

POZNAN UNIVERSITY OF TECHNOLOGY  
FACULTY OF COMPUTING  
INSTITUTE OF COMPUTING SCIENCE

BACHELOR'S THESIS

# MODULAR DECISION SUPPORT SYSTEM FOR THE PROMETHEE METHODS

*Authors:*

Magdalena DZIECIELSKA  
Sebastian PAWLAK  
Mateusz SARBINOWSKI  
Maciej UNIEJEWSKI

*Supervisor:*

Miłosz KADZIŃSKI, PhD

Poznań, 2017



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Description of methods implemented as algorithmic modules on the diviz platform</b>	<b>4</b>
2.1	Criteria Weights . . . . .	4
2.1.1	Surrogate Weighting Techniques . . . . .	4
2.1.2	Simos-Roy-Figueira (SRF) . . . . .	5
2.2	Preference Indices . . . . .	7
2.2.1	Preference . . . . .	8
2.2.2	Preference with Reinforced Preference . . . . .	11
2.2.3	Preference with interactions between criteria . . . . .	13
2.2.4	Discordance . . . . .	15
2.2.5	Veto . . . . .	17
2.2.6	Overall preference index . . . . .	18
2.3	Outranking Flows . . . . .	18
2.4	Profiles of the Alternatives . . . . .	20
2.5	Ranking Problems . . . . .	21
2.5.1	Promethee I . . . . .	21
2.5.2	Promethee II . . . . .	22
2.5.3	Promethee III . . . . .	23
2.5.4	Net Flow Score . . . . .	24
2.5.5	GDSS Ranking . . . . .	25
2.6	Sorting Problems . . . . .	26
2.6.1	PromSort . . . . .	27
2.6.2	Promethee Tri . . . . .	29
2.6.3	FlowSort based on Promethee I . . . . .	31
2.6.4	FlowSort based on Promethee II . . . . .	34
2.6.5	FlowSort GDSS . . . . .	35
2.6.6	Group Class Acceptabilities . . . . .	40
2.7	Choice Problem . . . . .	41
2.8	Clustering . . . . .	42
2.8.1	Ordered clustering . . . . .	43
2.8.2	Promethee II Ordered Clustering . . . . .	44
2.8.3	Promethee Cluster . . . . .	44
2.9	Visualization . . . . .	46
2.9.1	Graphical class assignment . . . . .	46
2.9.2	Latex table of class assignment . . . . .	47
<b>3</b>	<b>Construct Your Own Promethee Method in diviz</b>	<b>48</b>
3.1	diviz . . . . .	48
3.2	Modules implementation . . . . .	48
3.3	Workflow Design . . . . .	48

**4 Illustrative Case Studies** **50**  
4.1 Multiple criteria ranking and choice - example 1 . . . . . 50  
4.2 Multiple criteria sorting and clustering - example 2 . . . . . 58

**5 Conclusions** **66**

**6 Acknowledgments** **67**

**References** **68**

# 1 Introduction

The analysis of real-world problems requires consideration of multiple conflicting points of view that affect a decision. Nowadays, practically all decision situations involve economic, environmental, and social criteria which describe diverse consequences of the existing alternatives. A decision process needs to explore the conflicting character of criteria by taking into account preferences of the Decision Maker (DM). To this end, the field of Multiple Criteria Decision Analysis (MCDA) offers some tools that support such process.

PROMETHEE is one of the prevailing families of MCDA methods [5]. Its usefulness comes from the fact of combining advantages of various approaches. On one hand, PROMETHEE uses an outranking-based preference model, thus, deriving its results from comparing the alternatives pairwise. When implementing these comparisons, the method tolerates imperfect knowledge of data, operates on heterogeneous scales (thus, not demanding any prior normalization), and is capable of representing the situation of weak preference. Moreover, to compare the alternatives consistently with the DM's value system, PROMETHEE requires her/him to provide some intuitive parameters with a clear physical meaning. These concern relative importance or preference function associated with each criterion.

On the other hand, PROMETHEE – similarly to value- or utility-based methods – takes advantage of the comprehensive scores that are easier to understand for the DM. These scores, called flows, can be perceived as measures of desirability indicating how each alternative compares against all the remaining ones. Furthermore, PROMETHEE implements an analogy to voting procedures by taking into account the reasons for and against a preference of one alternative over the others, which enhances its intuitiveness and interpretability. Still, the comprehensive scores provided by PROMETHEE can be used in a way that admits incomparability in the comparison of a pair of alternatives, which, in turn, increases its flexibility in representing the DM's preferences.

Being simple, clear, and stable method, PROMETHEE has been applied in a variety of complex decision problems [19, 24, 12]. Its main application areas involve environmental and water management, finance, chemistry, logistics, transportation, manufacturing, and energy sector [2]. The exemplary real-world studies concern tree-harvesting in Malaysia [19], business and conferring financial support for firms [24], or scoring hospital services [12].

PROMETHEE has been originally designed to deal with the ranking problems (see, e.g., PROMETHEE I, II and III), and subsequently adapted to choice (e.g., PROMETHEE V), sorting (e.g., PromSort, PROMETHEE TRI, or FlowSort), and clustering (e.g., Promethee II Ordered Clustering or PROMETHEE Cluster). All PROMETHEE methods can be perceived as sequences of the elementary well-defined steps which implement in a specific way some general framework that is peculiar to this approach. Within this framework, all ordered pairs of objects (alternatives and/or class profiles) are first compared pairwise. Then, the outcomes of such comparisons are aggregated into flows, which offer a more comprehensive perspective on the desirability of each object. Finally, the flows are used to deliver recommendation in function of the specific problem to solve.

Obviously, each PROMETHEE method is distinguished by its unique exploitation phase which decides upon the rank of each alternative, its presence in the subset of the most preferred options, its assignment into one of pre-defined and ordered classes, or separation of the alternatives into different groups. Instead, the phase of construction of a preference relation in PROMETHEE has been implemented in the same way in all PROMETHEE methods. It consists in computing a degree indicating a joint strength of all preferentially independent criteria for which one object is preferred to the other. Conversely, in the ELECTRE methods, the phase of construction of an outranking relation can be conducted in various ways, and it seems reasonable to adapt some of these ideas to the context of PROMETHEE.

Overall, while phases of construction and exploitation of the preference structure are independent, a new PROMETHEE approach can be obtained each time when coupling together specific implementations of these two stages. Unfortunately, such flexibility is not offered neither by the existing approaches nor by the available software. Indeed, the existing PROMETHEE tools poorly expose the sequential character of the method, sticking to a rather univocal implementation of the elementary procedures. Moreover, many PROMETHEE approaches mentioned in the previous paragraph are not available in any software package. As a result, their use for practical decision analysis is restrained. In the same spirit, the selection of methods offered by the existing software pieces is rather scarce (see, e.g., Promethee MD [16] which offers Promethee I and II). Finally, the commercial software packages which implement a few PROMETHEE methods, such as Visual Promethee [23] or D-sight [17], cost hundreds of euro. Unlikely, the functionality of their respective free trial versions is seriously limited.

To address all aforementioned concerns, in this contribute to the development of *diviz* [25] with a focus on the development of PROMETHEE methods. In general, the platform postulates implementation of each elementary computation or visualization procedure as a separate software module. These modules can be chained in *diviz* to build complex MCDA algorithmic workflows. In this way, the users can rebuild the existing approaches, but also construct new ones. This way of proceeding increases the flexibility in adjusting decision aiding methods to the specific problem. It also enhances their transparency by exhibiting all partial results and encouraging users to analyze them. Moreover, *diviz* is an open-source tool which not only makes its use cost-free, but also supports potential contributors to enriching a variety of available algorithmic modules by implementing some new procedures by analogy to the existing ones.

This thesis contributes to the development of PROMETHEE in various ways. From the methodological viewpoint, we postulate greater flexibility in constructing PROMETHEE methods so that they can be appropriately adjusted to the characteristic of a specific decision problem. In this regard, we show how to combine different implementations of the construction and exploitation phases within a common methodological framework. At this point, we also significantly enrich a variety of algorithmic procedures that can be applied in the context of PROMETHEE. When it comes to the software development, we show how the postulate of flexibility can be realized in practice. This has been achieved by implementing several PROMETHEE-based components and making them available at *diviz*. These modules are highly parameterized and designed to interoperate, which makes them suitable for designing advanced MCDA methods. In what

follows, we discuss the procedures we account for at different stages of PROMETHEE.

At the stage of construction of a preference structure, we consider a variety of procedures for computing comprehensive preference degrees. As far as criteria weights are concerned, we take advantage of a few surrogate weighting procedures (e.g., Rank Order Centroid or the revised Simos procedure). When it comes to preference degrees, apart from considering an original proposal postulated in PROMETHEE that incorporates six pre-defined shapes of the preference functions, we additionally extend PROMETHEE for dealing with the effect of reinforced preference, discordance (veto) against preference, and interactions between criteria. Although these have been originally proposed in the context of ELECTRE, we adapt them to PROMETHEE as they vastly increase the flexibility of preference modeling, making the methodology more suitable for various decision contexts. Then, the preference degrees can be aggregated to derive positive, negative, and comprehensive flows. The diviz modules for computing both preference degrees and flows are universal in a sense of admitting the comparison of each alternative either against all remaining ones or with boundary or characteristic class profiles. In this way, their results can be subsequently exploited in function of different problems types.

At the stage of exploitation of a preference structure in PROMETHEE, we consider the following methods:

- PROMETHEE I, II and III that exploit the preference flows, and the Net Flow Scores procedures that exploit the preference degrees to rank the alternatives from the best to the worst;
- PROMETHEE V to select a subset of the most preferred options while taking into account some additional constraints (e.g., concerning the available budget);
- FlowSort, PromSort, and Promethee TRI that assign the alternatives to some pre-defined and ordered classes by comparing them against boundary or characteristic profiles for the a
- Promethee II Ordered Clustering, Promethee Cluster, and other clustering algorithms to divide a set of alternatives into a number of groups;
- Promethee Group, FlowSort GDSS, and procedures for deriving some group class acceptability indices that can be used for group decision making.

Additionally, we have implemented some visualization components that can be used to present the recommendation in a comprehensible way. Overall, taking advantage of the constructed modules, the user may construct her/his own PROMETHEE method in a few minutes without any programming skills. For this purpose, it is just enough to combine the components in one of several hundred admissible ways.

When conducting the thesis, the tasks were distributed in the following way:

- Magdalena Dziecielska was responsible for creating modules M11 - M16 and M22,
- Sebastian Pawlak prepared modules M1 - M10,

- Mateusz Sarbinowski have done modules M23 - M26,
- Maciej Uniejewski prepared modules M17 - M21 and M27 - M28.

Moreover, all members of the group contributed equally to the text of this thesis.

## 2 Description of methods implemented as algorithmic modules on the diviz platform

### 2.1 Criteria Weights

Defining criteria weights is one of the most important MCDA processes. In this section, we review different techniques for deriving criteria weights. We discuss two modules we have implemented. To facilitate further reference to these modules, we denote them by M1 and M2.

#### 2.1.1 Surrogate Weighting Techniques

In surrogate weighting techniques, the list of weights has certain characteristics [27]. All weights are constant values and are greater than or equal to zero. Additionally, the sum of all criteria weights is equal to one.

In surrogate weighting techniques we can note several methods for deriving weights of criteria:

- Equal Weights

All weights have the same value, so all criteria are deemed as equally important. This rule is presented by Equation 1:

$$w_i(EW) = \frac{1}{n}, \quad (1)$$

where  $n$  is the number of criteria.

- Rank-Sum

This rule takes into account the ranking of criteria. The first position indicates the most important criterion. Less important criteria have further positions. This technique is expressed by Equation 2:

$$w_i(RS) = \frac{2(n+1-i)}{n(n+1)} \quad (2)$$

where  $i$  is the criterion's position in the ranking provided by the DM

- Reciprocal of the Ranks



This technique has an other approach in using criteria ranking. It divides each reciprocal of rank by the sum of these reciprocals for all criteria. This rule is expressed by Equation 3:

$$w_i(RR) = \frac{\frac{1}{i}}{\sum_{j=1}^n \frac{1}{j}}. \quad (3)$$

- Rank-Order Centroid

The weights in this method reflect the centroid of the simplex defined by ranking of the criteria [28]. It is expressed by Equation 4:

$$w_i(ROC) = \frac{1}{n} \sum_{j=i}^n \frac{1}{j} \quad (4)$$

**Module M1. SurrogateWeights.** This module computes weights of criteria using surrogate weighting techniques  $w_i()$  ( $out_1$ ). Its structure is presented in Figure 2.1. It requires the user to specify criteria ranking ( $in_1$ ). In this ranking, each criterion is associated with a unique position. The same position for more than one criterion is not permitted. The first position in the ranking indicates the most important criterion. The user has to choose a method to calculate weights ( $param_1$ ). All techniques mentioned in Section 2.1.1 are supported.

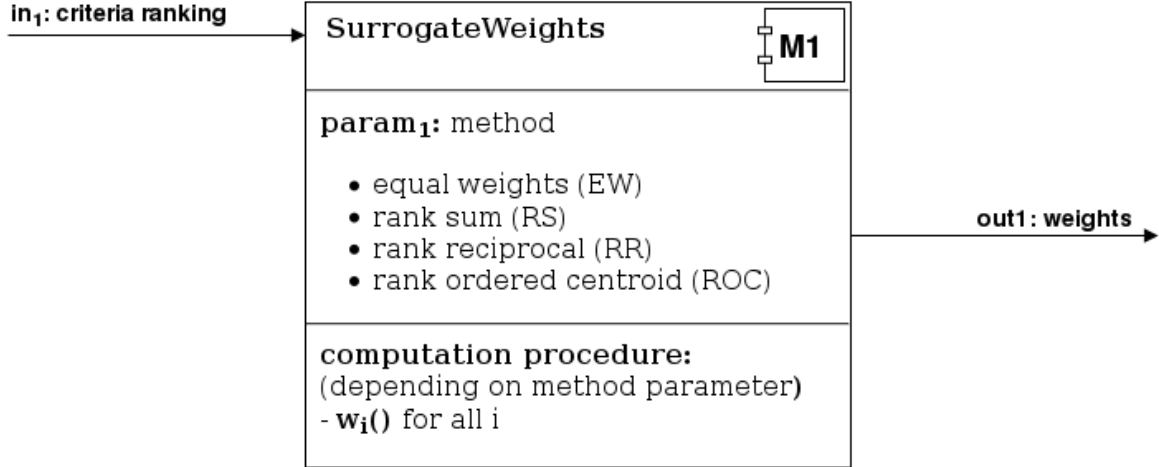


Figure 2.1: Structure of module M1 which computes weights of criteria using surrogate weights techniques .

### 2.1.2 Simos-Roy-Figueira (SRF)

The revised Simos' procedure proposed by Figueira and Roy [15] is much more flexible than methods mentioned in Section 2.1.1. In this approach the DM has  $m$  cards with the criteria names, and a set of white cards. (S)he must sort out cards in order of importance. Criteria with the same importance should be grouped together. The DM

can control the intensity of preference between the subsets of criteria by inserting white cards between them. This method requires the user additionally to specify the ratio between the weights of the first and last criteria in ranking.

The algorithm of SRF consists of several steps:

- Calculating non-normalized weights

To show this step we use the notation proposed by Corrente et al. [7] (see Equation 5):

$$w'_j = 1 + \frac{(z - 1)[l(j) - 1 + \sum_{s=1}^{l(j)-1} e_s]}{v - 1 + \sum_{s=1}^{v-1} e_s}, \quad (5)$$

where:

$j$  is the criterion's index,

$z$  is the ratio between weights of first (the least important) and last (the most important) criteria in the ranking,

$l(j)$  is the position in ranking,

$v$  is the position in ranking of the most important criterion,

$e_s$  is the number of white cards between subsets on position  $s$  and  $s + 1$ .

- Calculating normalized weights

The normalized weight is expressed by Equation 6 [7]:

$$w_j = \frac{w'_j}{\sum_{k=1}^m w'_k}, \quad (6)$$

- Rounding off the numerical values

Full algorithm SRF assumes that weights of criteria have numerical values rounded to specified number of decimal places (0-2) [15]. Moreover, the sum of the weights is equal to 100. To meet these demands, SRF splits the set of criteria into two subsets. The weights are multiplied by 100. After that, weights in one subset are rounded upwards, while weights in the second subset are rounded downwards.

**Module M2. SRFWeights.** This module computes weights of criteria  $w_j()$  ( $out_1$ ) using the revised Simos (or Simos-Roy-Figueira; SRF) method. These weights are normalized, but in this case the sum of all values is equal to 100. The structure of this module is presented in Figure 2.2.

The SRF module requires the user to specify criteria ranking ( $in_1$ ). This input file must meet certain rules:

- Order  
The first position in ranking indicates the least important criterion, whereas the last position in ranking indicates the most important criterion.
- Grouping  
The SRF method allows to group criteria with the same importance. To define group of criteria with the same importance in “criteria ranking” file, the user has to assign criteria the same position in ranking.
- White Cards  
Blank position in ranking (position in the ranking has not assigned criterion) is reflection of “white card” in the SRF method.
- Weight Ratio  
The user has to specify the ratio ( $param_1$ ) between weights of the most and the least important criteria. This value is required by the SRF procedure. It is the real number denoting the ratio  $z$  from Equation 5.
- Precision  
The user must specify the number of decimal places ( $param_2$ ) for the weights calculated by the SRFWeights module.

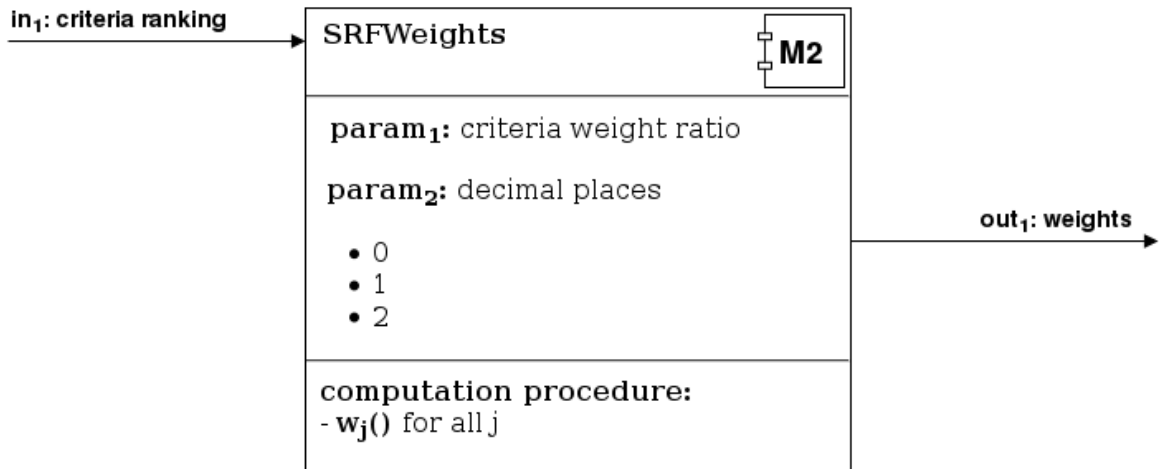


Figure 2.2: Structure of module M2 which computes weights of criteria using SRF method.

## 2.2 Preference Indices

In this section we discuss different methods for calculating preference indices. In PROMETHEE, such calculation is based on pairwise comparisons. First step is to calculate deviation between evaluations of two alternatives (or alternative and profile)

on each criterion. We present six functions which are used to compute preference index on some criterion (generalised criterion). Finally, we discuss how this partial preferences are aggregated into a global value. We consider here also an overall preference index which accounts for veto, preference index and discordance index.

Simultaneously, we present the functionality of several modules we have created. They are available on diviz platform. In most cases, it is possible to control the processing of modules by its parameters. The modules returns a single or multiple outputs, which can be used as input to others. Each presented module has its own number to simple identification (M3 - M8)

### 2.2.1 Preference

**Deviations.** As it was mentioned earlier, the preference index in PROMETHEE is based on pairwise comparisons [4]. To compare two alternatives on a specified criterion the deviation in evaluations are considered. It is expressed in Equation 7.

$$d_j(a, b) = g_j(a) - g_j(b) \quad (7)$$

where:

$a, b$  are the alternatives

$d_j(a, b)$  is the deviation between evaluations on criterion  $j$

$g_j(a), g_j(b)$  are evaluations of alternatives on criterion  $j$

This deviation is used to calculate preference of  $a$  over  $b$  when criterion  $j$  is maximized. If criterion is minimized the sign of deviation should be reversed. (Equation 8)

$$d_j(a, b) = -d_j(b, a) \quad (8)$$

**Partial Preference Indices.** For small deviations a small preference is allocated. If deviation is negligible, alternative  $a$  is not preferred over  $b$ . The greater deviation, the greater the preference. This preferences are real numbers in range from 0 to 1. Decision maker has a special function  $F_j$  for each criterion. (Equation 9)

$$P_j(a, b) = F_j[d_j(a, b)], \forall a, b \in A \quad (9)$$

where:

$P_j(a, b)$  is preference  $a$  over  $b$  on criterion  $j$

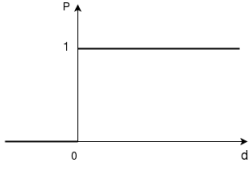
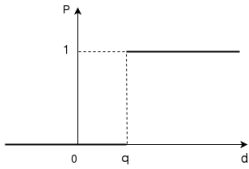
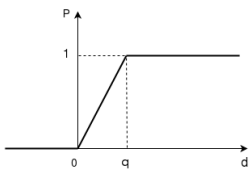
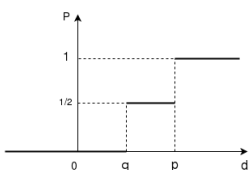
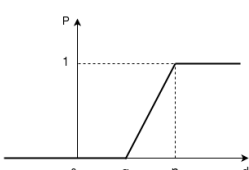
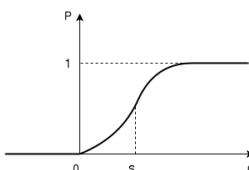
for which:

$$0 \leq P_j(a, b) \leq 1 \quad (10)$$

$$P_j(a, b) > 0 \Rightarrow P_j(b, a) = 0 \quad (11)$$

Preference function can take any form. Brans and Mareschal have defined a *generalised criterion*, which is the pair  $\{g_j(\cdot), P_j(a, b)\}$  associated for each criterion [4]. They proposed six types of preference functions. It is presented in Table 2.1.

