} \coloneqq \text{args } r$                // vector of function arguments, has $|S(x_i)|$ elements
      if $CanAssign(i, \boldsymbol{r} \text{ to } \boldsymbol{a})$ then
         $VisitAttribute(i + 1, \text{copy } \boldsymbol{a})$      // continue searching attribute $i + 1$
```

Algorithm 4 displays the main skeleton of *TopDown* that consists of two initializations and a top-level call of *VisitAttribute*. The latter (Algorithm 5) actually does all the hard work by visiting each aggregate attribute whose index in X is i. The method iterates over decision rules that map to the current target value a_i, making candidate assignments considering the descendant attributes of i, and running *VisitAttribute* recursively for the next attribute $i+1$. When all aggregate attributes have been visited (condition $i > n - b$), the assignment $\boldsymbol{a}$ is complete and ready for checking: it is recorded in R whenever its difference with $\boldsymbol{z}_0$ is lower than or equal to the lowest difference recorded so far. Finding a lower difference discards all previously recorded solutions. The assignment equal to $\boldsymbol{z}_0$ is not recorded.

Algorithm 6: TopDown, *method* CanContinue.

```
method $CanContinue(\boldsymbol{a})$: returns Boolean
  Input:  Current assignment $\boldsymbol{a}$.

$d \coloneqq \min$ difference recorded in $R$
return $\mathrm{diff}_Z(Z(\boldsymbol{a})) \leq d$          // here, $\mathrm{diff}_Z$ should disregard zero elements of $Z(\boldsymbol{a})$
```

There are two important methods called in

Algorithm 5: *CanContinue* and *CanAssign*. The former (Algorithm 6) checks if the difference of the current assignment to basic attributes $Z(\boldsymbol{a})$ (disregarding zero elements) with $\boldsymbol{z}_0$ is lower than or equal to the minimal difference recorded so far in R. If not, it makes no sense to continue, and the remaining part of the decision space can be skipped.

Algorithm 7: TopDown, *method* CanAssign.

```
method $CanAssign(i, \boldsymbol{r}$ to $\boldsymbol{a})$: returns Boolean
  Inputs:
    Attribute index $i$.
    $\boldsymbol{r}$: Vector of $k$ values, $k = |S(x_i)|$, to be assigned to consecutive descendants of attribute $x_i$.
    Current assignment $\boldsymbol{a}$.

for each $j = 1,2,\dots,k$ do
  $q \coloneqq$ index of attribute $x_j$ in the topologically ordered $X$
  if $a_q > 0$ and $a_q \neq r_j$ then          // already been there and the value is different
    undo changes made to $\boldsymbol{a}$
    return false
  else
    $a_q \coloneqq r_j$
    return true

Outputs:
  •  Returns true: Values of $\boldsymbol{r}$ assigned to proper indices of $\boldsymbol{a}$.
  •  Returns false: Unchanged $\boldsymbol{a}$.
```

The role of *CanAssign* (Algorithm 7) is twofold. First, it checks if the value a_i of the current attribute i has been already set. This can happen with full hierarchies when attribute x_i has been already reached by traversing some other branch of the hierarchy. The search for solutions can continue only if the previous and the current value of a_i are equal, otherwise the solution does not exist. Notice that in the case of equal values, the search has to continue, because different hierarchy branches may generate different assignments to basic attributes and thus different overall solutions. In this way, *TopDown* resolves the *BottomUp's* issue with full hierarchies (section 4.2).

The second role of $CanAssign$ is to actually assign to $\boldsymbol{a}$ target values that correspond to the i-th attribute's descendants and thus prepare $\boldsymbol{a}$ for the next $VisitAttribute$ step.

An example of *TopDown*'s operation is presented in Appendix 1.

5 Experimental Evaluation

The three target analysis algorithms, *Exhaustive*, *BottomUp* and *TopDown*, were experimentally evaluated using 54 DEX models of various sizes and characteristics (Table 4 in Appendix 2). Most of them were artificially constructed in order to explore small vs. large models, trees vs. hierarchies, and models that facilitate using *Exhaustive* and those that do not. Models whose names start with "bm_" are exactly the same benchmark models as used in the study of Gjoreski, et al. (2022). Models Car (Figure 2) and those named "Agri…" can be considered real-life; the latter are three different modifications of a model used in the decision-support system PATHFINDER[3], which is aimed at the assessment and management of sustainability of legume agri-food value chains.

Experimental models are from 2 to 4 levels deep and contain from 7 to 65 attributes, from which there are 2 to 39 basic attributes (Table 4). 25 models contain at least one linked attribute, i.e., they are hierarchies rather than ordinary trees. The column "Alternatives" in Appendix 2 displays the number of alternatives that were taken as A_0 in the experiments. Whenever the associated decision space was reasonably small enough to facilitate exhaustive experimentation, all possible assignments in that space were considered. For larger models, we took a sample of 100 random or predefined (from Gjoreski, et al., 2022) alternatives.

The experiments were run using all algorithms, all corresponding alternatives and all three difference measures. Whenever running time exceeded a few minutes, these settings were considered inappropriate for practice and some elements (algorithms, difference measures) were excluded. All algorithms were implemented in the same compiled language (Oxygene) in the same environment (Visual Studio 2019). All experiments were run on two different desktop computers. In Appendix 2, Table 5, we show execution times achieved on a slower one (Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz, running Windows 10).

Whenever more than one algorithm has been used on the same model and alternatives, results were compared. In most cases, results of different algorithms were identical, which is a good indication of the correctness of the algorithms and their implementations. The only differences occurred with *BottomUp* and full hierarchies, where the algorithm failed to find all possible solutions, as already indicated in section 4.2.

Looking at execution times (Table 5 and Figure 5), it is clear that *Exhaustive*'s time complexity is indeed exponential. In the experiments, *Exhaustive* could have been reasonably used with models consisting of about 15 attributes, where execution times per alternative were still in the range of seconds. In practice, this might be increased by a few attributes at most, which would push the execution times in the range of minutes and hours.

On the other side of the spectrum is *BottomUp*, which runs in just a few milliseconds even with the largest models. Its only disadvantage is that it does not find all possible solutions with full hierarchies.

TopDown is somewhere between the other two. It turned out suitable for models having up to 40 attributes, where average execution time was in the range of one second. Actually, execution times were much lower for the majority of models; the peak around 30 attributes (Figure 5) can be attributed

[3] https://true-project.webarchive.hutton.ac.uk/publications-resources/decision-support-system-dss/index.htm

to two “AgriFood…” models, which contain large decision tables (each having 243 decision rules), and model “bm_lvl_4_skewed_2_w_links”, which turned out particularly difficult when using the bidirectional difference measure.

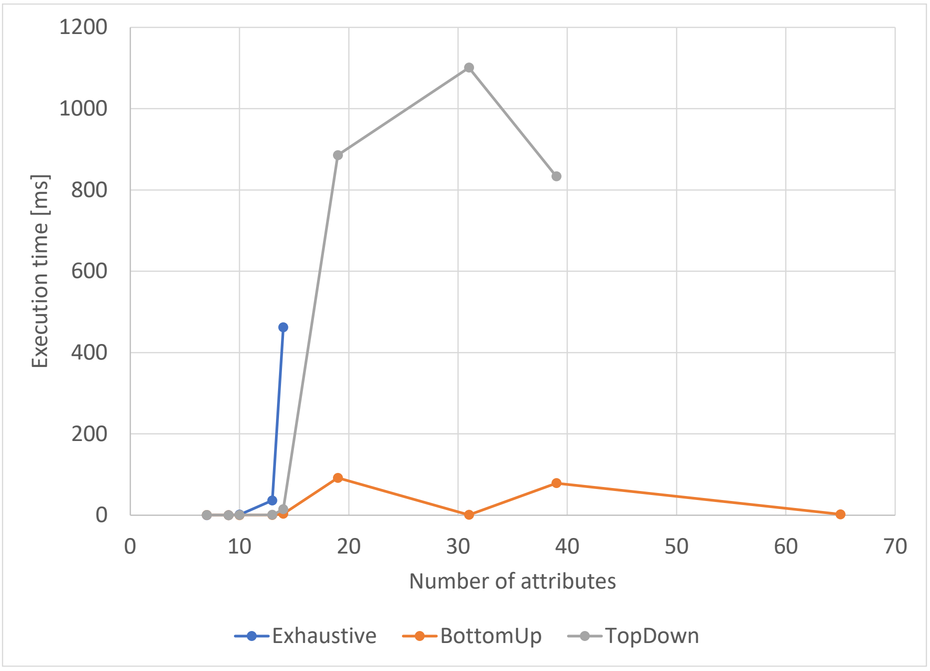


Figure 5: Average execution times per alternative, in milliseconds, depending on the number of all attributes.