Table 2.1: Types of generalised criteria(P(d):Preference function)

Generalised criterion	Definition	Parameters
Type 1: Usual Criterion		
	$P(d) = \begin{cases} 0 & d \leq 0 \\ 1 & d > 0 \end{cases}$	-
Type 2: U-shape Criterion		
	$P(d) = \begin{cases} 0 & d \leq q \\ 1 & d > q \end{cases}$	q
Type 3: V-shape Criterion		
	$P(d) = \begin{cases} 0 & d \leq 0 \\ \frac{d}{p} & 0 \leq d \leq p \\ 1 & d > p \end{cases}$	p
Type 4: Level Criterion		
	$P(d) = \begin{cases} 0 & d \leq q \\ \frac{1}{2} & q < d \leq p \\ 1 & d > p \end{cases}$	p,q
Type 5: V-shape with indifference Criterion		
	$P(d) = \begin{cases} 0 & d \leq q \\ \frac{d-q}{p-q} & q < d \leq p \\ 1 & d > p \end{cases}$	p,q
Type 6: Gaussian Criterion		
	$P(d) = \begin{cases} 0 & d \leq 0 \\ 1 - e^{-\frac{d^2}{2s^2}} & d > 0 \end{cases}$	s

Parameters present in third column have to be defined for criterion which are associated with this generalised criterion.

This parameters are:

$q$  is a threshold of indifference. It is the biggest deviation which is considered as insignificant by the decision-maker.

$p$  is a threshold of strict preference. It is the smallest deviation which is considered as sufficient to generate a full preference.

$s$  is an intermediate value between  $q$  and  $p$ . It defines the inflection point of the preference function.

**Aggregated Preference Indices.** Preferences of all criteria are aggregated as it is presented on Equation 12.

$$\begin{cases} \pi(a, b) = \sum_{j=1}^k P_j(a, b)w_j, \\ \pi(b, a) = \sum_{j=1}^k P_j(b, a)w_j \end{cases} \quad (12)$$

where:

$\pi(a, b)$  is showing with witch degree  $a$  is preferred to  $b$  over all criteria,

$w_j$  are criteria weights,

$k$  is the number of criteria.

Aggregated preference indices are usually positive because in most cases there are criteria for which  $a$  is better than  $b$  and other criteria for which  $b$  is better than  $a$ . This preferences have the following properties for all  $(a, b) \in A$ :

$$\begin{cases} \pi(a, a) = 0, \\ 0 \leq \pi(a, b) \leq 1, \\ 0 \leq \pi(b, a) \leq 1, \\ 0 \leq \pi(a, b) + \pi(b, a) \leq 1 \end{cases} \quad (13)$$

Global preference  $a$  over  $b$  is weak when  $\pi(a, b)$  is equal to 0 and it is strong when is equal to 1.

**Module M3. PrometheePreference.** Structure of this module is shown in Figure 2.3. It computes aggregated preference indices  $\pi(a, b)$  ( $out_1$ ) and partial preference indices  $P_j(a, b)$  ( $out_2$ ).

The module requires the user to specify an alternatives to consider ( $in_1$ ), a set of criteria together with the comparison thresholds and directions of preference ( $in_3$ ), performances of the alternatives( $in_4$ ) and criteria weights ( $in_6$ ).

The user has to parameterize the module ( $param_1$ ) to compare alternatives with each other, with boundary class profiles or characteristic class profiles. If comparison with

some profiles is selected then a set of categories profiles ( $in_2$ ) and their performances ( $in_5$ ) is required. In this mode alternatives are compared with class profiles and additionally class profiles are compared with each other.

Next parameter is generalised criterion ( $param_2$ ). If “specified” option is selected here then generalised criteria are required ( $in_7$ ). This input specifies generalised criterion (one of six defined types 1-6) for each criterion. Of course, each criterion can be assigned to other generalised criterion. The user may use simpler option and specify one generalised criterion for all criteria. Then (s)he can set option 1-6 on generalised criterion parameter ( $param_2$ ) and additional input is not needed.

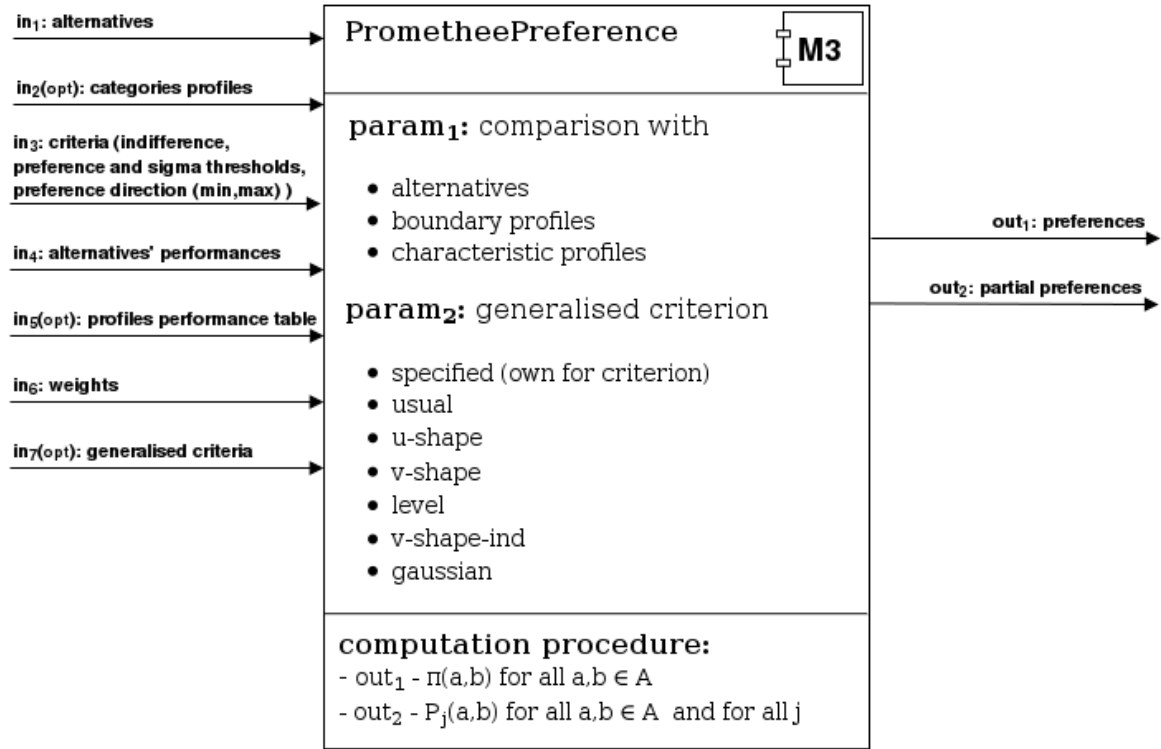


Figure 2.3: Structure of module M3 which computes Preference Indices

### 2.2.2 Preference with Reinforced Preference

Decision maker may want to take into account additional aspect in calculating preference indices. If  $a$  is preferred over  $b$  on specific criterion there may exist a small (e.g. the same as preference threshold) or a huge (e.g. several times more) difference in evaluations of these alternatives. The idea is to give some bonus for this huge difference in evaluations. We use reinforced preference threshold  $rp_j(\cdot)$  to check if difference in evaluations is considered to be large by the decision maker [30]. When this threshold is crossed then preference index calculated in normal way is multiplied by reinforcement factor  $\omega_j$ . This coefficient needs to be specified for each criterion, which takes the reinforced preference effect into account. Its value has to be greater than one ( $\omega_j > 1$ ). Upper limit is not

specified, but we recommend value in the range

$$1 - 2$$

. The set of criteria for which  $d_j(a, b) > rp_j$  is denoted in Equation 14.

$$F^{RP}(a, b) = \{j : aRP_jb \Leftrightarrow d_j(a, b) > rp_j\} \quad (14)$$

where:

$aRP_jb$  is reinforced preference  $a$  over  $b$  on criterion  $j$

$rp_j$  is the reinforced preference threshold on criterion  $j$

To calculate partial preference indices we use Equation 15.

$$P_j^{RP}(a, b) = \begin{cases} F_j[d_j(a, b)] & d_j(a, b) \leq rp_j \\ \omega_j & d_j(a, b) > rp_j \end{cases} \quad (15)$$

where:

$F_j$  is the function calculating preference (equation 9),

$\omega_j$  is the reinforcement factor of criterion  $j$ .

Aggregated preference index which includes reinforced preference effect is defined in Equation 16.

$$\pi^{RP}(a, b) = \frac{\sum_{j \in F^{RP}(a, b)} w_j \omega_j + \sum_{j \in F \setminus F^{RP}(a, b)} w_j \cdot P_j(a, b)}{\sum_{j \in F^{RP}(a, b)} w_j \omega_j + \sum_{j \in F \setminus F^{RP}(a, b)} w_j} \quad (16)$$

where:

$F$  is the set of all criteria.

**Module M4. PrometheusPreferenceReinforcedPreference.** This module computes aggregated preference indices with reinforced preference effect  $\pi^{RP}(a, b)$  (*out*<sub>1</sub>) and partial preference indices also with reinforced preference effect  $P_j^{RP}(a, b)$  (*out*<sub>2</sub>). Its structure is presented in Figure 2.4.

This module is very similar to module M3. It computes preference indices with reinforced preference effect which requires additional data inputs. The user has to specify reinforcement factors  $\omega_j$  (*in*<sub>8</sub>). All criteria with reinforcement factors require reinforced preference thresholds  $rp_j$  (*in*<sub>3</sub>).

This module does not support Gaussian generalised criterion (type 6) and reinforced preference effect specified in the same criterion. Gaussian function can be listed in generalised criteria (*in*<sub>7</sub>), but in these criteria reinforcement factor is forbidden. User cannot choose Gaussian criterion in generalised criterion parameter (*param*<sub>2</sub>). Criteria thresholds also should not include “sigma” and “reinforced preference” thresholds simultaneously.

The remaining inputs are the same as in module M3. Parameters are also the same as in module M3 except of type 6 generalised criterion which is not available. This module requires at least one criterion with reinforced preference effect (reinforcement factor and reinforced preference threshold specified).



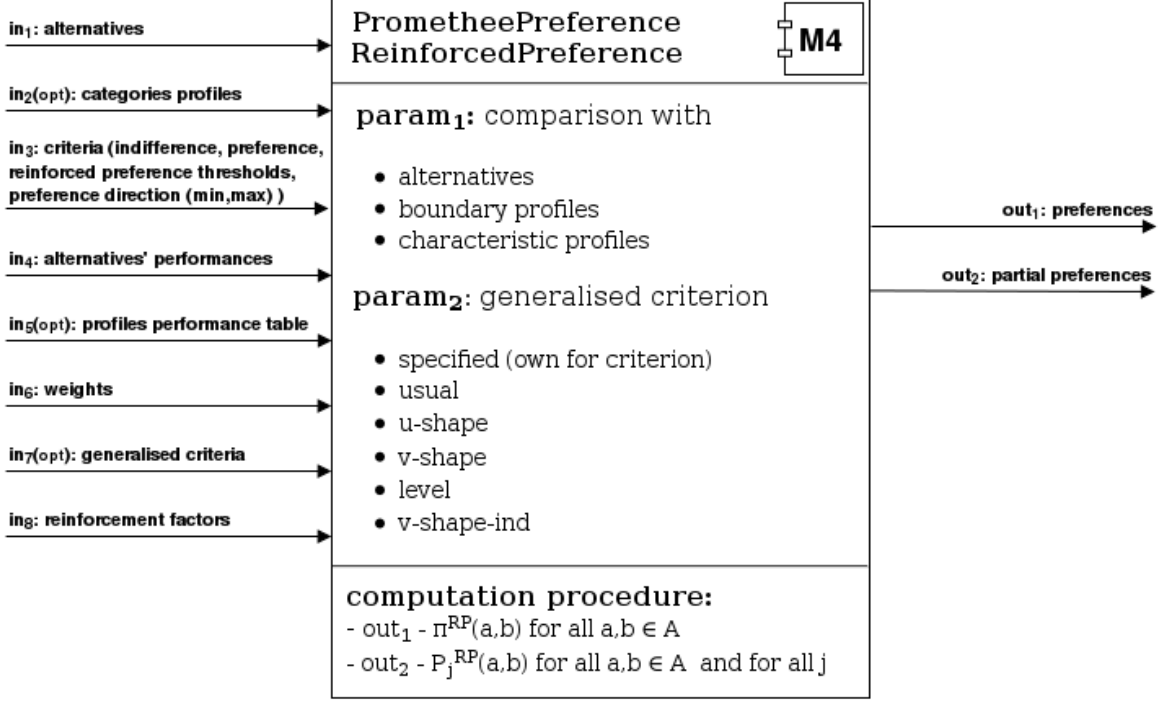


Figure 2.4: Structure of module M4 which computes Preference Indices with Reinforced Preference effect

### 2.2.3 Preference with interactions between criteria

Previously, we have assumed that there are no interactions between the criteria. We need to change the formula that calculates the aggregated preference indices to take these interactions into account [14]. Our new equation needs to consider the weights of the interaction coefficients. There are three types of interactions between criteria to consider:

- Mutual strengthening effect

If criteria  $g_i$  and  $g_j$  support  $aPb$  then their influence for aggregated preference index should be stronger (greater than  $w_i + w_j$ ). This effect can be modeled by mutual strengthening coefficient  $k_{ij}^s = k_{ji}^s > 0$ . The set of criteria pairs which have this interaction effect is denoted with Equation 17.

$$F^{SE}(a, b) = \{i, j \in J : i, j \in F^P(a, b) \text{ and } k_{ij}^s > 0\} \quad (17)$$

where:

$k_{ij}^s$  is a mutual strengthening coefficient,

$J$  is a set of criteria ID's  $J = \{1, 2, \dots, k\}$ ,  $k$  - number of criteria,

for which:

$$F^P(a, b) = \{j : aP_jb \Leftrightarrow P_j(a, b) > 0\} \quad (18)$$

- Mutual weakening effect

If criteria  $g_i$  and  $g_j$  support  $aPb$ , then their influence for aggregated preference index should be weaker (smaller than  $w_i + w_j$ ). This effect can be modeled by mutual weakening coefficient  $k_{ij}^w = k_{ji}^w < 0$ . The set of criteria pairs for which this interaction effect holds is denoted with Equation 19.

$$F^{WE}(a, b) = \{i, j \in J : i, j \in F^P(a, b) \text{ and } k_{ij}^w < 0\} \quad (19)$$

where:

$k_{ij}^w$  is mutual weakening coefficient

- Antagonistic effect

If criterion  $g_i$  support  $aPb$  and criterion  $g_j$  supports  $bPa$ , then the influence of criterion  $g_i$  for aggregated preference index should be weaker (smaller than  $w_i$ ). This effect can be modeled by an antagonism coefficient  $k_{ij}^a > 0$ , which intervenes negatively in  $\pi(a, b)$ . The set of criteria pairs which have this interaction effect is denoted with Equation 20.

$$F^{AE}(a, b) = \{(i, j) \in J \times J : i \in F^P(a, b), j \in F^P(b, a) \text{ and } k_{ij}^a > 0\} \quad (20)$$

where  $k_{ij}^a$  is antagonism coefficient. The antagonistic effect for pair  $(i, j) \in J \times J$  does not mean that reverse effect for  $(j, i)$  exist. Of course, it does not exclude existing of this reverse effect.

It should be noted that mutual strengthening effect and mutual weakening effect are mutually exclusive.

The new aggregated preference index with interactions between criteria is defined in Equation 21.

$$\pi^I(a, b) = \frac{\sum_{j \in J} P_j(a, b)w_j + \sum_{\{i, j\} \in F^{SE}(a, b)} Z_{ij}^{ab} \cdot k_{ij}^s + \sum_{\{i, j\} \in F^{WE}(a, b)} Z_{ij}^{ab} \cdot k_{ij}^w - \sum_{\{i, j\} \in F^{AE}(a, b)} Z_{ij}^{ab} \cdot k_{ij}^a}{\sum_{j \in J} w_j + \sum_{\{i, j\} \in F^{SE}(a, b)} Z_{ij}^{ab} \cdot k_{ij}^s + \sum_{\{i, j\} \in F^{WE}(a, b)} Z_{ij}^{ab} \cdot k_{ij}^w - \sum_{\{i, j\} \in F^{AE}(a, b)} Z_{ij}^{ab} \cdot k_{ij}^a} \quad (21)$$

Function  $Z_{ij}^{ab}$  is used to capture the interaction effects in the ambiguity zone. This can take one of many forms. Figueira, Greco and Roy proposed two approaches presented in Equations 22 and 23.

$$Z_{ij}^{ab} = Z(P_i(a, b), P_j(a, b)) = \min\{P_i(a, b), P_j(a, b)\} \quad (22)$$

$$Z_{ij}^{ab} = Z(P_i(a, b), P_j(a, b)) = P_i(a, b) \cdot P_j(a, b) \quad (23)$$

**Module M5. PrometheePreferenceWithInteractions.** Structure of this module is presented in Figure 2.5. It computes aggregated preference indices with interactions between criteria  $\pi^I(a, b)$  ( $out_1$ ) and partial preference indices  $P_j(a, b)$  ( $out_2$ ).

This module is very similar to module M3. It computes partial preference indices in normal way, and then uses additional information about interactions ( $in_8$ ) to calculate aggregated indices. All other data inputs are the same as in module M3.

The user has an additional parameter to specify. There are two types of  $Z$  functions prepared and it is required to choose one of them ( $param_3$ ).

If there are no interactions between criteria, it is recommended to use module M3 instead of this one.

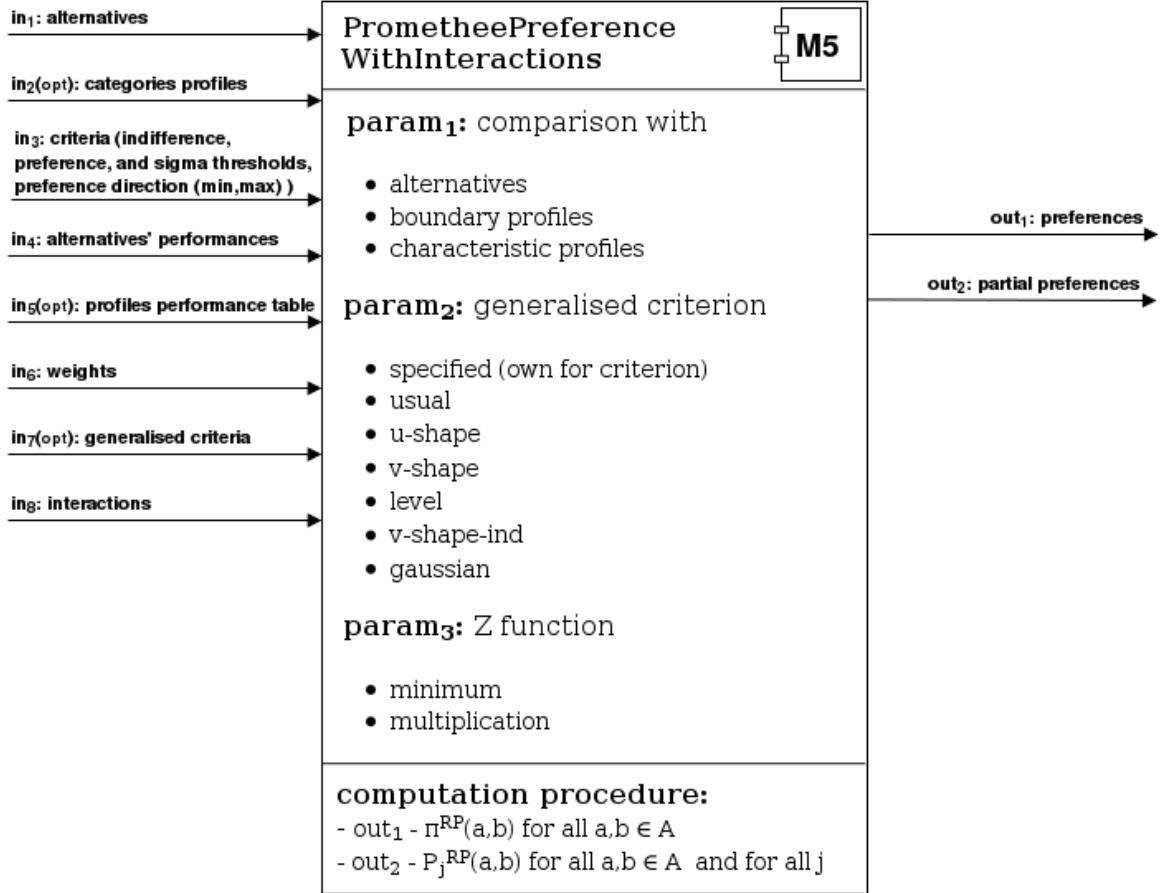


Figure 2.5: Structure of module M5 which computes Preference Indices with Interactions between criteria

### 2.2.4 Discordance

Preference index  $P_j(a, b)$  in PROMETHEE measures only the intensity of agreement with preposition  $aP_jb$ . Hu and Chen proposed overall preference index which includes both votes for and against  $aP_jb$  [18]. They suggested concordance index  $C_p(a, b)$  which is analogical to already existing in PROMETHEE aggregated preference index  $\pi(a, b)$ . It is presented in Equation 24.

$$C_p(a, b) = \sum_{j=1}^k P_j(a, b)w_j \quad (24)$$

Partial discordance indices  $D_j(a, b)$  are calculated from partial preference indices as in Equation 25.

$$D_j(a, b) = P_j(b, a) \quad (25)$$

An overall discordance  $D_p$  can be calculated by aggregating  $D_j$  (see Equation 26).

$$D_p(a, b) = 1 - \prod_{j=1}^k (1 - D_j(a, b))^{\tau/k} \quad (26)$$

where:

$\tau$  is technical parameter,  $\tau \in [1, k]$ , smaller  $\tau \rightarrow$  weaker discordance

An overall preference index is presented in Equation 27.

$$P(a, b) = C_p(a, b) \cdot (1 - D_p(a, b)) \quad (27)$$

**Module M6. PrometheeDiscordance.** Structure of this module is presented in Figure 2.6. It computes aggregated discordance indices  $D_p(a, b)$  ( $out_1$ ) and partial discordance indices  $D_j(a, b)$  ( $out_2$ ).