6 Implementation Considerations

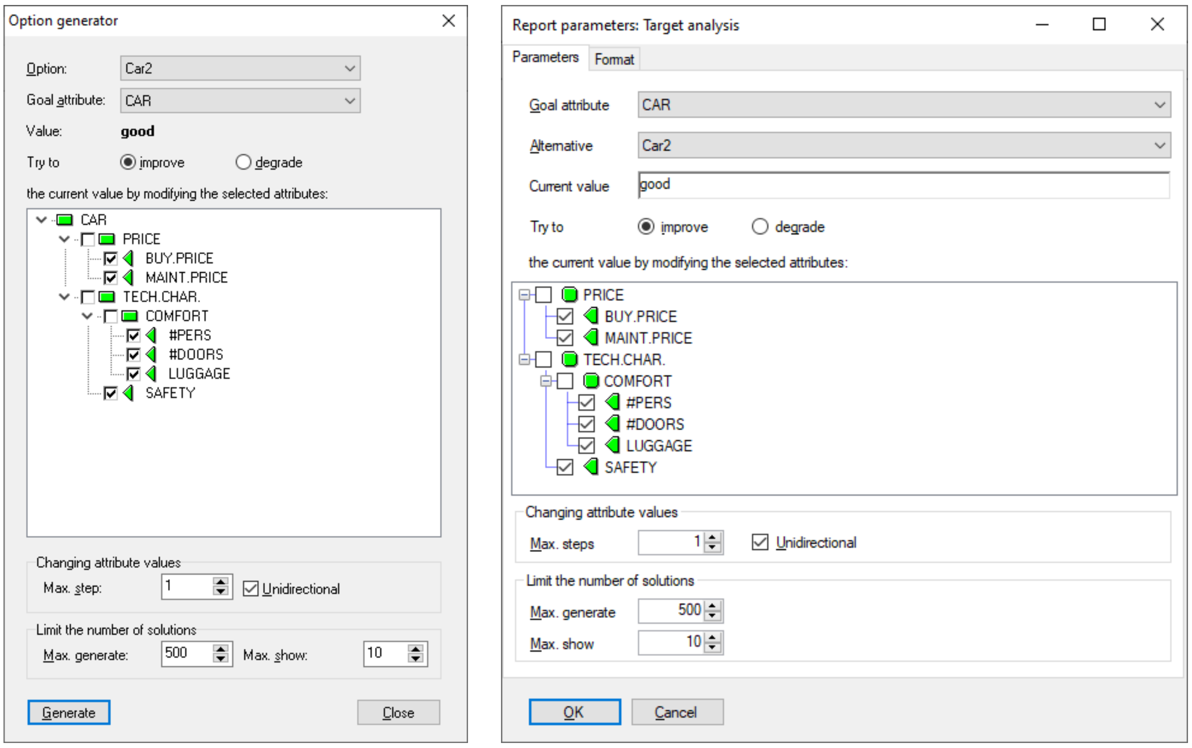
The three target analysis algorithms were implemented specifically for this study. This allowed for a “clean” and efficient implementation, free from dealing with issues that inevitable emerge in practical applications. Any realistic implementation must consider a number of additional requirements and “complications”, which make it more complex and generally slower. These include:

1. *Incomplete models*. DEX models are generally developed interactively and in successive steps. At any time, they may be incomplete, having undefined attribute scales or decision tables. Target analysis algorithm cannot run on incomplete models; therefore, a preliminary completeness test has to be performed.
2. *Linked attributes*. As already discussed, linked attributes turn attribute trees to hierarchies, which require special consideration and make *BottomUp* inferior in relation with the other two algorithms. Implementation-wise, we always convert a DEX attribute tree, which turned out to be difficult to work with directly, especially when containing linked attributes, to a specially designed hierarchy representation prior to applying the algorithms.
3. *Decision rules mapping to value sets*. In this study we assumed that each rule maps its arguments to exactly one value of the output attribute. In reality, rules may map to value intervals or sets. This is actually not a big hurdle for target analysis algorithms, as each such rule can be “virtually” expanded to a set of rules, each mapping to a single output value from the set. This just complicates the implementation.
4. *Decision rules not mapping to all values of the output attribute*. Theoretically, this is not a problem, as it just restricts the number of solutions, possibly to no solutions at all. Implementation-wise, this requires special care and may cause errors if not implemented correctly.
5. *Considering model parts*. In this study, we always considered the model root as the target variable, and all basic attributes as inputs. In practice, it is often necessary to consider just a part of the

model, taking some internal aggregate attribute as the root, and excluding some individual basic attributes or even whole branches (known as model "pruning").

6. *Restricting attribute values.* Apart from excluding some attributes from consideration, it is very practical if the user can restrict values that can be assigned to individual input attributes, for example to disregard changes that cannot occur at all, or just to reduce the search space. A useful restriction is to allow changing an attribute only for a given number of steps in one or both directions.
7. *Runtime considerations.* Despite that *TopDown* and *BottomUp* are fairly efficient, they may still take too much time with very large models. Also, they might "explode" and generate an excessive number of solutions. For this reason, a practical implementation must allow interrupting the execution or specifying the time limit, and limiting the maximum number of solutions generated or shown to the user.

Target analysis is currently implemented in DEX modelling software DEXi[4] (Bohanec, 2020) and DEXiWin[5] (Bohanec, 2024). In DEXi, it is called "Option Generator", named in the sense that the algorithm generates new options (alternatives) from the current one. In both cases, the implemented algorithm is of the *TopDown* kind, enhanced in order to accommodate the implementation issues mentioned above. Figure 6 shows the settings used to parameterize the algorithms in DEXi and DEXiWin.


Option generator

Option: Car2
Goal attribute: CAR
Value: **good**
Try to [✓ improve] degrade
the current value by modifying the selected attributes:

- CAR
  - ☐ PRICE
    - ☑ BUY.PRICE
    - ☑ MAINT.PRICE
  - ☐ TECH.CHAR.