It requires the user to specify alternatives to be considered ( $in_1$ ), a set of criteria ( $in_3$ ) and partial preference indices ( $in_4$ ). If user wants to calculate discordance indices for comparing alternatives with profile(s) he has to set it in “comparison with” parameter ( $param_1$ ) and to specify categories profiles ( $in_2$ ). This module requires also to specify technical parameter ( $param_2$ ). It has to be an integer number, whose maximum value is determined by the number of criteria defined in criteria file ( $in_3$ ).

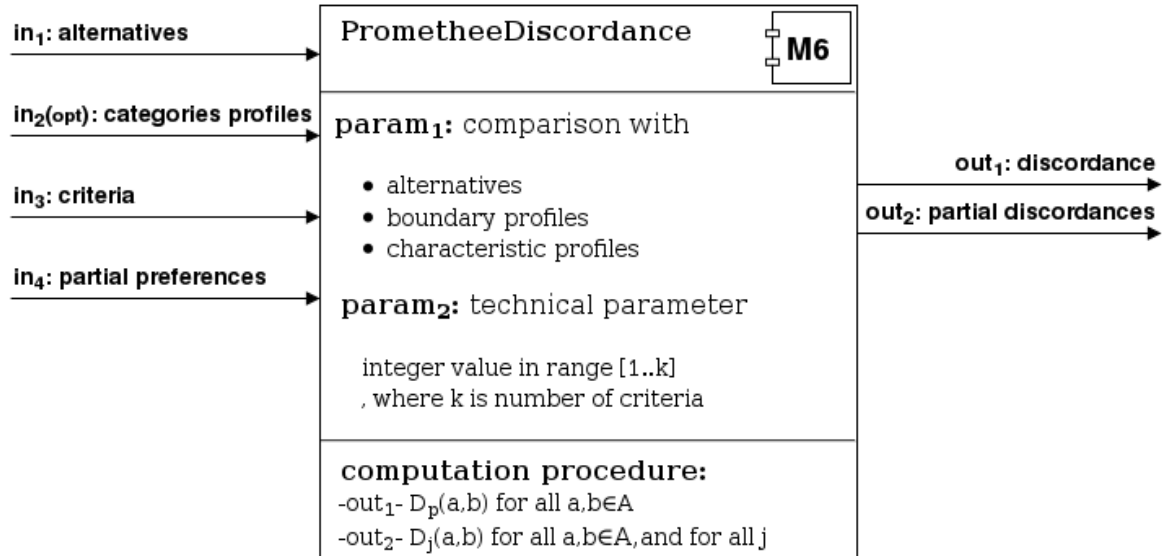


Figure 2.6: Structure of module M6 which computes Discordance Indices

### 2.2.5 Veto

In classic PROMETHEE approach there is no discordances and even veto thresholds. Only preference indices are considered [18]. In many cases it is necessary to refuse alternative completely if it has very weak evaluation on specific criterion. We adapt the idea of veto originally developed for the context of ELECTRE methods to the PROMETHEE methods [29]. The veto threshold is critical feature in decision making and it is very simple in concept. If alternative  $b$  has better evaluation ( $g_j(b)$ ) than  $a$  ( $g_j(a)$ ) on some criterion by at least the veto threshold ( $v_j$ ) then  $a$  cannot be preferred over  $b$  regardless of the ratings on the other criteria. According to this definition we can calculate partial veto indices as in Equation 28, where we use deviation ( $d_j(b, a)$ ) defined in Equation 7.

$$V_j(a, b) = \begin{cases} 1 & d_j(b, a) \geq v_j \\ 0 & d_j(b, a) < v_j \end{cases} \quad (28)$$

Aggregated veto indices can be calculated in two ways. In first approach, one criterion with exceeded veto threshold is enough to set aggregated veto index to 1 (Equation 29).

$$V_p(a, b) = \begin{cases} 1 & \text{if } \exists j \in J, V_j(a, b) = 1 \\ 0 & \text{if } \forall j \in J, V_j(a, b) = 0 \end{cases} \quad (29)$$

In second approach, the idea is to use criteria weights as in aggregated preference index (Equation 30). In this case, the preference is not completely discarded when the veto threshold is exceeded. It can be used like discordance which decrease overall preference index.

$$V_p(a, b) = \sum_{j \in J} V_j(a, b) \cdot w_j \quad (30)$$

Aggregated veto indices can be combined with aggregated preference indices into overall preference indices like in Equation 31.

$$P(a, b) = \pi(a, b) \cdot (1 - V_p(a, b)) \quad (31)$$

**Module M7. PrometheeVeto.** Structure of this module is presented in Figure 2.7. It computes aggregated veto indices  $V_p(a, b)$  ( $out_1$ ) and partial veto indices  $V_j(a, b)$  ( $out_2$ ).

It requires the user to specify alternatives to consider ( $in_1$ ), criteria ( $in_3$ ), which contains information about criteria, veto thresholds and preference directions; performance table ( $in_4$ ) and weights of criteria ( $in_6$ ).

Same as in other modules of this type, the user can parameterize this module ( $param_1$ ) to compare alternatives with each other, with boundary class profiles or characteristic class profiles. If comparison with some profiles is selected then a set of categories profiles ( $in_2$ ) and their performances ( $in_5$ ) is required.

Aggregation of veto indices can be processed in one of two predefined ways. User can use disjunctive method ( $param_2 : not\ specified$ ) or method with criteria weights ( $param_2 : specified$ ).

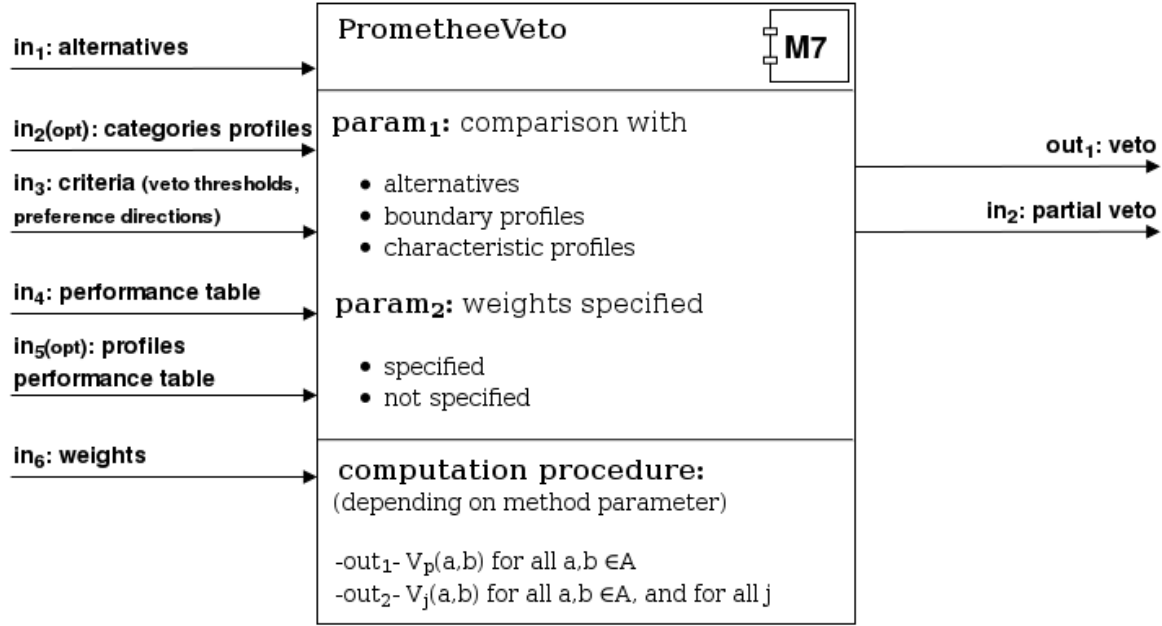


Figure 2.7: Structure of module M7 which computes Veto Indices

### 2.2.6 Overall preference index

Overall preference index proposed by Hu and Chen [18] can be used to combine both discordances with preferences and veto indices with preference. This overall preference index can be computed as in Equation 32.

$$P(a, b) = \pi(a, b) \cdot (1 - D(a, b)) \quad (32)$$

where:

$\pi(a, b)$  is aggregated preference index

$D(a, b)$  is aggregated discordance index or aggregated veto index

**Module M8. PrometheePreferenceDiscordance.** Structure of this module is presented in Figure 2.8. It computes overall preference indices  $P(a, b)$  ( $out_1$ ).

It requires the user to specify a set of alternatives ( $in_1$ ), aggregated preference indices  $\pi(a, b)$  ( $in_3$ ) and aggregated discordance indices or aggregated veto indices  $D(a, b)$  ( $in_4$ ). The module has to be parameterized to comparing alternatives with other alternatives, boundary profiles or central profiles ( $param_1$ ). Comparing with profiles requires the user to specify categories profiles ( $in_2$ ).

## 2.3 Outranking Flows

In this section, we define positive and negative outranking flows. We show how to compute them using preference indices mentioned in Section 2.2. At the same time we present the module that we have created (M9).

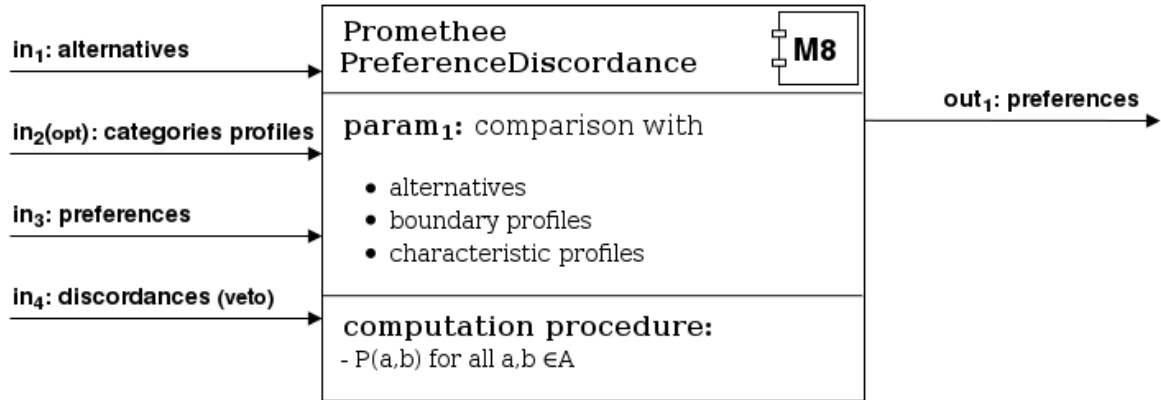


Figure 2.8: Structure of module M8 which combines preference and discordance (or Veto) indices

### Positive and Negative Outranking Flow

The positive outranking flow is a value which shows how alternative  $a$  is outranking all other alternatives [4]. It is showing the power of alternative, its outranking character (Figure 2.9a). Alternative is the better, the larger its positive outranking flow.

The negative outranking flow is a value which shows how alternative  $a$  is outranked by all other alternatives [4]. It is showing the weakness of alternative, its outranked character (Figure 2.9b). Alternative is the better, the lower its negative outranking flow.

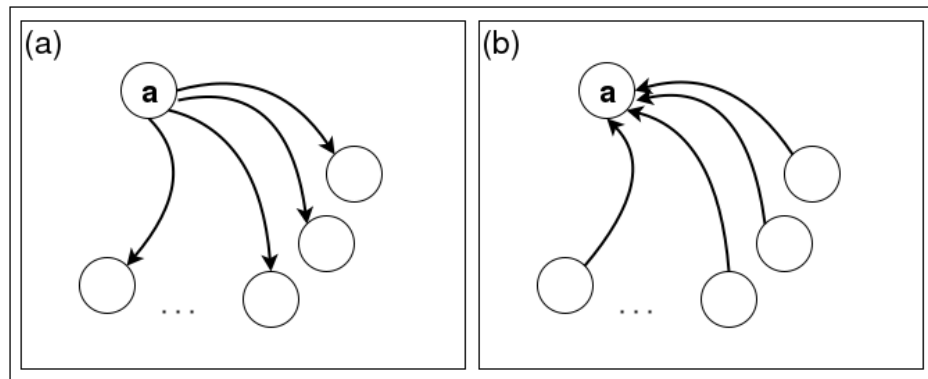


Figure 2.9: (a) The Positive Outranking Flow. (b) The Negative Outranking Flow

We can define the positive outranking flow like in Equation 33 and the negative outranking flow like in Equation 34 [4].

$$\phi^+(a) = \frac{1}{n-1} \sum_{x \in A \setminus \{a\}} \pi(a, x) \quad (33)$$

$$\phi^-(a) = \frac{1}{n-1} \sum_{x \in A \setminus \{a\}} \pi(x, a) \quad (34)$$

where:

$a, x$  are the alternatives,

$A$  is the set of all alternatives,

$n$  is the number of all alternatives.

**Module M9. PrometheeOutrankingFlows.** Structure of this module is presented in Figure 2.10. It computes the positive outranking flows  $\phi^+(a)$  ( $out_1$ ) and the negative outranking flows  $\phi^-(a)$  ( $out_2$ ).

It requires the user to specify alternatives to consider ( $in_1$ ), aggregated preference indices  $\pi(a, b)$  ( $in_3$ ). User can parameterize this module ( $param_1$ ) to compare alternatives with each other, with boundary class profiles or characteristic class profiles. If comparison with some profiles is selected then the categories profiles are required ( $in_2$ ).

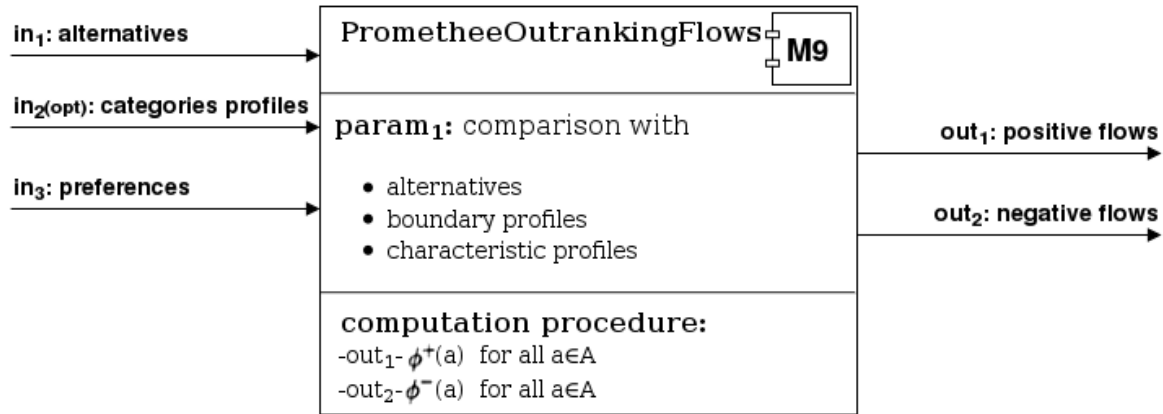


Figure 2.10: Structure of module M9 which computes the positive and negative outranking flows

## 2.4 Profiles of the Alternatives

In this section, we present the idea of profiling the alternatives using the single criterion net flows. We present the module which allows to create these profiles in diviz platform (M10).

### Profiles of the Alternatives

The profiles of alternatives are used to determine the “quality” of alternatives on different criteria. It is very helpful for DM. The profile of the alternative is constructed of the set of the single criterion net flows computed for all criteria  $(\phi_j(a), j = 1, 2, \dots, k)$  [4].

If the value of the single criterion net flow is greater than zero ( $\phi_j(a) > 0$ ), it expresses how the alternative  $a$  is outranking all the other alternatives on criterion  $g_j()$ . If this value is less than zero ( $\phi_j(a) < 0$ ) it expresses how the alternative  $a$  is outranked by all the other alternatives on criterion  $g_j()$ .



The single criterion net flow can be computed using partial preference indices  $P_j(a, b)$  as in Equation 35.

$$\phi_j(a) = \frac{1}{n-1} \sum_{x \in A \setminus \{a\}} [P_j(a, x) - P_j(x, a)] \quad (35)$$

By aggregating single criterion net flows using weights of criteria the net outranking flow can be easily computed (Equation 36).

$$\phi(a) = \sum_{j=1}^k \phi_j(a) \cdot w_j \quad (36)$$

**Module M10. PrometheeAlternativesProfiles.** Structure of this module is presented in Figure 2.11. It computes the single criterion net flows  $\phi_j(a)$  ( $out_1$ ) for all criteria and alternatives. These data allow to build the profiles of the alternatives.

It requires the user to specify a set of alternatives ( $in_1$ ), criteria set ( $in_2$ ) and partial preference indices ( $in_3$ ).

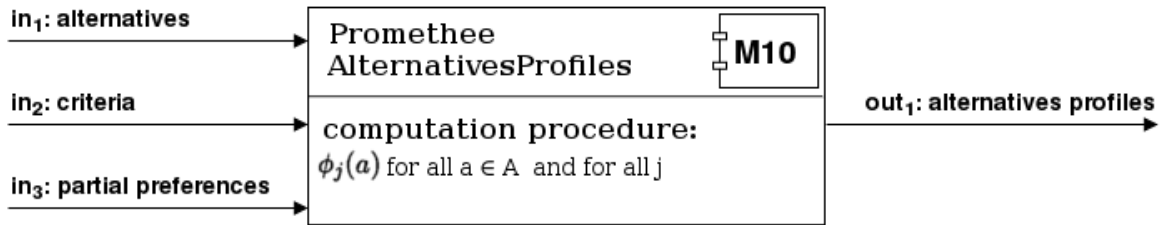


Figure 2.11: Structure of module M10 which computes Alternatives Profiles

## 2.5 Ranking Problems

In this section, we focus on different ways of ranking alternatives using PROMETHEE methods. In the first approach, the ranking is determined by the final comprehensive flows. The next one is ranking presented as the weak preference relation, which enables to show comparisons between alternatives. Last but not least is an approach which is the most associated with ranking and returns as the output positions in the ranking. We demonstrate a few modules we have implemented. They are numbered for easy identification (M11 - M16). In next subsections, we show examples to all described ranking methods.

### 2.5.1 Promethee I

Promethee I ranking is calculating method based on the positive and negative outranking flows. The final result is only partial - in some ambiguous cases this method does not decide which of the current alternatives is better. The general method of computing the ranking is presented in Equation 37 [4].

$$P_{I\_RANK} = \begin{cases} aP^I b & \iff \begin{cases} \phi^+(a) > \phi^+(b) \text{ and } \phi^-(a) < \phi^-(b) \text{ or} \\ \phi^+(a) = \phi^+(b) \text{ and } \phi^-(a) < \phi^-(b) \text{ or} \\ \phi^+(a) > \phi^+(b) \text{ and } \phi^-(a) = \phi^-(b) \end{cases} \\ aI^I b & \iff \phi^+(a) = \phi^+(b) \text{ and } \phi^-(a) = \phi^-(b) \\ a?^I b & \iff \begin{cases} \phi^+(a) > \phi^+(b) \text{ and } \phi^-(a) > \phi^-(b) \text{ or} \\ \phi^+(a) < \phi^+(b) \text{ and } \phi^-(a) < \phi^-(b) \end{cases} \end{cases} \quad (37)$$

It can be generalized to the relation of the weak preference (Equation 38 ).

$$P_{I\_RANK} = \begin{cases} aS^I b & \iff \begin{cases} \phi^+(a) > \phi^+(b) \text{ and } \phi^-(a) < \phi^-(b) \text{ or} \\ \phi^+(a) = \phi^+(b) \text{ and } \phi^-(a) < \phi^-(b) \text{ or} \\ \phi^+(a) > \phi^+(b) \text{ and } \phi^-(a) = \phi^-(b) \text{ or} \\ \phi^+(a) = \phi^+(b) \text{ and } \phi^-(a) = \phi^-(b) \end{cases} \\ a?^I b & \iff \begin{cases} \phi^+(a) > \phi^+(b) \text{ and } \phi^-(a) > \phi^-(b) \text{ or} \\ \phi^+(a) < \phi^+(b) \text{ and } \phi^-(a) < \phi^-(b) \end{cases} \end{cases} \quad (38)$$

**Module M11. PrometheeIRanking.** Structure of this module is presented in Figure 2.12. It computes ranking in Promethee I method based on the positive ( $in_2$ ) and negative outranking flows ( $in_3$ ) with available list of all alternatives ( $in_1$ ). The result of this module is the ranking presented as the weak preference relation, which means that the output file consists of pairs of alternatives for which the outranking relations is valid ( $out_1$ ).