    - ☐ COMFORT
      - ☑ #PERS
      - ☑ #DOORS
      - ☑ LUGGAGE
    - ☑ SAFETY

Changing attribute values
Max. step: 1 ☑ Unidirectional

Limit the number of solutions
Max. generate: 500 Max. show: 10

Report parameters: Target analysis

Goal attribute: CAR
Alternative: Car2
Current value: good
Try to [✓ improve] degrade
the current value by modifying the selected attributes:

- ☐ PRICE
  - ☑ BUY.PRICE
  - ☑ MAINT.PRICE
- ☐ TECH.CHAR.
  - ☐ COMFORT
    - ☑ #PERS
    - ☑ #DOORS
    - ☑ LUGGAGE
  - ☑ SAFETY

Changing attribute values
Max. steps: 1 ☑ Unidirectional

Limit the number of solutions
Max. generate: 500
Max. show: 10

Figure 6: Target Analysis settings in DEXi (left) and DEXiWin (right).

7 Conclusion

Target analysis is a task of finding assignments to input attributes that change the evaluation of some alternative to some target value, either better or worse than before. We proposed three algorithms for decision models of the DEX (Decision Expert) type. The first algorithm, *Exhaustive*, is used only for benchmarking, as it searches the whole decision space and is of exponential time complexity. Also, *Exhaustive* does not use any information about the inner structure of DEX models, and is thus of the

[4] https://kt.ijs.si/MarkoBohanec/dexi.html
[5] https://dex.ijs.si/dexisuite/dexiwin.html

"black-box" type. The remaining two algorithms, *BottomUp* and *TopDown*, are of the "white-box" type, because they do consider inner elements of DEX models: hierarchical structure of attributes, value scales of attributes, and decision tables and decision rules associated with aggregate attributes. The two algorithms differ in the primary direction of their operation. *BottomUp* constructs solutions recursively, starting from input attributes and lower-level subtrees towards the root of the model. On the contrary, *TopDown* gradually builds solutions from the root towards the inputs, or, considering the topological order of attributes, from left to right.

Experimental evaluation of the three algorithms was carried out using 54 DEX models of various sizes and characteristics, and using three difference measures: two unidirectional (in positive and negative direction) and bidirectional. The main findings are:

- *Exhaustive* is very simple to implement, but has exponential time complexity with respect to the number of input attributes. This limits its application to small models that consist of at most 15 or 20 attributes.
- *BottomUp* is most efficient of all and performs in the range of milliseconds for even the largest models considered in this study. Unfortunately, it is unsuitable when DEX models form full hierarchies; in this case, it generally finds just a subset of possible solutions.
- *TopDown* is a kind of a trade-off between the other two. It is generally slower than *BottomUp*, but can still run in the range of seconds for relatively large models (consisting of about 40 attributes). *TopDown* has no issues with full hierarchies and always generates all solutions.

The main reason for *BottomUp*'s efficiency is its ability to construct solutions by considering only value assignments that differ *the least* from the analyzed alternative. Since some solutions in hierarchies cannot be constructed by combining the best lower-level solutions, they are missed by the algorithm.

The main reason for *TopDown*'s efficiency is its ability to:

- consider only decision rules that map to the target value of the root attribute, and target values of lower-level attributes, which are determined dynamically in the process,
- quickly generate the first solution, which provides an anchor for comparing further candidate solutions, and
- disregarding (potentially large) parts of the search space when current candidate solutions become too distant from the best solution generated thus far.

Currently, target analysis is implemented in software DEXi and DEXiWin as an application-ready variation of *TopDown*. Since it is applicable for general models, including hierarchies, and has acceptable execution times, it seems a viable solution for practice. So far, we have used it in several decision-support projects, and found it useful and "fit-for-the-purpose".

For the future, given the excellent performance of *BottomUp* in this study, it might be a good idea to consider using this algorithm in the software whenever there are no linked attributes in the model. As any practical implementation has to consider requirements and limitations discussed in section 6, we still have to determine how these can be fulfilled by *BottomUp*. Another interesting question is how to make *BottomUp* applicable to hierarchies, too. So far, we were not able to find a "clean" solution. We know it is possible to combine *BottomUp* with *TopDown* or some other combinatorial search algorithm on internal attributes that contain linked subtrees, but this severely complicates the approach and makes *BottomUp* less efficient.

8 References

Bohanec, M. (2020): DEXi: Program for Multi-Attribute Decision Making, User's Manual, Version 5.04. IJS Report DP-13100, Jožef Stefan Institute, Ljubljana.
https://kt.ijs.si/MarkoBohanec/pub/DEXiManual504.pdf

Bohanec, M.: DEX (Decision EXpert) (2022): A qualitative hierarchical multi-criteria method. Multiple Criteria Decision Making (ed. Kulkarni, A.J.), Studies in Systems, Decision and Control 407, Singapore: Springer, 39–78. doi: 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

Buede, D.M. (2013): Decision Making and Decision Analysis. In: Gass, S.I., Fu, M.C. (eds) *Encyclopedia of Operations Research and Management Science*. Boston: Springer. https://doi.org/10.1007/978-1-4419-1153-7_217.

Guidotti, R. (2022): Counterfactual explanations and how to find them: literature review and benchmarking. *Data Mining and Knowledge Discovery* 38(5):1–55. http://dx.doi.org/10.1007/s10618-022-00831-6.

Gjoreski, M., Kuzmanovski, V., Bohanec, M. (2020): Generating alternatives for DEX models using Bayesian optimization. *Proceedings of the 23rd International Conference Information Society IS 2020*, Volume A: Slovenian Conference on Artificial Intelligence, Ljubljana: Institut Jožef Stefan, 23–26, 2020. https://kt.ijs.si/MarkoBohanec/pub/2020_IS_OptGen.pdf.

Gjoreski, M., Kuzmanovski, V., Bohanec, M. (2022): BAG-DSM: A method for generating alternatives for hierarchical multi-attribute decision models using Bayesian optimization. *Algorithms* 15(6) 197: 1–22, 2022. https://doi.org/10.3390/a15060197.

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.

Humi, M. (2017): *Introduction to Mathematical Modeling*. New York: Chapman and Hall/CRC. ISBN 9781315370309, https://doi.org/10.1201/9781315370309.

Shiflet, A.B., Shiflet, G.W. (2014): *Mathematical Modeling: A Comprehensive Introduction*. Second Edition. Cambridge University Press. Princeton: Princeton University Press, ISBN 978-0691160719.

Trdin, N., Bohanec, M. (2018): Extending the multi-criteria decision making method DEX with numeric attributes, value distributions and relational models. *Central European Journal of Operations Research*, 1–41, 2018. https://doi.org/10.1007/s10100-017-0468-9.

Appendix 1: Example *TopDown* Operation

In order to illustrate the *TopDown* algorithm (section 4.3, Algorithm 4), an annotated trace of its operation is shown below.

Inputs

Model: AvrMinMax_Links (Figure 4):

- aggregate attributes: [*Y*, *A*, *B*]
- basic attributes: [*A1*, *C*, *B1*]
- topological order of attributes: [*Y*, *A*, *B*, *A1*, *C*, *B1*]

Alternative A_0: assignment $\boldsymbol{z_0} = [1,1,1]$.

Target value: *Y* = 1.

Difference measure: $\mathrm{unidiff}^+$

Trace

Attribute index and name	Current assignment [Y, A, B, A1, C, B1]	Operation/Result	Solutions
1 Y	[1,0,0,0,0,0]	diff=0, can continue	none
		checking rule [1,1]→1 of Y	
	[1,1,1,0,0,0]	setting succeeds	
step in to 2 A	[1,1,1,0,0,0]	diff=0, can continue	
		checking rule [1,1]→1 of A	
	[1,1,1,1,1,0]	setting succeeds	
step in to 3 B	[1,1,1,1,1,0]	diff=0, can continue	
		checking rule [1,1]→1 of B	
	[1,1,1,1,1,1]	setting succeeds	
step in to 4 A1	[1,1,1,1,1,1]	[1,1,1] is original assignment, not recorded	
step back to 2 A	[1,1,1,0,0,0]	checking rule [1,2]→1 of A	
	[1,1,1,1,2,0]	setting succeeds	
step in to 3 B	[1,1,1,1,2,0]	diff=1, can continue	
		checking rule [1,1]→1 of B	
		setting fails ($2 \neq 1$)	
step back to 2 A	[1,1,1,0,0,0]	checking rule [1,3]→1 of A	
	[1,1,1,1,3,0]	setting succeeds	
step in to 3 B	[1,1,1,1,3,0]	diff=2, can continue	
		checking rule [1,1]→1 of B	
		setting fails ($3 \neq 1$)	
step back to 2 A	[1,1,1,0,0,0]	checking rule [2,1]→1 of A	
	[1,1,1,2,1,0]	setting succeeds	