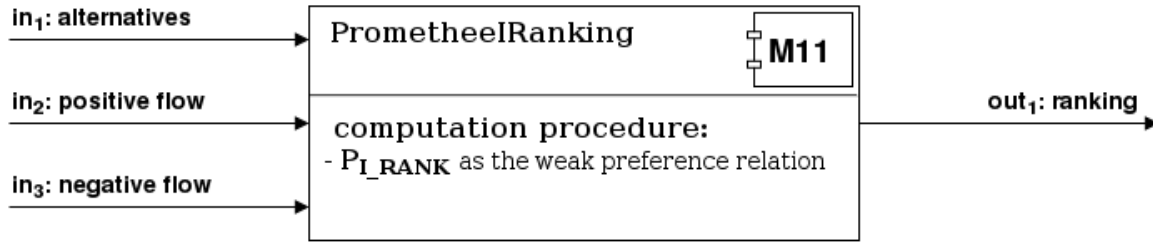


Figure 2.12: Structure of module M11 which computes Promethee I Ranking

## 2.5.2 Promethee II

In contrary to Promethee I, Promethee II returns a complete ranking without any incomparabilities. Firstly, there is calculated value which is called *net outranking flow*. It is a difference between positive and negative flow for each alternative (which is shown in Equation 39) and put all together in one set (see Equation 40) [4].

$$\phi(a) = \phi^+(a) - \phi^-(a) \quad (39)$$

$$P_{II\_FLOW} = \{\forall a \in alternatives : \phi(a)\} \quad (40)$$

**Module M12. PrometheeIIFlow.** Structure of this module is presented in Figure 2.13. It computes ranking in Promethee II method based on the positive ( $in_2$ ) and negative outranking flows ( $in_3$ ) with available list of all alternatives ( $in_1$ ). The fourth file ( $in_4$ ) is optional - it should be included only when the value of parameter ( $param_1$ ) is different from “alternatives”. It consists of list of boundary/characteristic profiles. This module requires a parameter which determines if a flow is calculated as a comparison of only alternatives (then we need to choose “alternatives”), comparison of alternatives and boundary profiles or characteristic profiles. The result of this module is a set which consists of net outranking flows described above. Moreover, flows can be calculated not only for alternatives but also for profiles - it depends on the selected program parameters.

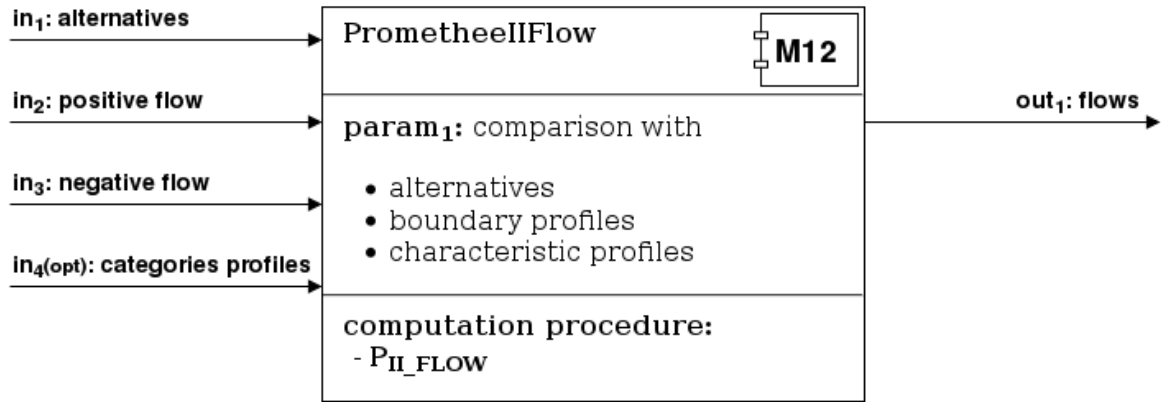


Figure 2.13: Structure of module M12 which computes Promethee II Flows

### 2.5.3 Promethee III

Ranking in Promethee III is not based on net outranking flows, as it is in Promethee I or Promethee II but it is based on intervals. It enables to relate to the current set of data in more practical way - there is no need to have the same values to make two alternatives indifferent. The ranking is calculated in the following way: based on the preferences (shown in Equation 12 ) and transformations typical for Promethee III (see Equation 41), we calculate intervals as revealed by Equation 42 [6].

$$\left\{ \begin{array}{l} \phi(\bar{a}) = \frac{1}{n} \sum_{b \in A} (\Pi(a, b) - \Pi(b, a)) \\ \sigma_a^2 = \frac{1}{n} \sum_{b \in A} (\Pi(a, b) - \Pi(b, a) - \phi(\bar{a}))^2 \\ \alpha > 0 \end{array} \right. \quad (41)$$

where:

$n$  is number of alternatives,

$\alpha$  is a parameter given by the DM.

$$\begin{cases} x_a = \phi(\bar{a}) - \alpha\sigma_a, \\ y_a = \phi(\bar{a}) + \alpha\sigma_a, \end{cases} \quad (42)$$

Then, the final ranking can be designated as it is shown in Equation 43.

$$P_{III\_RANK} = \begin{cases} aP_{III}b \iff x_a > y_b \\ aI_{III}b \iff x_a \leq y_b \text{ and } x_b \leq y_a \end{cases} \quad (43)$$

**Module M13. PrometheeIIIFlow.** Structure of this module is presented in Figure 2.14. It requires providing list of alternatives ( $in_1$ ), positive ( $in_2$ ) and negative outranking flows ( $in_3$ ), matrix of preference degrees ( $in_4$ ). There is also a need to give the  $param_1 - \alpha$  which is decimal value between 0 and 1. This module returns two files. Firstly, it computes intervals ( $out_1$ ) which are described above and shown in Equation 42. The other output is ranking ( $out_2$ ) – based on this from Equation 43 – which is represented as a weak preference relation.

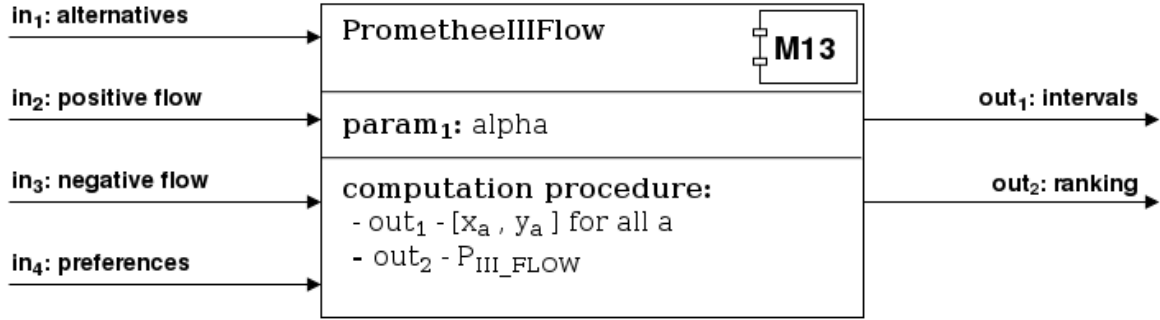


Figure 2.14: Structure of module M13 which computes Promethee III intervals and ranking

## 2.5.4 Net Flow Score

Net Flow Score is a ranking method which bases on calculating scores associated with each alternative [3]. There are nine basic scoring functions which are shown in Equations 44 - 52.

$$\text{max in favor} : \max_{b \in A' \setminus \{a\}} \tilde{R}(a, b) \quad (44)$$

$$\text{min in favor} : \min_{b \in A' \setminus \{a\}} \tilde{R}(a, b) \quad (45)$$

$$\text{sum in favor} : \sum_{b \in A' \setminus \{a\}} \tilde{R}(a, b) \quad (46)$$

$$\text{-max against} : - \max_{b \in A' \setminus \{a\}} \tilde{R}(b, a) \quad (47)$$

$$\text{-min against} : - \min_{b \in A' \setminus \{a\}} \tilde{R}(b, a) \quad (48)$$

$$\text{-sum against} : - \sum_{b \in A' \setminus \{a\}} \tilde{R}(b, a) \quad (49)$$

$$\text{max difference} : \max_{b \in A' \setminus \{a\}} \tilde{R}(a, b) - \tilde{R}(b, a) \quad (50)$$

$$\text{min difference} : \min_{b \in A' \setminus \{a\}} \tilde{R}(a, b) - \tilde{R}(b, a) \quad (51)$$

$$\text{sum of differences} : \sum_{b \in A' \setminus \{a\}} \tilde{R}(a, b) - \tilde{R}(b, a) \quad (52)$$

There are a few approaches in designating the Net Flow Score values. We want to focus on two of them. First one, which is known as *Net Flow Rule* is a base of all the others. It says that we just take the results given by the chosen scoring function. There is possibility that some alternatives have the same value but algorithm does not take this into account [3].

The other approach implies that we can run the procedure of calculating scores more than once. It happens only when there are a few alternatives with the same results of scoring function. In such a situation the whole procedure is started again but only for the set of alternatives with the same values. While creating ranking, there is need to take into account values from all iterations [20].

**Module M14. NetFlowScore.** Structure of this module is presented in Figure 2.15. It requires list of alternatives ( $in_1$ ) and matrix of preferences ( $in_2$ ). Decision maker also needs to provide two parameters: first one stands for function ( $param_1$ ): max, min or sum. The second one stands for direction ( $param_2$ ): in favor, against or difference. This module computes values of chosen scoring function for each alternative ( $NFS_a$ ).

**Module M15. NetFlowScoreIterative.** Structure of this module is presented in Figure 2.16. It requires a list of alternatives ( $in_1$ ) and matrix of preferences ( $in_2$ ). Decision maker also needs to provide two parameters. First one stands for function max, min or sum ( $param_1$ ). The second one stands for direction: in favor, against or difference ( $param_2$ ). This module computes the values of chosen scoring function according to the algorithm described above for each alternative and then assign to each alternative proper position in the ranking ( $NFS_a^i$ ).

### 2.5.5 GDSS Ranking

GDSS Ranking is a method which can be used to create ranking basing on decisions from many decision makers. It is simply calculated as the weighted sum of flows for every alternative and is shown in Equation 53 [22].

$$\Phi_a^G = \sum_{n=1}^N \phi_a^n \omega_n \quad (53)$$

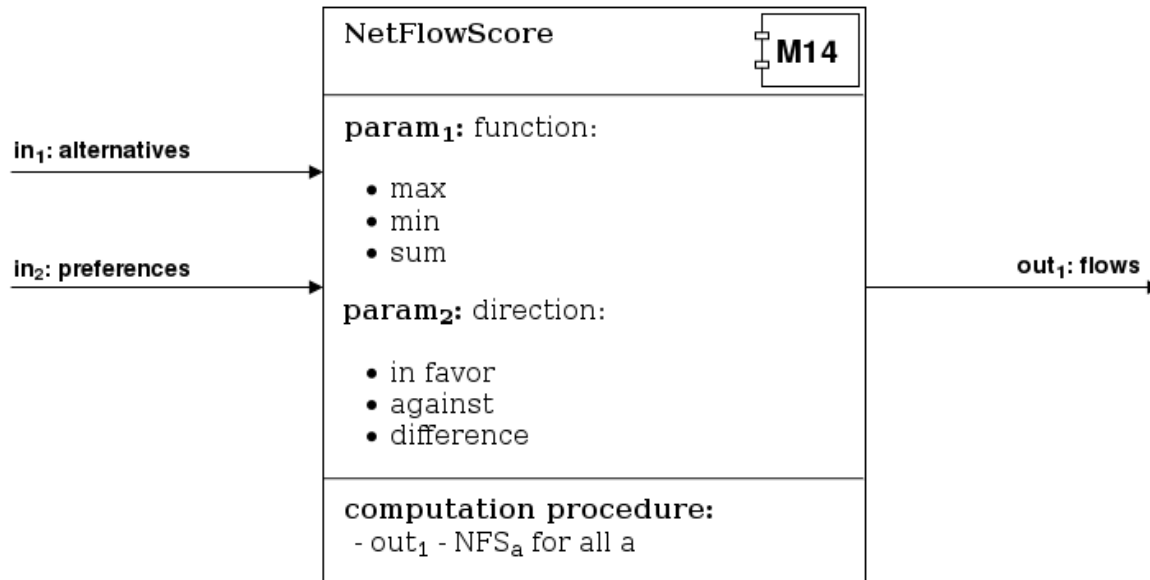


Figure 2.15: Structure of module M14 which computes Net Flow Score

**Module M16. PrometheeGroupRanking.** Structure of this module is presented in Figure 2.17. It requires providing list of alternatives ( $in_1$ ), flows from every decision maker ( $in_{2-11}$ ) and a file with weights assigned to every decision maker ( $in_{12}$ ). It computes values of weighted flows for each alternative ( $out_1$ ) and puts aggregated flows into one file ( $out_2$ ).

## 2.6 Sorting Problems

In this section, we focus on different ways of resolving sorting problems using PROMETHEE methodology. Each alternative  $a \in A$  has to be assigned to one or, in general, some of  $K$  categories  $C_1, C_2, \dots, C_K$ . Without loss of generality, we suppose that they are ordered from the worst to the best, so that  $C_h$  is preferred to  $C_{h-1}$  when  $h > l$ . We can define each category by one central profile which represents typical alternative which should be assigned to the category, two limiting profiles which limit each category as the best and worst alternative, or boundary profiles which separate pairs of subsequent categories. When using central profiles, we need to define  $K$  profiles  $r_1, r_2, \dots, r_K \in R$ , one for each category. In case of using boundary profiles, we need to define  $K - 1$  profiles  $r_1, r_2, \dots, r_{K-1} \in R$  separating categories. For the limiting profiles we need to define  $K + 1$  profiles  $r_1, r_2, \dots, r_{K+1} \in R$  setting the classes limits ( $r_1$  will be here the worst possible alternative and  $r_{K+1}$  will be best possible alternative). Performance of profiles is connected with categories preferences, as they are ordered. Most of methods described in this section propose its own dominance conditions concerning profiles.

Each sorting method described in this thesis is based either on total net flows (as in Promethee II) or positive and negative flows (as in Promethee I). At the output, we obtain either precise or imprecise assignment of each alternative to, respectively, one category or few categories. There are plenty of methods in PROMETHEE family

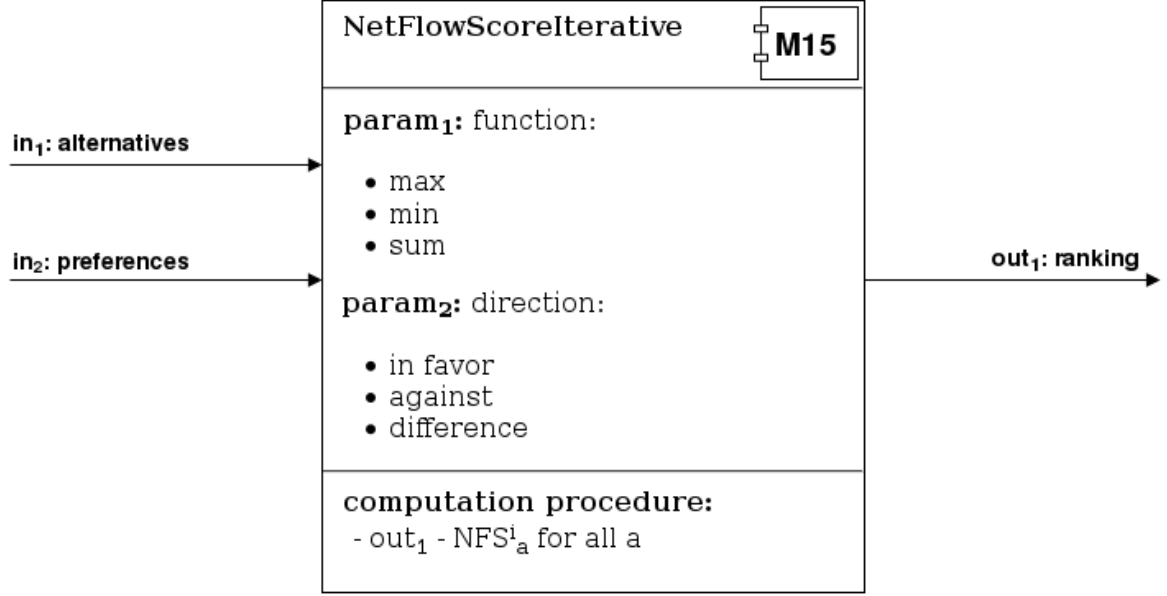


Figure 2.16: Structure of module M15 which computes Net Flow Score Iterative.

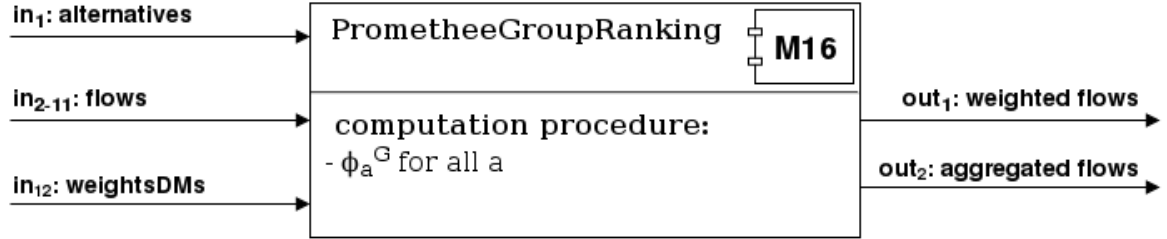


Figure 2.17: Structure of module M16 which computes GDSS Ranking.

resolving sorting problems in different ways. In the next subsections, we focus on some of them and propose a few modules which are based on these methods (M17 - M22).

### 2.6.1 PromSort

PromSort is a sorting method from Promethee family that bases on outranking relations between alternatives and boundary profiles. It uses Promethee I positive and negative flows.

We can define 3 types of outranking relation between alternative  $a_i$  and profile  $r_k$  - preference (shown in Formula 54), indifference (Formula 55) and incomparability (Formula 56) [1].

$$(a_i P r_k) \quad \text{if} \quad \begin{cases} \phi^+(a_i) > \phi^+(r_k) \wedge \phi^-(a_i) < \phi^-(r_k) \\ \phi^+(a_i) = \phi^+(r_k) \wedge \phi^-(a_i) < \phi^-(r_k) \\ \phi^+(a_i) > \phi^+(r_k) \wedge \phi^-(a_i) = \phi^-(r_k) \end{cases} \quad (54)$$

$$(a_i I r_k) \quad \text{if} \quad \phi^+(a_i) = \phi^+(r_k) \wedge \phi^-(a_i) = \phi^-(r_k) \quad (55)$$

$$(a_i R r_k) \text{ if } \begin{cases} \phi^+(a_i) > \phi^+(r_k) \wedge \phi^-(a_i) > \phi^-(r_k) \\ \phi^+(a_i) < \phi^+(r_k) \wedge \phi^-(a_i) < \phi^-(r_k) \end{cases} \quad (56)$$

As input categories are ordered, PromSort requires the profiles to fulfill the dominance condition as in Formula 57. In this formula,  $p_j$  is a preference threshold for criterion  $j$ ,  $g_j$  is the evaluation of profile. Each profile needs to be preferred over profiles which are worse than it.

$$\forall j, \forall k 1, \dots, K - 1 \quad g_j(b_k) + p_j \leq g_j(b_{k+1}) \quad (57)$$

The assignment procedure in PromSort can be divided into two steps. In the first step, the DM needs to compare alternative  $a_i$  with each profile  $r_k$  from the best ( $K - 1$ ) to the worst (1). After the comparison there are a few possible options:

1. If alternative  $a_i$  is preferred to profile  $r_{K-1}$  it should be assigned to the category  $C_K$ .
2. If each profile is preferred to alternative  $a_i$ , the latter should be assigned to the category  $C_1$ .
3. If  $r_p$  is the first profile such that  $a_i P r_p$  and  $r_s$  is the first profile that  $a_i I r_s$  or  $a_i R r_s$  and  $p > s$ , alternative  $a_i$  should be assigned to the category  $C_{p+1}$ .
4. If none of above holds, alternative  $a_i$  should not be assigned to any of the categories. It should be assigned to the category  $s$  or  $s + 1$  in the next step ( $s$  is the number of profile such that  $r_s$  is the first profile that  $a_i I r_s$  or  $a_i R r_s$ ).

After first step some of alternatives can be not assigned to any of categories. In the other step, the assigned categories are being used to assign the unassigned ones. For each category we can define the set of alternatives assigned in the first step  $X_k$ , where  $k$  is a category number.

At first, the DM needs to calculate positive and negative distance for each alternative  $a_i$  basing on calculated  $s$  and  $s + 1$  from the first step as in given formulas:

$$d_k^+ = \sum_{x \in X_s} (\phi(a_i) - \phi(x)) \quad (58)$$

$$d_k^- = \sum_{x \in X_{s+1}} (\phi(x) - \phi(a_i)) \quad (59)$$

After calculating positive and negative distances, we can determine a total distance for the assignment using following formula:

$$d_k = \frac{1}{n_s} d_k^+ - \frac{1}{n_{s+1}} d_k^-, \quad (60)$$

where  $n_s$  means the number of alternatives assigned to category  $s$  in a first step of assignment in PromSort.

After computing the total distance for a given alternative  $a_i$  and profiles  $r_s$  and  $r_{s+1}$ , we can compare it with cut point parameter  $b$  which is given by the DM. While comparing cut point and total distance the following situations can be instantiated:



- If  $d_k > b$ , alternative  $a_i$  should be assigned to category  $C_{s+1}$ .
- If  $d_k < b$ , alternative  $a_i$  should be assigned to category  $C_s$ .
- If  $d_k = b$ , alternative  $a_i$  can be assigned to  $C_s$  or  $C_{s+1}$  according to additional parameter (preference of decision maker to assign alternatives to better class - *pref*) or just assigned to both of them in a imprecise assignment.

Note that cut point parameter represents the DM's preferences. When it is set to 0, the DM relies on  $d_k$ . When the parameter is lower than zero, the DM prefers to assign alternatives to better categories. If cut point is higher than 0, the DM prefers to assign alternatives to worse categories.

To prevent procedure from imprecise assignment, we incorporated one more parameter into it - preference of the DM to assign alternatives to better class (*pref*). One can say that this parameter is redundant - decision makers preferences are already used in cut point. However, adding additional decision maker's parameter is the easiest and probably the most intuitive way to achieve precise assignment when the DM's goal is to avoid imprecise assignments.

If the DM can get on the output of PromSort procedure some imprecise assignments (s)he can resign from providing an additional parameter - preference of decision maker to assign alternatives to better class (*pref*).

**Module M17. PromSort.** This module computes the assignments of given alternatives to categories using PromSort. Its structure is presented in Figure 2.18.