step in to 3 B	[1,1,1,2,1,0]	diff=1, can continue	
		checking rule [1,1]→1 of B	
	[1,1,1,2,1,1]	setting succeeds	
step in to 4 A1	[1,1,1,2,1,1]	[2,1,1] recorded	diff=1: [2,1,1]
step back to 2 A	[1,1,1,0,0,0]	checking rule [3,1]→1 of A	
	[1,1,1,3,1,0]	setting succeeds	
step in to 3 B	[1,1,1,3,1,0]	diff=2, cannot continue	
step back to 1 Y		checking rule [1,2]→2 of Y	
	[1,1,2,0,0,0]	setting succeeds	
	[1,1,2,0,0,0]		
step in to 2 A	[1,1,2,0,0,0]	diff=0, can continue	
		checking rule [1,1]→1 of A	
	[1,1,2,1,1,0]	setting succeeds	
step in to 3 B	[1,1,2,1,1,0]	diff=0, can continue	

		checking rule [1,2]→1 of B	
	[1,1,2,1,1,2]	setting succeeds	
step in to 4 A1	[1,1,2,1,1,2]	[1,1,2] recorded	diff=1: [1,1,2],[2,1,1]
step back to 3 B	[1,1,2,1,1,0]	checking rule [2,1]→2 of B	
		setting fails (1 ≠ 2)	
		checking rule [2,1]→2 of B	
		setting fails (1 ≠ 2)	
step back to 2 A	[1,1,2,0,0,0]	checking rule [1,2]→1 of A	
	[1,1,2,1,2,0]	setting succeeds	
step in to 3 B	[1,1,2,1,2,0]	diff=1, can continue	
		checking rule [1,2]→2 of B	
		setting fails (2 ≠ 1)	
		checking rule [2,1]→2 of B	
	[1,1,2,1,2,1]	setting succeeds	
step in to 4 A1	[1,1,2,1,2,1]	[1,2,1] recorded	diff=1: [1,1,2],[1,2,1],[2,1,1]
step back to 3 B	[1,1,2,1,1,0]	checking rule [2,2]→2 of B	
	[1,1,2,1,2,2]	setting succeeds	
step in to 4 A1	[1,1,2,1,2,2]	[1,2,2] not recorded, diff=2	
step back to 2 A	[1,1,2,0,0,0]	checking rule [1,3]→1 of A	
	[1,1,2,1,3,0]	setting succeeds	
step in to 3 B	[1,1,2,1,3,0]	diff=2, cannot continue	
step back to 2 A	[1,1,2,0,0,0]	checking rule [2,1]→1 of A	
	[1,1,2,2,1,0]	setting succeeds	
step in to 3 B	[1,1,2,2,1,0]	diff=1, can continue	
		checking rule [1,2]→2 of B	
	[1,1,2,2,1,2]	setting succeeds	
step in to 4 A1	[1,1,2,2,1,2]	[2,1,2] not recorded, diff=2	
		checking rule [2,1]→2 of B	
		setting fails (1 ≠ 2)	
		checking rule [2,2]→2 of B	
		setting fails (1 ≠ 2)	
step back to 2 A	[1,1,2,0,0,0]	checking rule [3,1]→1 of A	
	[1,1,2,3,1,0]	setting succeeds	
step in to 3 B	[1,1,2,3,1,0]	diff=2, cannot continue	
	[1,1,2,0,0,0]		
step back to 2 A		checking rule [3,1]→1 of A	
	[1,1,2,3,1,0]	setting succeeds	
step in to 3 B	[1,1,2,3,1,0]	diff=2, cannot continue	
step back to 1 Y	[1,0,0,0,0,0]	checking rule [2,1]→1 of Y	
	[1,2,1,0,0,0]	setting succeeds	
step in to 2 A	[1,2,1,0,0,0]	diff=0, can continue	
		checking rule [2,2]→2 of A	
	[1,2,1,2,2,0]	setting succeeds	
step in to 3 B	[1,2,1,2,2,0]	diff=2, cannot continue	
step back to 2 A	[1,2,1,0,0,0]	checking rule [2,3]→2 of A	
	[1,2,1,2,3,0]	setting succeeds	
[1,2,1,2,3,0]	[1,2,1,2,3,0]	diff=3, cannot continue	
step back to 2 A	[1,2,1,0,0,0]	checking rule [3,2]→2 of A	
	[1,2,1,3,2,0]	setting succeeds	
	[1,2,1,3,2,0]	diff=3, cannot continue	

The algorithm finds the final solution composed of three assignments to [A1,C,B1]:

[1,1,2], [1,2,1], [2,1,1],

that differ by diff=1 from $\boldsymbol{z}_0$=[1,1,1].

Appendix 2: Experimental Evaluation: Models and Results

Table 4: Models used in the study and their characteristics.