The module requires providing on inputs a list of alternatives ids ( $in_1$ ), list of categories ids with their ranks ordered from the worst to the best ( $in_2$ ), list of boundary profiles ( $in_3$ ) describing the categories given in input  $in_2$ . Decision maker can also provide limiting profiles in input  $in_3$ , module will not use profiles  $r_1$  and  $r_{K+1}$  then. To check the dominance of profiles module use also a table with profiles performances on each criterion ( $in_4$ ) and a list of criteria with their scales and preference threshold ( $in_5$ ). Decision maker has to provide also the positive ( $in_6$ ) and negative ( $in_7$ ) flows for each alternative from input  $in_1$  and for each boundary profile from input  $in_3$ .

PromSort module uses two parameters, which were described in section above. First of them - cut point ( $param_1$ ) is a decimal value greater or equal to -1 and lower or equal to 1. The second parameter - assign to a better class - is an additional information of decision maker preferences and as a logical value it can be true or false.

On the output module returns imprecise assignment from first step of assignment procedure ( $out_1$ ) and precise final assignment ( $out_2$ ).

## 2.6.2 Promethee Tri

Promethee Tri is a sorting method which uses central profiles as a description for categories and single criterion net flows to assign alternatives to categories.

There are two formulas for calculating criterion net flows for alternatives and for profiles [13]:

$$\phi_j(r_k) = \frac{1}{|R| - 1} \sum_{r \in R \setminus \{r_k\}} (P_j(r_k, r) - P_j(r, r_k)) \quad (61)$$

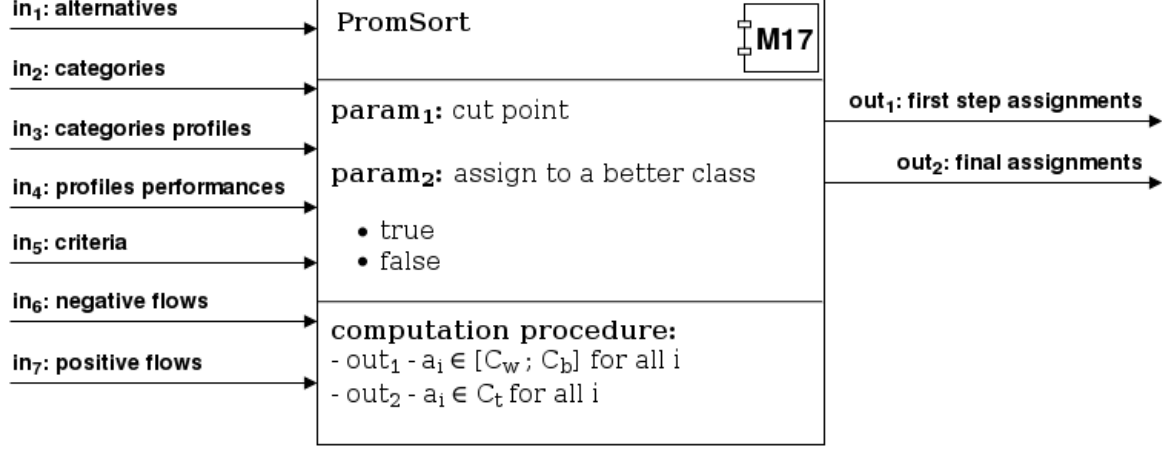


Figure 2.18: Structure of module M17 which computes PromSort class assignment

$$\phi_j(a_i) = \frac{1}{|R|} \sum_{r \in R} (P_j(a_i, r) - P_j(r, a_i)) \quad (62)$$

where:

$|R|$  is a cardinality of set of profiles,

$P_j(r_k, r)$  is a preference function between profile  $r_k$  and profile  $r$  on criterion  $j$ .

Let us focus on two formulas above. As one can see the criterion net flow is based on criterion preference function. Profiles are being compared to all other profiles, alternatives are being compared to all profiles. Thanks to it assignment of alternative do not depends on flows of another alternatives. The dividing by  $|R|$  or  $|R| - 1$  makes criterion net flows normalized.

To get assignments in Promethee Tri decision maker has to calculate for each profile  $r_k$  the deviation  $e(a_i, r_k)$  between alternative and each central profile [1] as shown in the following formula:

$$e(a_i, r_k) = \sum_{j \in J} |\phi_j(a) - \phi_j(r_k)| w_j \quad (63)$$

where:

$J$  is a set of criteria,

$w_j$  is a weight of criterion  $j$ .

After calculating the deviation, the DM can make the final assignment. Alternative is being assigned to the category described by profile with the smallest deviation (as shown in Formula 64).

$$a_i \in C_t \text{ if } e(a, r_t) = \min_{k=1, \dots, K} e(a, r_k) \quad (64)$$

**Module M18. PrometheeTri.** This module computes the assignments of given alternatives to categories using Promethee Tri method. Its structure is presented in Figure 2.19.

In inputs module reads alternatives ids ( $in_1$ ), categories ids with their order from the worst to the best ( $in_2$ ), profiles describing categories ( $in_3$ ), criteria ids ( $in_4$ ), and criteria weights ( $in_5$ ). In the six'th input there should be added preferences on criteria between each alternative and each profile and between each profile with each other profile.

As a first parameter PrometheeTri module reads a bool value ( $param_1$ ), by which decision maker can indicate that (s)he prefers to assign alternative to the better category when assignment is not clear (that can happen when two minimal deviations are equal). As a second parameter decision maker has to provide an information if (s)he wants to use deviation function in assignment with criterion flows (as in the Formula 63) or if (s)he wants to count it with global net flows (it can be achieved by removing the absolute value from Formula 63). In the first case parameter “use marginal value” ( $param_2$ ) needs to be set to true, in second option - to false.

As an output PrometheeTri module returns the precise assignments calculated in Promethee Tri algorithm.

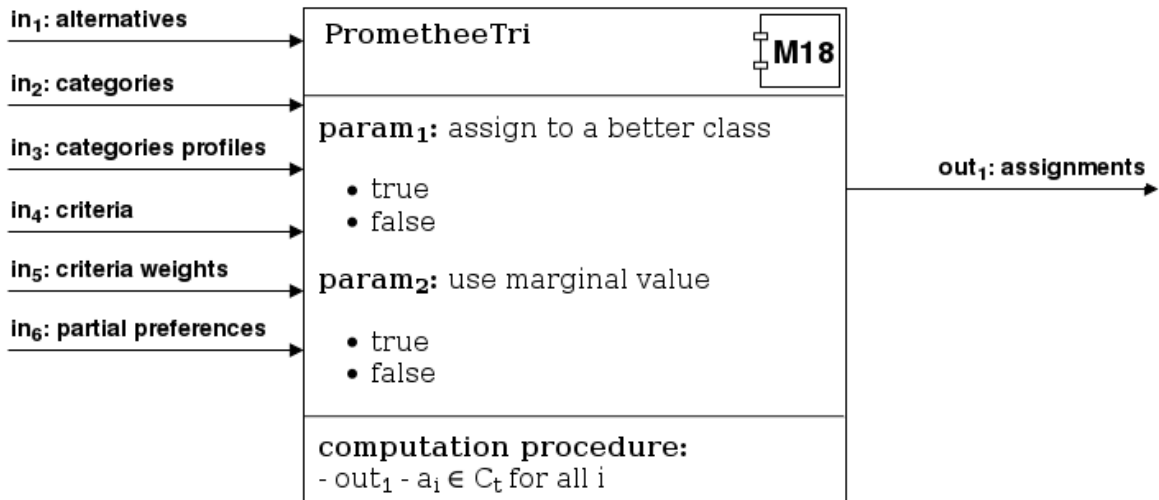


Figure 2.19: Structure of module M18 which computes Promethee Tri class assignment

### 2.6.3 FlowSort based on Promethee I

In FlowSort method based on Promethee I we can use either limiting, boundary or central profiles to describe categories. It uses both positive and negative flows computed in result of comparisons of each alternative and each profile with all the profiles. Assignment is calculated here separately for positive and negative flow. Both assignments can be different, so that final assignment is imprecise.

Regardless of the profiles type, they need to fulfill the dominance condition [26], which can be described with following formula:

$$\forall r_h, r_l \in R \text{ such that } h > l : \quad g_j(r_h) \geq g_j(r_l) \quad \forall j \geq 1 \wedge j \leq q \quad (65)$$

This condition is motivated by requirements of categories which need to be ordered. In given formula  $g_j(r_h)$  is the evaluation of profile  $r_h$  on criterion  $j$  and  $q$  is the number of criteria.

**Sorting with limiting profiles.** While working with limiting profiles we need to compare positive and negative flows of each alternative  $a \in A$  with all limiting profiles  $r \in R$ . The assignment bases on following equations:

$$C_{\phi^+}(a_i) = C_h \quad \text{if } \phi^+(r_h) < \phi^+(a_i) \leq \phi^+(r_{h+1}) \quad (66)$$

$$C_{\phi^-}(a_i) = C_h \quad \text{if } \phi^-(r_h) \geq \phi^-(a_i) > \phi^-(r_{h+1}) \quad (67)$$

As mentioned before  $C_{\phi^+}(a_i)$  can be different from  $C_{\phi^-}(a_i)$ . What is more  $C_{\phi^+}(a_i)$  can be preferred to  $C_{\phi^-}(a_i)$  or  $C_{\phi^-}(a_i)$  can be preferred to  $C_{\phi^+}(a_i)$ . When we assume that  $C_b$  is the better of them and  $C_w$  is the worse one we can assign alternative  $a_i$  to the categories  $[C_w; C_b]$ .

As one can notice profile  $r_1$  (which represents worst possible alternative) and  $r_{K+1}$  (which represents best possible alternative) are not necessarily needed as all alternatives which are worst then limiting profile  $r_2$  should be assigned to class  $C_1$ , and all alternatives better then limiting profile  $r_K$  should be assigned to class  $C_K$ . We can easily rewrite the assignment procedure to use it with boundary profiles instead of limiting ones.

**Sorting with boundary profiles.** When working with boundary profiles we can define for each alternative  $a_i$  the positive and negative flows assignment as follows:

$$C_{\phi^+}(a_i) = C_h \quad \text{if } \begin{cases} \phi^+(a_i) \leq \phi^+(r_h) & \text{for } h = 1 \\ \phi^+(a_i) > \phi^+(r_{h-1}) & \text{for } h = K \\ \phi^+(r_{h-1}) < \phi^+(a_i) \leq \phi^+(r_h) & \text{for } h > 1 \wedge h < K \end{cases} \quad (68)$$

$$C_{\phi^-}(a_i) = C_h \quad \text{if } \begin{cases} \phi^-(a_i) > \phi^-(r_h) & \text{for } h = 1 \\ \phi^-(a_i) \leq \phi^-(r_{h-1}) & \text{for } h = K \\ \phi^-(r_{h-1}) \geq \phi^-(a_i) > \phi^-(r_h) & \text{for } h > 1 \wedge h < K \end{cases} \quad (69)$$

$C_{\phi^+}(a_i)$  can be different from  $C_{\phi^-}(a_i)$  as while sorting with limiting profiles.

**Sorting with central profiles.** When central profiles are given as an input of FlowSort based on Promethee I flows the assignment rule need to change. As we are operating on positive and negative flows the assignment still is imprecise – we get separately the positive and negative assignment. As we do not know the limiting or boundary profiles we need to find another way of separation different classes. To do this we are comparing positive/negative flow of each alternative with the simple average of each pair of nearest profiles as in formulas below:

$$C_{\phi^+}(a_i) = C_h \quad \text{if } \begin{cases} \phi^+(a_i) \leq \frac{\phi^+(r_h) + \phi^+(r_{h+1})}{2} & \text{for } h = 1 \\ \frac{\phi^+(r_{h-1}) + \phi^+(r_h)}{2} < \phi^+(a_i) & \text{for } h = K \\ \frac{\phi^+(r_{h-1}) + \phi^+(r_h)}{2} < \phi^+(a_i) \leq \frac{\phi^+(r_h) + \phi^+(r_{h+1})}{2} & \text{for } h > 1 \wedge h < K \end{cases} \quad (70)$$

$$C_{\phi^-}(a_i) = C_h \quad \text{if} \quad \begin{cases} \phi^-(a_i) > \frac{\phi^-(r_h) + \phi^-(r_{h+1})}{2} & \text{for } h = 1 \\ \frac{\phi^-(r_{h-1}) + \phi^-(r_h)}{2} \geq \phi^-(a_i) & \text{for } h = K \\ \frac{\phi^-(r_{h-1}) + \phi^-(r_h)}{2} \geq \phi^-(a_i) > \frac{\phi^-(r_h) + \phi^-(r_{h+1})}{2} & \text{for } h > 1 \wedge h < K \end{cases} \quad (71)$$

As mentioned before both assignments  $C_{\phi^+}(a_i)$  and  $C_{\phi^-}(a_i)$  can be different, so we need to find better and worse of them to indicate optimistic and pessimistic assignment.

**Module M19. FlowSortI.** This module computes the assignments of given alternatives to categories using FlowSort procedure based on Promethee I flows. Its structure is presented in Figure 2.20.

The module requires providing on inputs a list of alternatives ids ( $in_1$ ) and a list of categories ids with their marks ( $in_2$ ). Categories should have marks from 1 to  $C$  where  $C$  is the number of categories. Category with mark 1 should be the worst category, category with mark  $C$  should be the best one. In input  $in_3$  decision maker need to provide a description of categories with either central, limiting or boundary profiles. To check the dominance of given profiles module requires to provide a table with performances of profiles ( $in_4$ ) and a list of criteria with their scales ( $in_5$ ). To compute the categories assignments module needs also on inputs negative and positive flows computed for alternatives and profiles ( $in_6$  and  $in_7$ ). Decision maker ought to provide them for all alternatives and profiles given in inputs  $in_1$  and  $in_3$ . Otherwise module returns error message. Decision maker needs also to provide as a parameter information of profiles used in calculations ( $param_1$ ). When (s)he wants to use central profiles (s)he needs to choose option “central profiles”. When (s)he wants to use boundary or limiting profiles (s)he need to choose “boundary profiles” option.

On the output module returns the imprecise assignment of each alternative to pessimistic and optimistic category ( $out_1$ ).

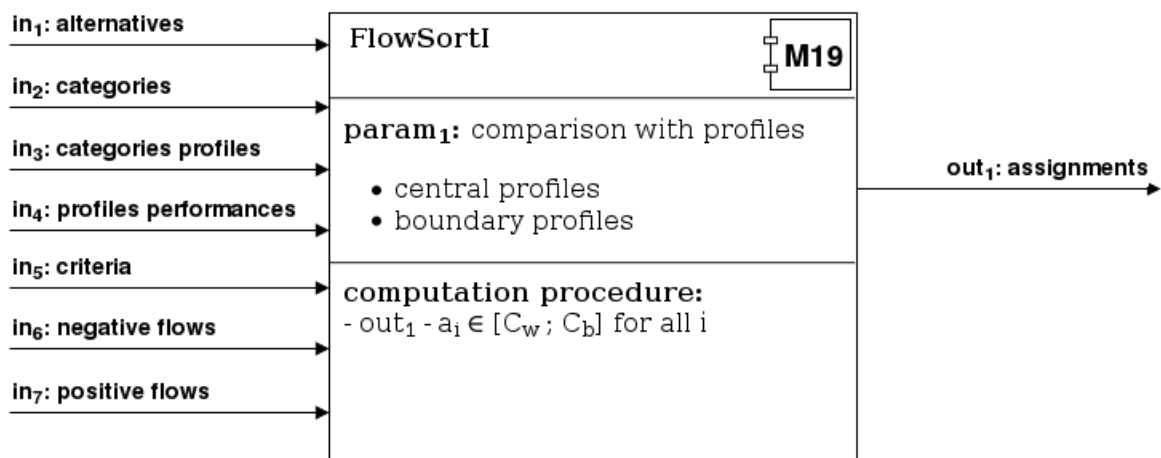


Figure 2.20: Structure of module M19 which computes FlowSort based on Promethee I class assignment

#### 2.6.4 FlowSort based on Promethee II

In FlowSort method based on Promethee II flows we can use either limiting, boundary or central profiles as in described before FlowSort method based on Promethee I flows. The difference is that Promethee II flows are net flows, which provides the precise assignment in FlowSort procedure [26].

As mentioned before in FlowSort based on Promethee I flows we need to check the domination of input profiles. When we are using Promethee II flows the condition does not change and is the same as in Formula 65.

**Sorting with limiting profiles.** The assignment rule for limiting profiles is given in Formula 72

$$C_\phi(a_i) = C_h \quad \text{if } \phi(r_h) < \phi(a_i) \leq \phi(r_{h+1}) \quad (72)$$

Given rule is analogous to assignment with the limiting profiles in FlowSort based on Promethee I flows. As net flows combines positive and negative flows the assignment will also be a result of that combination and will not be worse from the pessimistic assignment nor better from optimistic assignment from Promethee I based approach [26].

**Sorting with boundary profiles.** As we did before for FlowSort based on Promethee I flows we can easily rewrite assignment procedure for boundary profiles in Promethee II based approach. The assignment rule for boundary profiles is given in Formula 73.

$$C_\phi(a_i) = C_h \quad \text{if } \begin{cases} \phi(a_i) \leq \phi(r_h) & \text{for } h = 1 \\ \phi(r_{h-1}) < \phi(a_i) & \text{for } h = K \\ \phi(r_{h-1}) < \phi(a_i) \leq \phi(r_h) & \text{for } h > 1 \wedge h < K \end{cases} \quad (73)$$

**Sorting with central profiles.** When working with central profiles the assigned rule can be defined as follows:

$$C_\phi(a_i) = C_h \quad \text{if } \begin{cases} \phi(a_i) \leq \frac{\phi(r_h) + \phi(r_{h+1})}{2} & \text{for } h = 1 \\ \frac{\phi(r_{h-1}) + \phi(r_h)}{2} < \phi(a_i) & \text{for } h = K \\ \frac{\phi(r_{h-1}) + \phi(r_h)}{2} < \phi(a_i) \leq \frac{\phi(r_h) + \phi(r_{h+1})}{2} & \text{for } h > 1 \wedge h < K \end{cases} \quad (74)$$

The assignment is analogous with the imprecise assignment from Promethee I based FlowSort as in the procedure for limiting profiles. It is caused by a way of calculating net flows.

**Module M20. FlowSortII.** This module computes the assignments of given alternatives to categories using FlowSort procedure based on Promethee II flows. Its structure is presented in Figure 2.21.

Inputs of this module are similar to inputs from module FlowSortI (presented in Section 2.6.3). Decision maker needs to provide a list of alternatives ids (*in1*), a list of categories ids with their ranks ordered from the worst to the best (*in2*), a list of

either limiting, boundary or central profiles defining the categories ( $in_3$ ), a table with performances of profiles ( $in_4$ ), a list of criteria and their scales ( $in_5$ ) and net flows of all alternatives and profiles ( $in_6$ ). (S)He needs to provide also as a parameter ( $param_1$ ) the type of input profiles. When (s)he wants to use boundary or limiting profiles (s)he need to choose “boundary profiles” option, when (s)he wants to use central profiles (s)he need to choose “central profiles” option.

As in FlowSortI module the dominance of profiles is being checked.

On the output module returns precise assignment of each alternative to proper category.

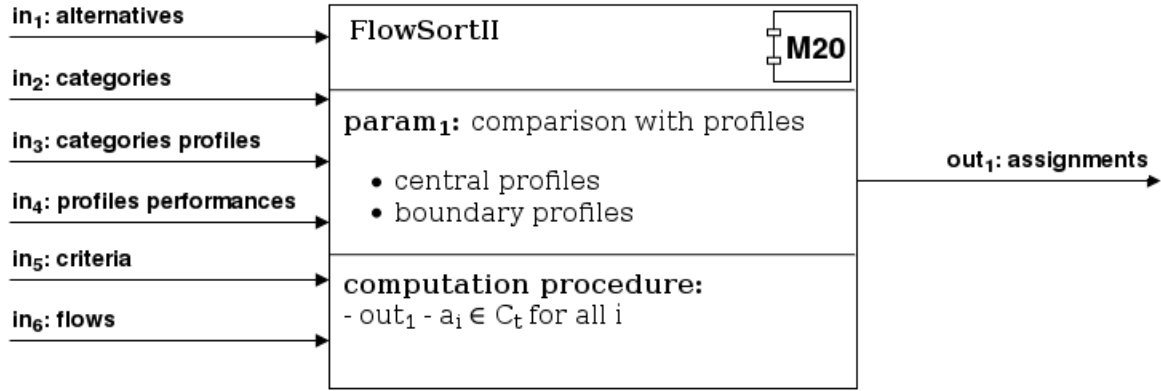


Figure 2.21: Structure of module M20 which computes FlowSort based on Promethee II class assignment

### 2.6.5 FlowSort GDSS

FlowSort GDSS is a sorting method from Promethee family which enables to assign alternatives to predefined, ordered categories taking into account preferences of multiple decision makers. It was proposed by Francesco Lolli et al. in 2015 [21].

Alternatives and categories are the only common data for all decision makers. Each of them has his own description of categories with globally chosen type of profiles. Chosen profiles can be either central, boundary or limiting. Each decision makers has also his own weight which measures his impact in the final assignment.

Profiles given by decision makers need to fulfill the dominance condition which can be described by the following formula:

$$r_k^{t,j} > r_{k+1}^{s,j}; \quad \forall r_k^{t,j}, r_{k+1}^{s,j} \in R^j, \quad \forall j = 1, \dots, J \quad \text{and} \quad \forall t, s = 1, \dots, T \quad (75)$$

where:

$J$  is a set of criteria,

$T$  is a set of decision makers,

$R^j$  is a set of profiles preferences on criterion  $j$ ,

$r_k^{t,j}$  is a performance of profile number  $k$  on criterion  $j$  given by decision maker  $t$ .

At the beginning FlowSort GDSS requires from decision maker calculating the net flow  $\phi_j(a_i)$  of each alternative  $a_i \in A$  on each criterion  $j \in J$  using following formula:

$$\phi_j(a_i) = \frac{1}{|R^j|} \sum_{r_k^{t,j} \in R^j} [P_j(a_i, r_k^{t,j}) - P_j(r_k^{t,j}, a_i)] \quad (76)$$

where:

$J$  is a set of criteria,

$T$  is a set of decision makers,

$R^j$  is a set of profiles preferences on criterion  $j$ ,

$|R^j|$  is a cardinality of set of profiles preferences on criterion  $j$ ,

$r_k^{t,j}$  is a performance of profile number  $k$  on criterion  $j$  given by decision maker  $t$ ,

$P_j(a_i, r_k^{t,j})$  is a preference of alternative  $a_i$  to profile  $r_k^{t,j}$  on criterion  $j$ .

After calculating net flows of alternative  $a_i$  on each criterion decision maker can calculate the global net flow of this alternative as a weighted sum of net flows on each criterion:

$$\phi(a_i) = \sum_{j=1}^J w_{g_j} \phi_j(a_i) \quad (77)$$

where:

$w_{g_j}$  is a weight of criterion  $j$ ,

$\phi_j(a_i)$  is a net flow of alternative  $a_i$  on criterion  $j$  computed in Formula 76.

As one can see global net flow  $\phi(a_i)$  can be calculated also as a average of net flows of alternative  $a_i$  calculated separately for every decision maker. In this case each decision maker calculates his own net flow of alternative  $a_i$  as a result of comparisons with his own profiles  $r_h \in R$ . After that step local net flows of all decision makers are being added and divided by number of decision makers. That approach enables us divide whole assignment process to one more separated step which can be done by each decision maker by his own.

FlowSort GDSS requires from decision makers also calculating criterion net flow of each profile with influence of each alternative  $a_i \in A$ . We will mark it as  $\phi_{j,i}(r_\kappa^{\tau,j})$  and calculate as following:

$$\phi_{j,i}(r_\kappa^{\tau,j}) = \frac{1}{|R^j| + 1} \left\{ \sum_{r_k^{t,j} \in R^j} [P_j(r_\kappa^{\tau,j}, r_k^{t,j}) - P_j(r_k^{t,j}, r_\kappa^{\tau,j})] + [P_j(r_\kappa^{\tau,j}, a_i) - P_j(a_i, r_\kappa^{\tau,j})] \right\} \quad (78)$$

where:



$J$  is a set of criteria,

$T$  is a set of decision makers,

$\tau$  is a chosen decision maker number,

$\kappa$  is a chosen profile number,

$R^j$  is a set of profiles preferences on criterion  $j$ ,

$|R^j|$  is a cardinality of set of profiles preferences on criterion  $j$ ,

$r_h^{t,j}$  is a performance of profile number  $h$  on criterion  $j$  given by decision maker  $t$ ,

$P_j(r_\kappa^{\tau,j}, r_k^{t,j})$  is a preference of profile  $r_\kappa^{\tau,j}$  to profile  $r_k^{t,j}$  on criterion  $j$ .

The Formula 78 is being calculated for every pair of alternative  $a_i \in A$  and profile  $r_k^{t,j} \in R^j$  on each criterion  $j \in J$ . After that we can calculate global net flows  $\phi_i(r_\kappa^\tau)$  for each profile as a weighted sum of net flows on criteria:

$$\phi_i(r_\kappa^\tau) = \sum_{j=1}^J w_{g_j} \phi_{j,i}(r_\kappa^{\tau,j}) \quad (79)$$

where:

$w_{g_j}$  is a weight of criterion  $j$ ,

$\phi_{j,i}(r_\kappa^{\tau,j})$  is a net flow of profile  $r_\kappa^{\tau,j}$  on criterion  $j$  computed in Formula 78.

After calculating all needed net flows we can move to the next step - assignment of alternatives. There are two types of assignments in FlowSort GDSS - unanimous and non unanimous. The unanimous assignment takes place when all decision makers assign alternative  $a_i$  to the same category  $C_k$ . The non unanimous assignment is being used when decision makers preferences are not consistent. There are different rules of assignment for sorting with limiting profiles and sorting with central profiles.

**Sorting with boundary profiles.** The unanimous assignment rule for sorting with boundary profiles for each decision maker  $t$  is described below:

$$C_t(a_i) = C_k \quad \text{if} \quad \begin{cases} \phi(a_i) \leq \phi_i(r_k^t) & \text{for } k = 1 \\ \phi_i(r_{k-1}^t) < \phi(a_i) & \text{for } k = K \\ \phi_i(r_{k-1}^t) < \phi(a_i) \leq \phi_i(r_k^t) & \text{for } k > 1 \wedge k < K \end{cases} \quad (80)$$

If for each  $t \in T$  assignment  $C_t$  is the same it is the final assignment. If there are some decision makers with different assignments the final assignment will be made with non unanimous assignment rule. What is important thanks to use dominance condition on profiles (Formula 75) after the first step of assignment only two different assignments of decision makers can be taken into account - to category  $C_k$  or  $C_{k+1}$ .

In non unanimous assignment rule distances functions  $d_i(k)$  and  $d_i(k + 1)$  are being calculated using following formulas:

$$d_i(k) = \sum_{t:\phi(a_i) < \phi_i(r_k^t)} w_{d_t} [\phi(a_i) - \phi_i(r_k^t)] \quad (81)$$

$$d_i(k + 1) = \sum_{s:\phi(a_i) \geq \phi_i(r_k^s)} w_{d_s} [\phi_i(r_{k+1}^s) - \phi(a_i)] \quad (82)$$

where:

$w_{d_t}$  is a weight of decision maker  $t$ ,

$k$  is a number of worse category from first step,

$k + 1$  is a number of better category from first step,

$\sum_{t:\phi(a_i) \geq \phi_i(r_k^t)}$  means, that we are making a sum of profiles  $r_k^t$  only for profiles of decision makers who have chosen a worse assignment in a first step,

$\sum_{s:\phi(a_i) < \phi_i(r_k^s)}$  means, that we are making a sum of profiles  $r_k^s$  only for profiles of decision makers who have chosen a better assignment in a first step.

After calculating the distances the only thing to do is to compare  $d_i(k)$  with  $d_i(k + 1)$  and make a final assignment. In final assignment there are 3 possibilities of an assignment:

- If  $d_i(k + 1) > d_i(k)$  assign alternative  $a_i$  to category  $C_k$ .
- If  $d_i(k + 1) < d_i(k)$  assign alternative  $a_i$  to category  $C_{k+1}$ .
- If  $d_i(k + 1) = d_i(k)$  assign alternative  $a_i$  to category  $C_k$  or  $C_{k+1}$  according to additional global parameter - preference of decision maker to assign alternative to a better category.

**Sorting with central profiles.** While working with central profiles the unanimous assignment rule for each decision maker  $t \in T$  is defined by another formula:

$$C_t(a_i) = C_k \quad \text{if } |\phi_i(r_k^t) - \phi(a_i)| < |\phi_i(r_h^t) - \phi(a_i)|, \forall h = 1, \dots, K, \quad h \neq k \quad (83)$$

As one can see, the assignment for decision maker  $t$  is being made here by finding a profile with the nearest flow for decision maker  $t$ . When assignments of decision makers are equal the assignment is final. Otherwise there need to be performed a non unanimous assignment.

As in the assignment with boundary profiles, decision makers can have only two different assignments in this step. To find a final assignment decision makers need to count the distance functions as follows:

$$d_i(k) = \sum_{t \in T_k} w_{d_t} |\phi_i(r_k^t) - \phi(a_i)| \quad (84)$$

$$d_i(k+1) = \sum_{s \in T_{k+1}} w_{d_s} |\phi_i(r_{k+1}^s) - \phi(a_i)| \quad (85)$$

where:

$w_{d_t}$  is a weight of decision maker  $t$ ,

$k$  is a number of worse category from first step,

$k+1$  is a number of better category from first step,

$T_k$  is a set of decision makers who assigned alternative  $a_i$  to a worse category,

$T_{k+1}$  is a set of decision makers who assigned alternative  $a_i$  to a better category.

After calculating the distance functions decision makers can make a final assignment using same rule as while using boundary profiles:

- If  $d_i(k+1) > d_i(k)$  assign alternative  $a_i$  to category  $C_k$ .
- If  $d_i(k+1) < d_i(k)$  assign alternative  $a_i$  to category  $C_{k+1}$ .
- If  $d_i(k+1) = d_i(k)$  assign alternative  $a_i$  to category  $C_k$  or  $C_{k+1}$  according to additional global parameter - preference of decision maker to assign alternative to a better category.

**Module M21. FlowSortGDSS.** This module computes the assignments of given alternatives to categories using FlowSort GDSS procedure. Its structure is presented in Figure 2.22.

On the input module reads global ids of alternatives ( $in_1$ ), global ids of categories with their ranks ordered from the worst to the best as  $in_2$ . It requires also to provide lists of profiles describing the categories for all (at least two) decision makers ( $in_{3-4}$  are required,  $in_{5-12}$  are optional). To check the dominance condition of profiles decision maker need to provide list of criteria with their scales ( $in_{13}$ ). Module also requires providing flows of alternatives of all (at least two) decision makers (mandatory  $in_{14-15}$  and optional  $in_{16-23}$ ). Performances of profiles ( $in_{24-33}$ ) for each decision maker (at least two of them) are also needed on the input. To compute global flows for profiles module requires providing also preferences between alternatives and profiles for each decision maker ( $in_{34-35}$  and  $in_{36-43}$  optionally) and list of net flows for profiles calculated by comparisons with all decision makers profiles ( $in_{44}$ ). As the number of decision makers is known from the beginning, the number of used inputs in structures given separately by each of decision maker ( $in_{3-12}$ ,  $in_{14-23}$ ,  $in_{24-33}$  and  $in_{34-43}$ ) need to be the same and equal to number of decision makers. The module requires providing data from at least 2 and at most 10 decision makers.

As a first parameter module reads global type of profiles used by decision makers ( $param_1$ ). Decision makers can use either central profiles or boundary profiles. When they need to provide limiting profiles they can do it using “boundary” option - module will use only needed profiles. As a second parameter ( $param_2$ ) decision makers need to provide their preference of assign to a better category in case of ambiguous assignment in

the step of non unanimous assignment. As the last 10 parameters ( $param_{3-12}$ ) decision makers need to provide their own weights. If number of decision makers is lower then 10 they should put “0” to weights of absent decision makers.

On the output module returns imprecise assignments after an unanimous assignment step ( $out_1$ ) and precise final assignment ( $out_2$ ).

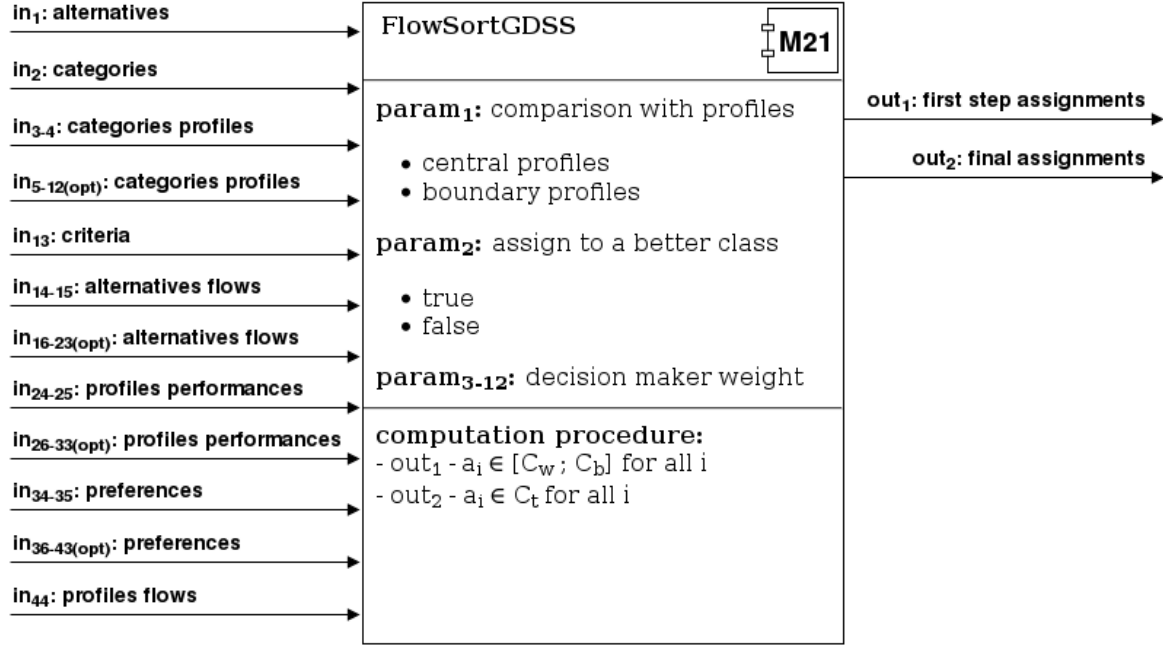


Figure 2.22: Structure of module M21 which computes FlowSort GDSS class assignment

## 2.6.6 Group Class Acceptabilities

In this section, we present an approach that supports finding a common solution in terms of a sorting problem. The starting point is the set of assignments to classes for every alternative defined as upper ( $U_k(a_i)$ ) and lower bound ( $L_k(a_i)$ ) (there is also a possibility that upper bound = lower bound, which means that assignment is precise). Then one should iterate through all the classes and alternatives and firstly express if current decision maker ( $k$ ) assigns this alternative ( $a_i$ ) to the class (as it is shown in Equation 86).

$$E_k(a_i, C_x) = \begin{cases} 1 & \text{if } C_x \in [L_k(a_i), U_k(a_i)] \\ 0 & \text{if } C_x \notin [L_k(a_i), U_k(a_i)] \end{cases} \quad (86)$$

The next step consists in the aggregation of all individual values by the Equation 87.

$$E(a_i, C_x) = \frac{\sum_{k=1}^K E_k(a_i, C_x)}{K} * 100\% \quad (87)$$

Such an approach has some disadvantages. For example, when two separated classes have bigger value than the class between them. The cure for that is “unimodal” version

shown in Equation 88. It takes into account both classes better and weaker than current and then assign value basing on this knowledge [8].

$$E'(a_i, C_x) = \begin{cases} E(a_i, C_x) * 100\% & \text{if } x = 1 \vee x = h \\ \max\{E(a_i, C_x), \min\{\max_{y < x} E(a_i, C_y), \max_{y > x} E(a_i, C_y)\}\} * 100\% & \text{if } 1 < x < h \end{cases} \quad (88)$$

**Module M22. GroupClassAcceptabilities.** Structure of this module is presented in Figure 2.23. It requires providing list of alternatives ( $in_1$ ), list of categories ( $in_2$ ) where are not only shown all the available classes but also their numeric equivalents to make it possible to compare the classes. Last inputs are assignments of each alternative to the classes defined generally as imprecise assignment which means there is lower and upper bound (it is possible that they are equal) -  $in_{3-12}$ . There are two outputs from this module. First one presents alternatives support, calculated with the usage of Equation 87 ( $out_1$ ). The second output represents unimodal alternatives support, computed accordingly to Equation 88 ( $out_2$ ).

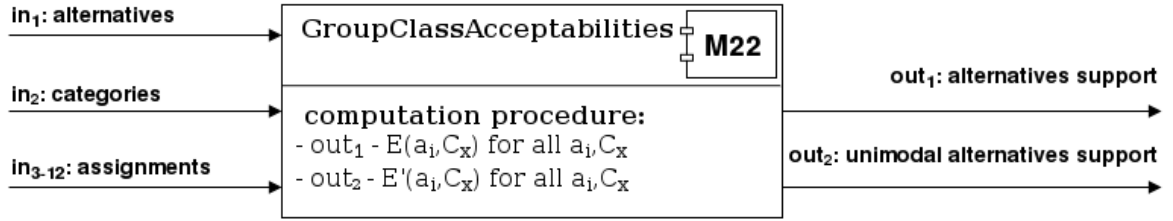


Figure 2.23: Structure of module M22 which computes Group Class Acceptabilities.

## 2.7 Choice Problem

The problem of choosing is about how to find the most preferred alternatives from the perspective of the DM. The algorithms applied to solve this problem can also be used to reduce the number of acceptable alternatives. In this section we present the method which solves this problem. At the same time we present the module that implements this method (M23).

### Promethee V

Promethee V [4] is applied to find a subset of the most preferred alternatives with respect to linear constraints by constructing and solving linear programming problem. The Promethee V procedure consists of two following steps:

- STEP 1: Computation of the Promethee II flows for all alternatives (see Section 2.5.2)
- STEP 2: Solving the following linear programming model:

$$\max\left\{\sum_{i=1}^k \phi(a_i)x_i\right\} \quad (89)$$

$$\sum_{i=1}^n \lambda_{p,i}x_i \circ \beta_p, \quad (90)$$

$$x_i \in \{0, 1\}, i = 1, 2, \dots, n, \quad (91)$$

where symbol  $\circ$  stands for  $\geq, \leq$  or  $=$ .

Objective function has boolean variables which represent the selection of each alternative and flows of each one as coefficients. The higher the flow, the better the alternative. If a boolean variable  $x_i$  is equal to 1 it means that set of results contains alternative  $a_i$ . Linear constraints are defined by the DM and may include various business constraints such as: cardinality, budget, investment, time, etc.

**Module M23. PrometheeV.** The structure of this module is presented in Figure 2.24. It solves binary linear programming problem given by the DM. The result of this module is a set of alternatives values in which value 0 represents alternative not included in the selected subset and value 1 represent alternative incorporated in the optimal set ( $out_1$ ). This module requires the user to specify an alternatives to consider ( $in_1$ ), net outranking flows ( $in_2$ ) and linear constraints ( $in_3$ ). This outranking flows can be calculated by module M12 or any function corresponding to the net flows from Promethee II.

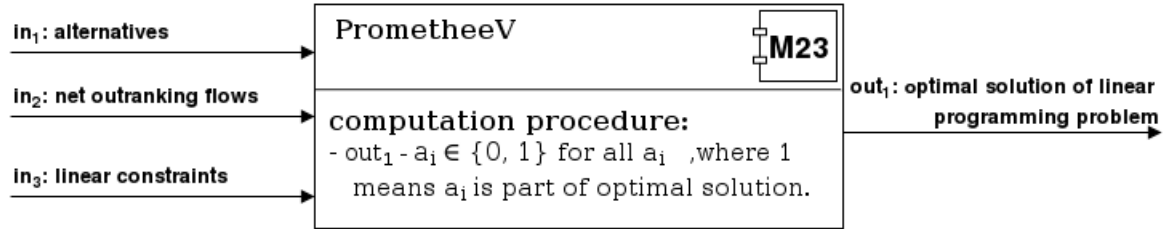


Figure 2.24: Structure of module M23 which computes results of the Promethee V method.

## 2.8 Clustering

In case of sorting problems, the DM must define groups before applying a sorting algorithm but in many cases (s)he knows only the number of required groups. Then, (s)he should use a clustering algorithm. Main problem of clustering is assigning similar or indifferent alternatives into groups, which are called clusters. These groups may be ordered or not. In this section, we focus on different methods of clustering. We present a few modules we have implemented. They are numbered for easy identification (M24 - M26).

In this section we use the following notation. Let  $A = \{a_1, a_2, \dots, a_n\}$  be a set of alternatives and  $C = \{c_1, c_2, \dots, c_k\}$  be a set of clusters. Each alternative must be

assigned only to one cluster, but clusters may aggregate many alternatives:

$$A = \cup_{i=1, \dots, k} C_i \quad (92)$$

$$\forall_{i \neq j} C_i \cap C_j = \emptyset \quad (93)$$

In many cases ordered clusters are needed:

$$C_1 > C_2 > \dots > C_k \quad (94)$$

where symbol  $>$  in  $C_i > C_j$  means that cluster  $C_i$  has a lower rank than  $C_j$ . In ordered clusters (the lower rank, the better cluster), if alternative  $a_i$  is assigned to a cluster with lower rank than alternative  $a_j$ ,  $a_i$  is considerably better than alternative  $a_j$ .

### 2.8.1 Ordered clustering

Ordered Clustering [10] is a method of grouping alternatives into K-ordered group, where number of groups (K) is defined by the DM. This method, based on preference matrix, builds directed, acyclic graph whose longest path equal to K-1. This graph allows us to extract the required number of groups in K steps (see point 6. in algorithm).

Algorithm:

1. Define  $\pi_{i,j}$  as a preference table and create a graph with all alternatives as vertices and without arrows.
2. Choose  $\max_{i,j} \{\pi_{i,j}\}$ , if this value is equal to zero then move onto step 6.
3. Add arrows to the graph between node i and j
4. Check if graph has a cycle or some path longer than K-1; if it is true, delete an arrow between node i and j
5. Set  $\pi_{i,j} = 0$  and go to step 2.
6. Determine first cluster by finding all nodes which have input degree equal to 0, then delete these nodes from the graph. Each next cluster is determined like the first one as long as there are some nodes in the graph.

The upper bound of algorithm's iterations is equal to  $n^2 - n$ , where  $n$  is the number of alternatives.

**Module M24. OrderedClustering.** The structure of this module is presented in Figure 2.25. It groups alternatives into clusters using Ordered Clustering method based on the preference matrix ( $in_2$ ) and the set of alternatives ( $in_1$ ). The result of this module is the ordered alternatives' sets in which each cluster has at least one alternative ( $out_1$ ). Lower set id represents a better cluster. The user needs to specify the number of clusters, which is the parameter of this module ( $param_1$ ).

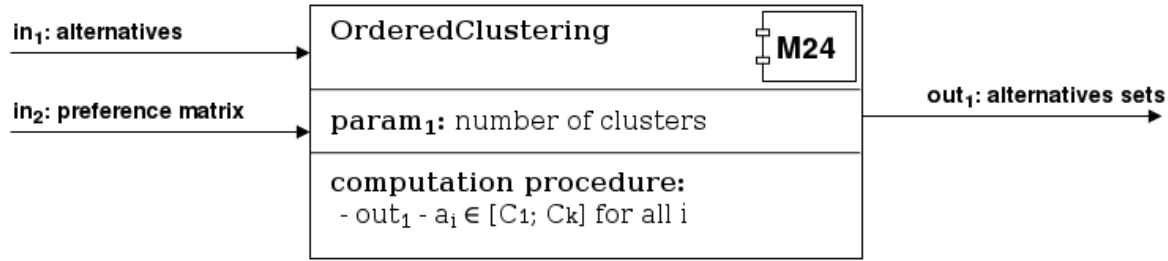


Figure 2.25: Structure of module M24 which computes the results of Ordered Clustering method.