Model name	**Levels**	**Number of Attributes**				**Alternatives**
		Basic	Aggregate	Linked	All	
Car (Figure 2)	3	6	4	0	10	all 972
AvrMinMax_NoLink (Figure 3)	2	4	3	0	7	all 81
AvrMinMax_Link (Figure 4)	2	3	3	1	7	all 27
LinkedAttributesTest	3	3	4	2	9	all 27
LinkedBoundsTest	2	2	3	2	7	all 9
LinkedBoundsTest3	2	2	4	4	10	all 9
Problem	2	4	3	0	7	all 81
Problem2	2	4	3	0	7	all 81
AgriPruned	2	11	3	0	14	random 100
AgriFood	2	15	4	0	19	random 100
AgriFoodChainIntegrated	2	15	9	15	39	random 100
bm_lvl_3_normal_0_wo_links	2	9	4	0	13	predefined 100
bm_lvl_3_normal_0_wo_links	2	9	4	0	13	predefined 100
bm_lvl_3_normal_0_wo_links	2	9	4	0	13	predefined 100
bm_lvl_3_normal_0_w_links	2	8	4	1	13	predefined 100
bm_lvl_3_normal_1_wo_links	2	9	4	0	13	predefined 100
bm_lvl_3_normal_1_w_links	2	8	4	1	13	predefined 100
bm_lvl_3_normal_2_wo_links	2	9	4	0	13	predefined 100
bm_lvl_3_normal_2_w_links	2	8	4	1	13	predefined 100
bm_lvl_3_skewed_0_wo_links	2	9	4	0	13	predefined 100
bm_lvl_3_skewed_0_w_links	2	8	4	1	13	predefined 100
bm_lvl_3_skewed_1_wo_links	2	9	4	0	13	predefined 100
bm_lvl_3_skewed_1_w_links	2	8	4	1	13	predefined 100
bm_lvl_3_skewed_2_wo_links	2	9	4	0	13	predefined 100
bm_lvl_3_skewed_2_w_links	2	8	4	1	13	predefined 100
bm_lvl_3_uniform_0_wo_links	2	9	4	0	13	predefined 100
bm_lvl_3_uniform_0_w_links	2	8	4	1	13	predefined 100
bm_lvl_4_normal_1_wo_links	3	20	11	0	31	predefined 100
bm_lvl_4_normal_1_w_links	3	19	11	1	31	predefined 100
bm_lvl_4_normal_2_wo_links	3	20	11	0	31	predefined 100
bm_lvl_4_normal_2_w_links	3	19	11	1	31	predefined 100
bm_lvl_4_skewed_0_wo_links	3	20	11	0	31	predefined 100
bm_lvl_4_skewed_0_w_links	3	19	11	1	31	predefined 100
bm_lvl_4_skewed_1_wo_links	3	20	11	0	31	predefined 100
bm_lvl_4_skewed_1_w_links	3	19	11	1	31	predefined 100
bm_lvl_4_skewed_2_wo_links	3	20	11	0	31	predefined 100
bm_lvl_4_skewed_2_w_links	3	19	11	1	31	predefined 100
bm_lvl_4_uniform_0_wo_links	3	20	11	0	31	predefined 100
bm_lvl_4_uniform_0_w_links	3	19	11	1	31	predefined 100
bm_lvl_5_normal_0_wo_links	4	39	26	0	65	predefined 100
bm_lvl_5_normal_0_wo_links	4	39	26	0	65	predefined 100
bm_lvl_5_normal_0_w_links	4	38	26	1	65	predefined 100
bm_lvl_5_normal_1_wo_links	4	39	26	0	65	predefined 100
bm_lvl_5_normal_1_w_links	4	38	26	1	65	predefined 100
bm_lvl_5_normal_2_wo_links	4	39	26	0	65	predefined 100
bm_lvl_5_normal_2_w_links	4	38	26	1	65	predefined 100
bm_lvl_5_skewed_0_wo_links	4	39	26	0	65	predefined 100
bm_lvl_5_skewed_0_w_links	4	38	26	1	65	predefined 100
bm_lvl_5_skewed_1_wo_links	4	39	26	0	65	predefined 100
bm_lvl_5_skewed_1_w_links	4	38	26	1	65	predefined 100
bm_lvl_5_skewed_2_wo_links	4	39	26	0	65	predefined 100
bm_lvl_5_skewed_2_w_links	4	38	26	1	65	predefined 100
bm_lvl_5_uniform_0_wo_links	4	39	26	0	65	predefined 100
bm_lvl_5_uniform_0_w_links	4	38	26	1	65	predefined 100
Total: 54						
min	2	2	3	0	7	
max	4	39	26	15	65	

Table 5: Execution times in milliseconds per alternative. “NA” indicates “not attempted”. Cells containing “0” indicate execution times so small that they were not even detected.