### 2.8.2 Promethee II Ordered Clustering

It is a clustering method based on FlowSort II and k-means algorithm, which is used to assign alternatives to clusters [13].

Algorithm:

1. Decision maker (DM) defines the number of clusters  $K$ .
2. Generate random  $K$  central profiles.
3. Assigns each alternative to one cluster using the FlowSort method (see Section 2.6.4).
4. Update performances of the central profile  $r_h$  through calculating an average of performances of alternatives  $a_i$  contained in the  $C_h$  cluster.
5. Repeat the procedure, from step 3, until a membership of alternatives no longer changes.

Being generated at random, the central profiles must be dominated, which means that if the  $r_i$  profile dominates  $r_j$ , then  $r_i$  is better than  $r_j$ , according on all criteria. In this way, the resulting groups are ordered. The only task of FlowSort is to assign alternatives to groups.

**Module M25. PrometheeIIOrderedClustering.** The structure of this module is presented in Figure 2.26. It combines k-means method (main flow of algorithm) with FlowSort (used to assign alternatives to clusters). This module computes the ordered alternatives sets ( $out_1$ ), where the lower set id indicates better (more preferred) cluster. It requires the user to specify an alternatives to consider ( $in_1$ ), a set of criteria ( $in_2$ ), weights of criteria ( $in_3$ ) and preference indices ( $in_4$ ).

### 2.8.3 Promethee Cluster

Promethee Cluster [13] combines k-means algorithm (used to group alternatives into clusters) with Promethee Tri (used to establish the difference between alternatives). When implementing this module, we used a deviation formula different than in the Promethee Tri method (see Section 2.6.2). This is presented in Equation 95.



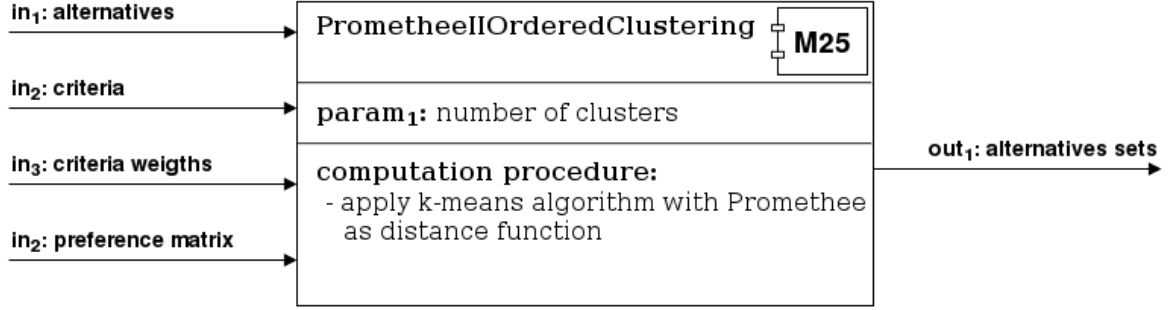


Figure 2.26: Structure of module M25 which computes the results of Promethee II Ordered Clustering method.

$$e_p(a, r_h) = \left( \sum_{j \in J} |\phi_j^R(a) - \phi_j(r_h)|^p w_j \right)^{1/p} \quad (95)$$

where:

- $R = \{r_1, r_2, \dots, r_k\}$  is a set of central profiles,
- $\phi_j^R$  is a flow calculated by using a set which contains an alternative  $a$  and the set  $R$ ,
- $\phi_j$  is a flow calculated on set  $R$ .

As opposed to the original formula, we expanded it with coefficient  $p$  and calculated flows based on different data set. Deviation  $e_k(a, r_h)$  is a measure of the difference between alternatives. The smaller  $e_k(a, r_h)$ , the more  $a$  and  $r_h$  have in common (e.g., if  $e_k(a, r_h) = 0$  it means that  $a$  and  $r_h$  are the same).

Algorithm:

1. The DM defines the number of clusters  $k$  and  $p$  coefficient.
2. Central profiles are selected among the existing alternatives.
3. Deviation between all alternatives and central profiles is calculated.
4. If deviation is minimal, then the alternative  $a_k$  alternative is assigned to a category  $C_l$ :

$$a_i \in C_{l_i} \text{ if } e_q(a_i, r_{l_i}) = \min_{h=1, \dots, k} \{e_q(a_i, r_h)\} \quad (96)$$

5. Central profiles are redefined by calculating median of each alternatives on each criterion:

$$g_j(r_h) = \text{median}\{g_j(a_i), a_i \in C_h\} \quad (97)$$

6. Steps 3-5 are repeated until cluster membership no longer changes.

**Module M26. PrometheeCluster.** Structure of this module is presented in Figure 2.27. It groups alternatives into clusters using k-means algorithm and Promethee Tri evaluation of distance between alternatives and central profiles. The result of this module are unordered sets of alternatives, where each cluster has at least one alternative ( $out_1$ ). This module requires the user to specify an alternatives to consider ( $in_1$ ), a set of criteria ( $in_2$ ), weights of criteria ( $in_3$ ) and preference indices ( $in_4$ ).

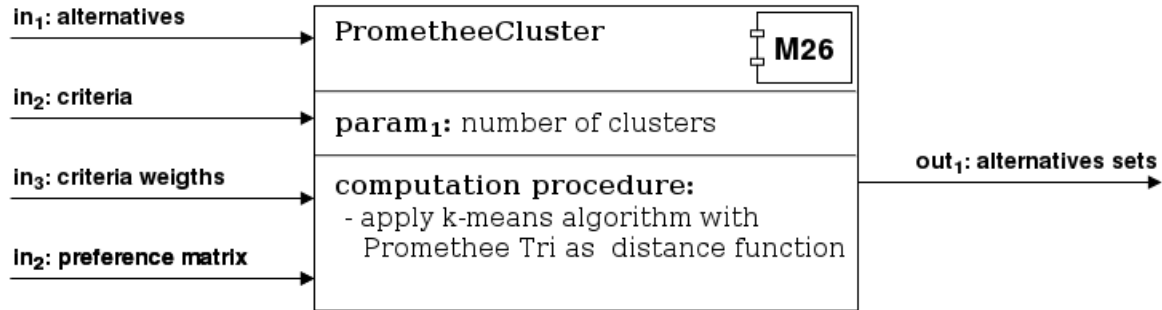


Figure 2.27: Structure of module M26 which computes Promethee Cluster method.

## 2.9 Visualization

In this section we describe two graphical modules (M27 and M28), which visualize assignments of alternatives to categories.

### 2.9.1 Graphical class assignment

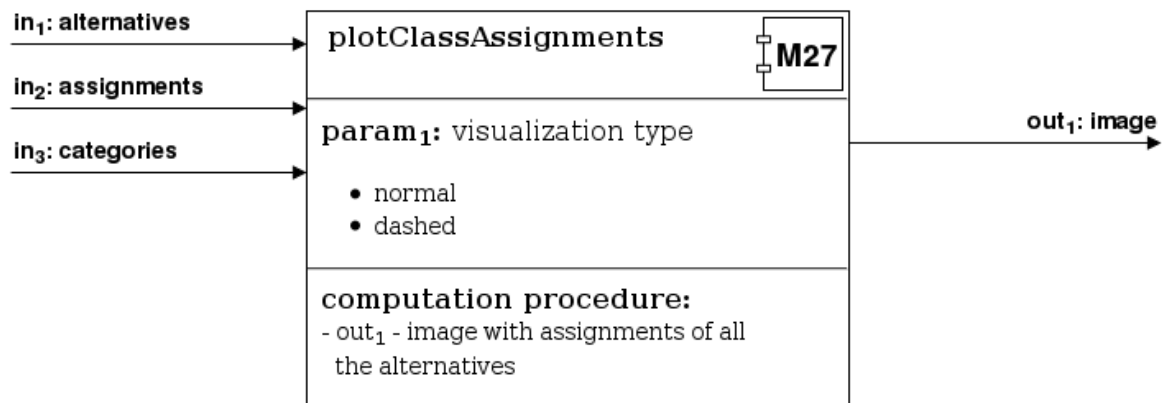


Figure 2.28: Structure of module M27 which visualizes class assignments with an image.

**Module M27. plotClassAssignments.** Structure of this module is presented in Figure 2.28. It reads alternatives ids ( $in_1$ ), alternatives assignments ( $in_2$ ) and ids of categories to which alternatives can be assigned with categories ranks ordered from the worst to the best ( $in_3$ ). The input alternatives assignments for this module can be either precise or imprecise. As a parameter module requires providing information of required

visualization type ( $param_1$ ). There are two possible types of visualization - “normal”, and “dashed”. The chosen one is being returned on the output ( $out_1$ ). Example outputs are shown on the Figure 2.29.

As one can see the “normal” visualization presents for each alternative all categories. Only the interval of categories form the pessimistic alternative assignment to the optimistic one is colored in gray. In “dashed” visualization categories has its spaces. For each alternative the assignment interval is selected by a rounded shape.

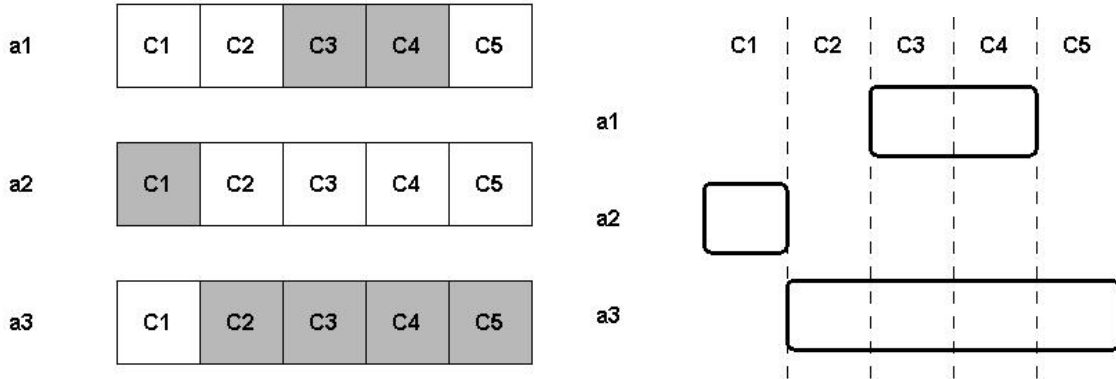


Figure 2.29: Outputs from plotClassAssignments module. On the left “normal” visualization, on the right “dashed” one.

## 2.9.2 Latex table of class assignment

**Module M28. LaTeXClassAssignments.** Structure of this module is presented in Figure 2.30. It reads alternatives ids ( $in_1$ ), alternatives assignments ( $in_2$ ) and ids of categories to which alternatives can be assigned with categories ranks ordered from the worst to the best ( $in_3$ ). The input alternatives assignments for this module can be either precise or imprecise. As the output module returns the LaTeX table visualizing the alternatives assignments ( $out_1$ ). The example output is shown in Table 2.2.

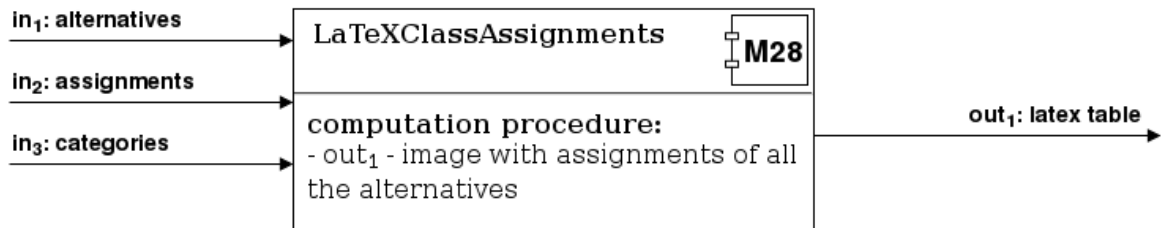


Figure 2.30: Structure of module M28 which visualizes class assignments with a latex table.

Table 2.2: Example LaTeX table generated for alternatives with imprecise assignments.

<b>Alternative</b>	<b>Lower class</b>	<b>Upper class</b>
a1	C3	C4
a2	C1	C1
a3	C2	C5

## 3 Construct Your Own Promethee Method in diviz

### 3.1 diviz

*diviz* is a workbench used to design, execute and share complex MCDA methods and experiments [25]. It is an open-source software which allows to combine work of many researchers. The platform provides an integrated environment for creating MCDA workflows from elementary MCDA components.

*diviz* has been constructed in a client-server infrastructure. The client is a desktop Java application which allows to design MCDA workflows, enter data, and analyze results with a simple and comfortable graphical user interface. The distant servers are used to execute workflows and to enable cooperation of all components. All computation modules, visualization and reporting tools are available in *diviz* via XMCDA web-services. To communicate with each other they use an XMCDA standard. It is an XML-based format which represents the MCDA concepts such as alternatives, criteria, outranking flows and so on, using general data structures coded with appropriate tags and attributes.

### 3.2 Modules implementation

The collection of 28 modules created within the project has been made available on *diviz* platform. Their functionality was described in Section 2. They represent methods for deriving weights of criteria, computing preference degrees and outranking flows as well as for constructing recommendation in function of the ranking, choice, sorting and clustering problems. They also support visualization of the results.

All presented modules are written in Java 8 which is a simple high-level, object-oriented, general-purpose programming language. The *diviz* platform offers a full support for this language. A library dedicated for XMCDA is also written in Java. In some modules, we used external libraries. This has been explained in description of these modules.

### 3.3 Workflow Design

The design of decision analysis workflow is provided by user-friendly and intuitive graphical user interface. Each algorithm available on *diviz* is represented by a rectangular box. Each input and output of a module can be linked to a file (all files used by *diviz* are unified XML files) or another algorithm provided that they accept the same data type. User can add modules and files to a project using a drag-and-drop function.

In many cases, an algorithm requires the DM to set some parameters. This can be done with a dialogue box that is displayed after double-clicking the module. Modules contain an additional output file, named “messages”, including the information either on a successfully completed calculation or on the errors (failure). *diviz* allows exporting projects (workflows) with or without input files.

*diviz* can be used as a powerful educational program which shows partial results of each step of an algorithm or as a commercial tool for decision making in the context of real-world problems. One of the most important advantages of *diviz* is that the calculation is delegated on the external server, regardless of the user’s machine limitations. *diviz* does not require the user to have any programming skills, but to understand the functionality of the modules.

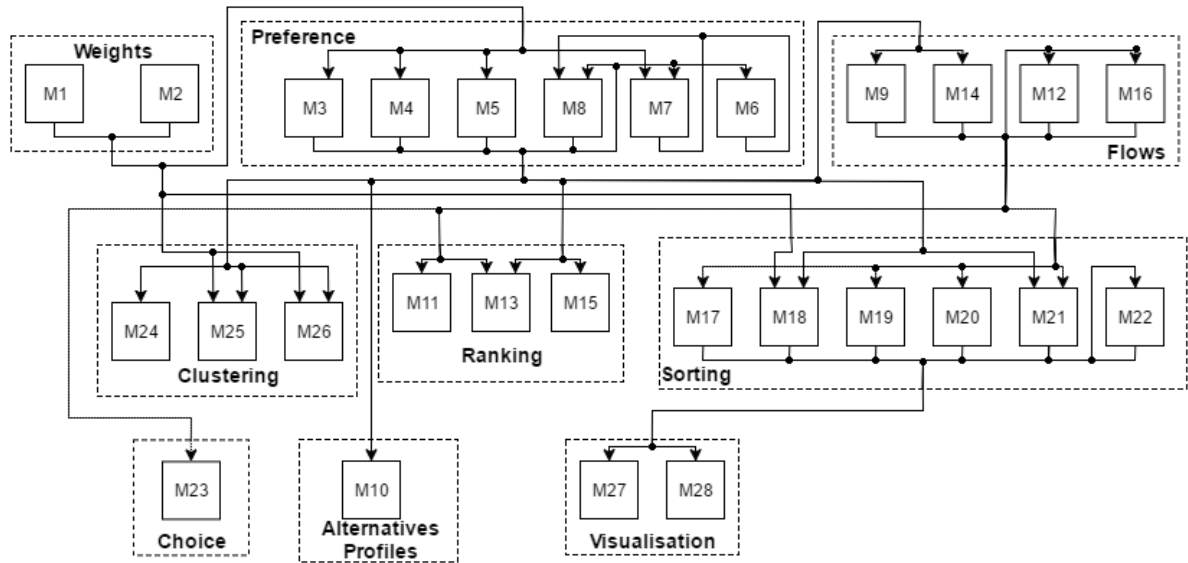


Figure 3.1: All possible connections between modules.

In Figure 3.1, we presented all our modules with all possible connections between them. To facilitate the analysis of our system of modules we divided them into groups. The modules of each group are taking part in creating some structures (weights, preferences, flows, alternatives profiles), resolving some problems (choice, clustering, ranking, sorting), or visualizing the results. The flow of data between the modules is realized in accordance with the direction of the arrows. We can distinguish three main types of information which are transmitted between our modules (and groups of modules): weights of criteria, preference indices, and outranking flows. Weights of criteria produced by modules M1 and M2 are accepted by most of modules from “Preference” group (M3, M4, M5, M7), “Clustering” group (M25, M26) and by one module from “Sorting” group (M18). Preferences computed in modules M3, M4, M5, M8 are used in creating outranking flows (M9, M14), in “Clustering” (M24, M25, M26), “Ranking” (M13, M15), “Sorting” (M18, M21), and in constructing “Alternatives Profiles”. Outranking flows derived in modules M9, M12, M14 and M16 are used as a base for “Choice” module M23, in “ranking”

modules (M11, M13) and in “Sorting” modules (M17, M19, M20, M21). Assignments created by “Sorting” modules can be visualized by modules M27 and M28. Results from all modules can be treated as independent incorporated as partial inputs for further processing.

Using our modules, the user can construct both the existing (based on the PROMETHEE family) as well as some new methods. For example, to reconstruct PROMETHEE II (s)he can create workflow which consists of modules: M3, M9, and M12. The weights of criteria can be specified directly or generated by one of “Weights” modules. If we used module M5 which takes into account interactions between criteria, module M7 which introduces the veto effect and combine their results with module M8, we can obtain preference indices which has been never calculated in that way. A variety of possibilities for combining modules and creating new combinations is the strongest point of our system.

## 4 Illustrative Case Studies

Let us consider two illustrative case studies which will be solved in the decision aiding process using methods from the PROMETHEE family. All results will be shown step by step and then thoroughly discussed.

### 4.1 Multiple criteria ranking and choice - example 1

We reconsider the problem of selecting a subset of research projects which will be financed by the government from the set of 20 research proposals [31]. The decision to be made will take into account not only performances of the alternatives on multiple criteria, but also the whole performance of portfolio. We created the workflow in *diviz*, which shows the processes carried out in analysis of this problem. It is presented in Figure 4.1. The list of alternatives with evaluations on several criteria and the related cost (budget) is presented in Table 4.1.

**Criteria.** The alternatives are evaluated in terms of six following criteria:

- $g1$ : the presentation’s writing quality which accepts scores on a 5-point ordinal scale
- $g2$ : the publication score which describes the combined quality value of all researches who submit proposal on a 100-point scale,
- $g3$ : an adequacy of the proposal to the government’s priorities on a 3-point ordinal scale,
- $g4$ : the scientific quality scores on a 5-point ordinal scale,
- $g5$ : experience of researchers scores on a 5-point ordinal scale,
- $g6$ : international collaboration expressed on a binary scale.

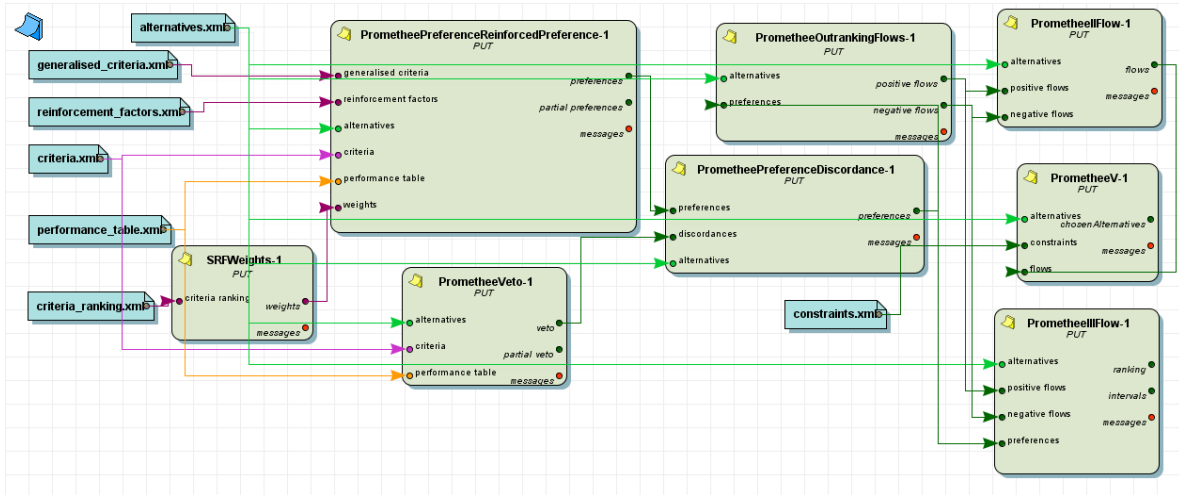


Figure 4.1: Workflow of Example 1 in diviz

Table 4.1: Performances of alternatives in Example 1.