Model name	**Algorithm**		
	Exhaustive	**TopDown**	**BottomUp**
Car	2.27	0.13	0.03
AvrMinMax_NoLink	0	0	0
AvrMinMax_Link	0	0.59	0
LinkedAttributesTest	0	0	0
LinkedBoundsTest	0	0	0
LinkedBoundsTest3	0	0	0
Problem	0	0.20	0.20
Problem2	0	0	0.20
AgriPruned	462.04	14.54	3.12
AgriFood	NA	885.66	91.66
AgriFoodChainIntegrated	NA	833.22	78.66
bm_lvl_3_normal_0_wo_links	51.64	0.94	0.31
bm_lvl_3_normal_0_wo_links	50.75	0.94	0.16
bm_lvl_3_normal_0_wo_links	50.59	0.94	0.31
bm_lvl_3_normal_0_w_links	17.34	0.62	0.16
bm_lvl_3_normal_1_wo_links	51.79	0.8	0.16
bm_lvl_3_normal_1_w_links	16.87	0.47	0.16
bm_lvl_3_normal_2_wo_links	50.87	0.94	0.31
bm_lvl_3_normal_2_w_links	15.94	0.54	0
bm_lvl_3_skewed_0_wo_links	51.01	0.47	0.31
bm_lvl_3_skewed_0_w_links	16.87	0.31	0.2
bm_lvl_3_skewed_1_wo_links	51.35	0.62	0.16
bm_lvl_3_skewed_1_w_links	17.06	0.47	0
bm_lvl_3_skewed_2_wo_links	51.01	0.78	0.16
bm_lvl_3_skewed_2_w_links	16.87	0.31	0
bm_lvl_3_uniform_0_wo_links	52.57	0.94	0.31
bm_lvl_3_uniform_0_w_links	16.7	0.62	0.16
bm_lvl_4_normal_1_wo_links	NA	208.87	0.47
bm_lvl_4_normal_1_w_links	NA	176.85	0.31
bm_lvl_4_normal_2_wo_links	NA	326.01	0.31
bm_lvl_4_normal_2_w_links	NA	429.63	0.94
bm_lvl_4_skewed_0_wo_links	NA	97.56	0.46
bm_lvl_4_skewed_0_w_links	NA	64.93	0.31
bm_lvl_4_skewed_1_wo_links	NA	101.42	0.16
bm_lvl_4_skewed_1_w_links	NA	81.34	0
bm_lvl_4_skewed_2_wo_links	NA	701.55	0.47
bm_lvl_4_skewed_2_w_links	NA	7719.46	0.46
bm_lvl_4_uniform_0_wo_links	NA	1824.95	2.67
bm_lvl_4_uniform_0_w_links	NA	1478.37	2.03
bm_lvl_5_normal_0_wo_links	NA	NA	1.25
bm_lvl_5_normal_0_wo_links	NA	NA	1.41
bm_lvl_5_normal_0_w_links	NA	NA	1.09
bm_lvl_5_normal_1_wo_links	NA	NA	0.31
bm_lvl_5_normal_1_w_links	NA	NA	0.31
bm_lvl_5_normal_2_wo_links	NA	NA	1.95
bm_lvl_5_normal_2_w_links	NA	NA	0.79
bm_lvl_5_skewed_0_wo_links	NA	NA	0.47
bm_lvl_5_skewed_0_w_links	NA	NA	0.78
bm_lvl_5_skewed_1_wo_links	NA	NA	0.64
bm_lvl_5_skewed_1_w_links	NA	NA	0.62
bm_lvl_5_skewed_2_wo_links	NA	NA	0.31
bm_lvl_5_skewed_2_w_links	NA	NA	0.62
bm_lvl_5_uniform_0_wo_links	NA	NA	11.56
bm_lvl_5_uniform_0_w_links	NA	NA	7.50

Appendix 3: 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)	$Z = \{x \in X: S(x) = \emptyset\}$
Concatenation of ordered sets	$X = Y \circ Z$
Scales	$D = \{\text{scale}\, x_i \mid x_i \in X\}$
Scale	$\text{scale}\, x = [v_{x,1}, v_{x,2}, \dots, v_{x,m_x}] = [1, 2, \dots, m_x]$
Scale order	$\text{order}\, x \in \{\text{unordered, ascending, descending}\}$
Decision space	$\text{space}\, K = \text{scale}\, x_1 \times \text{scale}\, x_2 \times \dots \times \text{scale}\, x_k,\ k = \lvert K \rvert$
Descendant function	$S: X \rightarrow 2^x$
Parents of x	$P(x) = \{p \in X: x \in S(p)\}$
Aggregation functions	$F = \{f_x, x \in Y\}$
… associated with $x \in Y$	$f_x: \text{space}\, S(x) \rightarrow \text{scale}\, x$
Decision table for $x \in Y$	$T_x = \{r_i, i = 1, 2, \dots, \lvert \text{space}\, S(x) \rvert\}$
Decision rule	$r \in T_x: \mathbf{a} \in \text{space}\, S(x) \rightarrow v \in \text{scale}\, x$
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	$Y(A_i) = [a_{x,i} \in A_i, \forall x \in Y]$
Overall evaluation of alternative A_i	$E(A_i) = a_{1,i}$
Given alternative	A_0
Initial assignment	$\boldsymbol{z}_0 \in \text{space}\, Z$
i-th element of vector (assignment) $\boldsymbol{a}$	a_i
Concatenation of two assignments $\boldsymbol{a}$ and $\boldsymbol{b}$	$\boldsymbol{a} \circ \boldsymbol{b} = [a_1, a_2, \dots, b_1, b_2, \dots]$
Extracted $\boldsymbol{b}$ elements from $\boldsymbol{a}$	$\boldsymbol{a}[\boldsymbol{b}] = [a_{b_1}, a_{b_2}, \dots]$
Difference function	$\text{diff}_K(A_1, A_2) = \sum_{i=1}^{\lvert K \rvert} \text{diff}(x_i, a_{i,1}, a_{i,2})$
… associated with attribute x	$\text{diff}(x, a_1, a_2), x \in X, a_{1,}, a_2 \in \text{scale}\, x$
Recorded solutions in R_x corresponding to a	$R_x[a], x \in X, a \in \text{scale}\, x$