Alternative	g1	g2	g3	g4	g5	g6	Budget
a1	2	47	2	3	1	0	27
a2	2	3	2	4	4	0	29
a3	5	63	1	5	1	0	20
a4	1	92	3	5	5	1	34
a5	4	13	2	4	2	0	32
a6	5	5	3	5	1	0	22
a7	1	27	3	2	5	1	34
a8	4	28	0	5	3	0	30
a9	3	73	0	2	1	1	28
a10	4	3	2	2	3	0	21
a11	3	75	2	4	3	1	32
a12	3	40	2	1	3	0	37
a13	2	48	1	3	3	0	26
a14	5	57	1	0	2	1	16
a15	1	16	3	4	2	1	13
a16	3	22	0	3	4	0	32
a17	4	90	2	0	0	0	35
a18	4	33	2	1	5	1	20
a19	1	95	3	1	3	0	40
a20	2	18	1	4	2	1	39

The parameters which are associated with the criteria are presented in Table 4.2. In particular, all criteria have different importances for the DM. We provided a ranking of the criteria which is used to determine the underlying weights with the SRF method

(2.1.2). This ranking and the derived weights are presented in Table 4.2. In creating this ranking we used the rules presented in Module M2 (Section 2.1.2). The most important criterion is associated with the greatest rank position. We choose criteria weight ratio equal to 5, and expected the result to be expressed as integers.

Table 4.2: Description of criteria in Example 1.

Criterion	g1	g2	g3	g4	g5	g6
Rank	1	6	3	9	7	3
Weight	6	20	11	29	23	11
Generalised Criterion	3	6	1	5	1	1
Type	gain	gain	gain	gain	gain	gain
Indifference Thr.	0	-	0	1	0	0
Preference Thr.	2	-	0	2	0	0
Sigma Thr.	-	4	-	-	-	-
Veto Thr.	-	40	-	4	-	-
Reinforced Preference Thr.	-	-	2	3	3	-
Reinforcement Factor	-	-	1.2	1.5	1.3	-

In Table 2.1, we listed six generalized criteria types which can be used to calculate preference indices. In this example, each attribute is associated with a generalized criterion. The latter determines the types of thresholds which need to be provided before computing the preference degrees. A single criterion is assigned to Gaussian Criterion; thus, it requires specification of the sigma threshold. The remaining criteria are associated with the indifference and preference thresholds (note that in some cases there are equal to zero).

The reinforced preference thresholds are assigned to three selected criteria. Their use implies that if the performance of one alternative on a specific criterion is higher than that of another alternative by more than the value specified by this threshold, then the preference index on this criterion will take the value of this criterion reinforcement factor, being greater than one.

**Preference Indices.** In Table 4.3, we provide examples of both partial and aggregated preference degrees. We tried to depict the most representative cases that can occur in calculating preference.

Let us consider the results of calculations for a pair (a1,12). The preference indices in g1, g3, g5 and g6 are equal to 0, since a1 has at most the same evaluation as a12. This situation occurs in all types of generalized criteria. On criterion g4 – which is associated with V-shape preference function – the performance difference is equal to the preference threshold. Hence, the preference index is equal to 1.

On g2, the respective index is equal to 0.784. When Gaussian preference function is used, the preference increases smoothly with the increase in performance difference. In our case, the inflection point of the preference function is expressed by sigma threshold



equal to 4. We can treat this point more or less as the middle point between indifference and preference threshold. With the performance difference being equal to 7, the attained preference index (0.784) represents this case quite well. The value of preference index on criterion g2 should never reach 1. When comparing pairs of alternatives (a1,a5) and (a4,a1), the difference in their evaluations is very huge. The value of preference degrees is so close to 1 that the computer cannot express this, so it rounds the value to 1.

When comparing a3 with a5, we can notice that on criterion g4 the preference index is equal to 0, but alternative a3 is more favorable. The difference between evaluations is equal to 1. In this case, the indifference threshold (=1) was not crossed, so the observed difference was deemed negligible in terms of DM's preferences. For the V-shape criterion, the value of preference index is increasing from 0 to 1 in the area between 0 and preference threshold. Examples of this phenomenon are visible comparing alternatives (a3,a5) and (a5,a11) on criterion g1.

Table 4.3: Exemplary Preference Indices in Example 1.

Alt. a	Alt. b	g1	g2	g3	g4	g5	g6	Aggregated Preference
a1	a12	0	0.784	0	1	0	0	0.447
a2	a14	0	0	1	1.5	1	0	0.677
a3	a5	0.5	1	0	0	0	0	0.23
a4	a1	0	1	1	1	1.3	1	0.944
a5	a11	0.5	0	0	0	0	0	0.03
a6	a16	1	0	1.2	1	0	0	0.472
a12	a3	0	0	1	0	1	0	0.34
a12	a4	1	0	0	0	0	0	0.06

**Reinforced Preference Effect.** To illustrate the reinforced preference effect, let us refer to criteria g3, g4 and g5. On the attributes, for some pairs of alternatives the preference degrees are greater than one. In each of these cases, the evaluation of one alternative is much better than the other, so it deserves to get a bonus in the preference index. The partial preference indices from all criteria are aggregated to a comprehensive value using the weights of criteria. This is conducted according to Equation 16.

Table 4.4: Exemplary Veto Indices in Example 1.

Alt. a	Alt. b	g1	g2	g3	g4	g5	g6	Aggregated Veto
a1	a12	0	0	0	0	0	0	0
a2	a14	0	1	0	0	0	0	1
a3	a5	0	0	0	0	0	0	0
a4	a1	0	0	0	0	0	0	0
a5	a11	0	1	0	0	0	0	1
a6	a16	0	0	0	0	0	0	0
a12	a3	0	0	0	1	0	0	1
a12	a4	0	1	0	1	0	0	1

**Veto Effect.** For two out of six considered criteria, the veto thresholds have been defined. In Table 4.4, we illustrate the discordance effect while referring to the same exemplary pairs of alternatives as in Table 4.3.

The veto effect can occur only criteria g2 and g4. For example, with alternative a2 being worse than alternative a14 by more than value specified by the DM as a veto threshold, the veto index is equal to 1. In other cases the veto effect does not occur. The aggregated veto can be calculated in two ways. In first method we can use the weights of criteria to calculate the aggregated veto. In this example we assumed that the veto on one criterion is enough to imply that the comprehensive veto is equal to 1.

Table 4.5: Sample Overall Preference Indices in Example 1

Alt. a	Alt. b	Aggregated Preference	Aggregated Veto	Overall Preference
a1	a12	0.447	0	0.447
a2	a14	0.677	1	0
a3	a5	0.23	0	0.23
a4	a1	0.944	0	0.944
a5	a11	0.03	1	0
a6	a16	0.472	0	0.472
a12	a3	0.34	1	0
a12	a4	0.06	1	0

Table 4.6: Outranking flows in Example 1 (values rounded to 4 decimal places)

Alternative	Positive Flows	Negative Flows
a1	0.2030	0.3755
a2	0.1663	0.2897
a3	0.4045	0.1409
a4	0.7232	0.0095
a5	0.1598	0.3395
a6	0.1594	0.2956
a7	0.3742	0.2942
a8	0.3114	0.2218
a9	0.2885	0.2125
a10	0.0816	0.4576
a11	0.5226	0.0479
a12	0.1961	0.4389
a13	0.2778	0.2965
a14	0.1876	0.3817
a15	0.2308	0.2948
a16	0.2320	0.3907
a17	0.1353	0.1862
a18	0.2789	0.3342
a19	0.3144	0.1036
a20	0.2106	0.3465

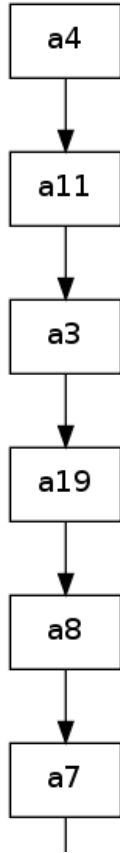


Figure 4.2: The part of visualization the results Promethee II.

Table 4.7: Flows in Promethee II in Example 1

Alternative	Flow
a1	-0.1726
a2	-0.1233
a3	0.2636
a4	0.7137
a5	-0.1798
a6	-0.1363
a7	0.08
a8	0.0896
a9	0.076
a10	-0.376
a11	0.4747
a12	-0.2428
a13	-0.0187
a14	-0.194
a15	-0.064
a16	-0.1587
a17	-0.0509
a18	-0.0553
a19	0.2107
a20	-0.136

**Overall Preference Indices.** The aggregated preference indices are combined with aggregated veto indices into overall preference indices. This is conducted by using the Formula 32. The results of putting together these values for some exemplary pairs of alternatives are presented in Table 4.5. The preference indices are just reduced to zero when the veto effect occurs; otherwise, they remain unchanged. This construction of workflow allows to combine different sources of information about preferences and discordances. In this example, we used a binary veto effect, but the discordance implemented as the M6 module can be used as well.

**Outranking flows.** All these overall preferences are used to derive positive and negative outranking flows which are used within many methods for ranking the alternatives. Outranking flows calculated in our workflow are presented in Table 4.6.

**Promethee II.** The next step consists in computing the final ranking results in two different ways. The first one is based on the flows calculated with Promethee II method which are computed on the basis of positive and negative outranking flows.

The obtained values are shown in Table 4.7. The greater the respective value, the better the alternative. The visualisation of this result is shown in Figure 4.2. This figure

has been obtained with the `plotAlternativesValuesPreorder` module. Currently, the latter accepts only input provided in the XMCDA2 format. Thus, we had to manually change the format of the output derived with our module. However, after the scheduled adjustment of `plotAlternativesValuesPreorder` to XMCDA3, it will interoperate with our modules in an automatic way.

**Promethee III.** The other one is a matrix of weak preferences which is the result of Promethee III method. It is shown in Table 4.8. The method was run with parameter  $\alpha = 0.005$ . Setting this parameter so small enabled us to observe more diverse interactions between alternatives.

Comparing those two results we can easily see that the outcomes are similar. Alternative *a4* which has the greatest value of the outranking flow in Promethee II is better than the others alternatives according to the results presented in Table 4.8: there is no any other alternative which is even weakly preferred over *a4*. The same dependencies can be easily seen also with respect to other alternatives.

The visualization of this result is shown in Figure 4.3. This figure has been obtained with the `plotAlternativesComparisons` module. Analyzing Figures 4.2 and 4.3, we can see that they are the same. Some small differences appear only in the middle of both rankings, but the general order is maintained.

Table 4.8: Ranking in Promethee III in Example 1.

	a1	a2	a3	a4	a5	a6	a7	a8	a9	a10	a11	a12	a13	a14	a15	a16	a17	a18	a19	a20
a1					S					S		S		S						
a2	S				S	S				S		S		S		S				S
a3	S	S			S	S	S	S	S	S		S	S	S	S	S	S	S	S	S
a4	S	S	S		S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S
a5										S		S		S						
a6	S				S					S		S		S		S				S
a7	S	S			S	S			S	S		S	S	S	S	S	S	S		S
a8	S	S			S	S	S		S	S		S	S	S	S	S	S	S		S
a9	S	S			S	S			S	S		S	S	S	S	S	S	S		S
a10																				
a11	S	S	S		S	S	S	S	S	S		S	S	S	S	S	S	S	S	S
a12										S										
a13	S	S			S	S				S		S		S	S	S	S	S		S
a14										S		S		S						
a15	S	S			S	S				S		S		S		S				S
a16	S				S					S		S		S						
a17	S	S			S	S				S		S		S	S	S		S		S
a18	S	S			S	S				S		S		S	S	S				S
a19	S	S			S	S	S	S	S	S		S	S	S	S	S	S	S		S
a20	S				S	S				S		S		S		S				S

**Promethee V.** Promethee V generates a subset of the most preferred alternatives when we have to deal with some constraints. We aim at optimizing a sum of alternatives' qualities represented by their flows. In this case, we have budgetary constraints in which each alternative has a factor defining the cost of its choice, and the summary cost of all chosen alternatives must be lower or equal to 150. Table 4.9 shows the results of

this method in which a value of 1 means that the respective alternative is chosen by the Promethee V algorithm. An alternative which was not chosen by the algorithm does not have to be worse than the chosen one; just it has not contributed to a construction of the most preferred subset.

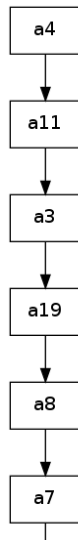


Figure 4.3: The part of visualization the results Promethee III.

Table 4.9: Promethee V in Example 1.

Alternative	Selected
a1	0
a2	0
a3	1
a4	1
a5	0
a6	0
a7	0
a8	0
a9	0
a10	0
a11	1
a12	0
a13	0
a14	0
a15	0
a16	0
a17	0
a18	0
a19	1
a20	0

## 4.2 Multiple criteria sorting and clustering - example 2

We consider a real world problem from banking sector [11]. In this problem, the DM aim is to assign 40 alternatives to 5 categories. In Figure 4.4 we present the workflow created in *diviz*, which reproduces all processes carried out in the analysis of this example. All alternatives with their performances in terms of 7 criteria are presented in Table 4.10.

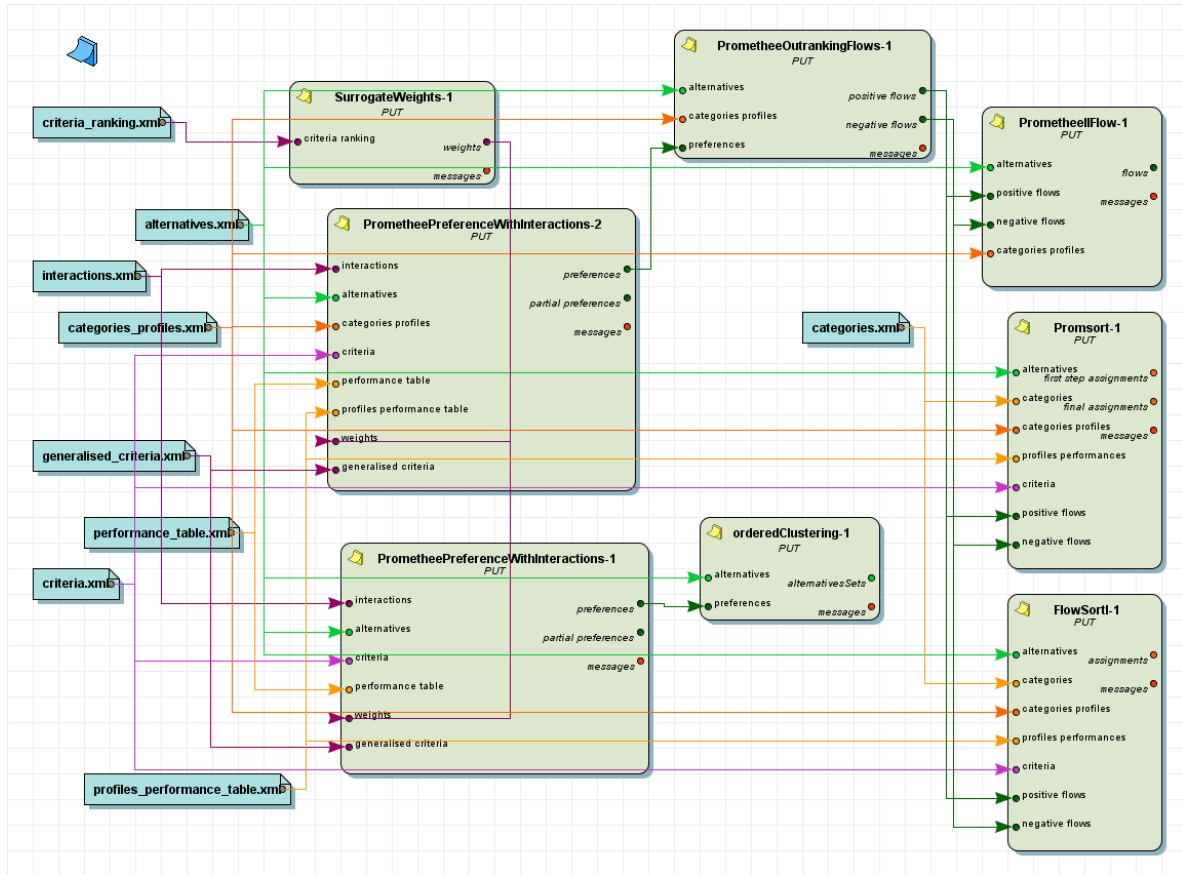


Figure 4.4: Workflow of Example 2 in *diviz*

We defined 5 categories of financial risk:  $C_1$ : Very High Risk (the worst category),  $C_2$ : High Risk,  $C_3$ : Medium Risk,  $C_4$ : Low Risk, and  $C_5$ : Very Low Risk (the best category). These categories are delimited by 4 boundary profiles. The performances of these profiles can be found in Table 4.11. Each profile can be perceived as the boundary between the two neighboring categories:

- $b_{12}$ : High Risk / Very High Risk,
- $b_{23}$ : Medium Risk / High Risk,
- $b_{34}$ : Low Risk / Medium Risk,
- $b_{45}$ : Very Low Risk / Low Risk.

Table 4.10: Performance of alternatives in Example 2

Alternative	g1	g2	g3	g4	g5	g6	g7
a1	35.8	67	19.7	0	0	5	4
a2	16.4	14.5	59.8	7.5	5.2	5	3
a3	35.8	24	64.9	2.1	4.5	5	4
a4	20.6	61.7	75.7	3.6	8	5	3
a5	11.5	17.1	57.1	4.2	3.7	5	2
a6	22.4	25.1	49.8	5	7.9	5	3
a7	23.9	34.5	48.9	2.5	8	5	3
a8	29.9	44	57.8	1.7	2.5	5	4
a9	8.7	5.4	27.4	4.5	4.5	5	2
a10	25.7	29.7	46.8	4.6	3.7	4	2
a11	21.2	24.6	64.8	3.6	8	4	2
a12	18.3	31.6	69.3	2.8	3	4	3
a13	20.7	19.3	19.7	2.2	4	4	2
a14	9.9	3.5	53.1	8.5	5.3	4	2
a15	10.4	9.3	80.9	1.4	4.1	4	2
a16	17.7	19.8	52.8	7.9	6.1	4	4
a17	14.8	15.9	27.9	5.4	1.8	4	2
a18	16	14.7	53.5	6.8	3.8	4	4
a19	11.7	10	42.1	12.2	4.3	5	2
a20	11	4.2	60.8	6.2	4.8	4	2
a21	15.5	8.5	56.2	5.5	1.8	4	2
a22	13.2	9.1	74.1	6.4	5	2	2
a23	9.1	4.1	44.8	3.3	10.4	3	4
a24	12.9	1.9	65	14	7.5	4	3
a25	5.9	-27.7	77.4	16.6	12.7	3	2
a26	16.9	12.4	60.1	5.6	5.6	3	2
a27	16.7	13.1	73.5	11.9	4.1	2	2
a28	14.6	9.7	59.5	6.7	5.6	2	2
a29	5.1	4.9	28.9	2.5	46	2	2
a30	24.4	22.3	32.8	3.3	5	3	4
a31	29.5	8.6	41.8	5.2	6.4	2	3
a32	7.3	-64.5	67.5	30.1	8.7	3	3
a33	23.7	31.9	63.6	12.1	10.2	3	2
a34	18.9	13.5	74.5	12	8.4	3	3
a35	13.9	3.3	78.7	14.7	10.1	2	2
a36	-13.3	-31.1	63	21.2	29.1	2	1
a37	6.2	-3.2	46.1	4.8	10.5	2	1
a38	4.8	-3.3	71.1	8.6	11.6	2	2
a39	0.1	-9.6	42.5	12.9	12.4	1	1
a40	13.6	9.1	76	17.1	10.3	1	1

The interpretation of the criteria used to evaluate alternatives is as follows:

- $g1$ : (Financial ratio) Earning Before Interest and Taxes/Total Assets,
- $g2$ : (Financial ratio) Net Income/Net Worth,
- $g3$ : (Financial ratio) Total Liabilities/Total Assets,
- $g4$ : (Financial ratio) Interest Expenses/Sales,
- $g5$ : (Financial ratio) General and Administrative Expenses/Sales,
- $g6$ : (Qualitative criterion) Managers Work Experience,
- $g7$ : (Qualitative criterion) Market Niche/Position.

All details describing the criteria are included in Table 4.12. It contains information about ranks, weights, generalized criteria, types of criteria (gain/cost), and thresholds. In Table 4.13, we also present the interaction types and coefficients defined for some selected pairs of criteria.

Weights of criteria in this example are calculated using Rank-Order Centroid method described in Section 2.1.1. The most important criterion is assigned the first position in ranking. The obtained weights in table are rounded for a better visual effect. The weight of criterion  $g2$  is more than eighteen times greater than weight of criterion  $g6$ . In the SRF method, this ratio can be set as a parameter, but in this example it is independent of decision maker.

In this case we have a wide spectrum of generalized criteria (solely Gaussian Criterion was not used). In the previous example, we explained the role of generalized criteria 1, 3, 5 and 6. Hence, here we will focus on the generalized criteria which are used for the first time.

In sorting problems, alternatives are compared with class profiles, because the DM needs only to assign the alternatives to specific categories. In Table 4.14, we present the list of exemplary partial and aggregated preference indices.

Criterion  $g2$  is associated with the U-shape preference function (type 2) with preference threshold equal to 4. The preference index on this criterion in comparing alternative  $a9$  with profile  $b34$  is equal to 0, because the performance of  $a9$  is worse than that of  $b34$ . This index is the same if its performance is better, but not more than indifference threshold (see, e.g., ( $a26$ ,  $b23$ )). The evaluation greater than indifference threshold is enough to achieve full preference on this criterion (compare, e.g.,  $b34$  with  $a17$ ). Level Criterion (type 4) is assigned to criterion  $g3$ . In this function preference index can take 3 values: 0 if the difference is less or equal to indifference criterion, 0.5 if it is between indifference and preference criterion, or 1 if it is greater than preference criterion. These three cases are showed in sample preferences: ( $a27$ ,  $b12$ ), ( $b12$ ,  $a32$ ), ( $b12$ ,  $a40$ ).

**Interactions between criteria.** In this example aggregated preferences are not just the weighted sum of partial preferences. In Table 4.13, we list the interactions between criteria which have some impact on the result. This effect is explained in Section 2.2.2. For ( $b12$ ,  $a40$ ), criteria  $g6$  and  $g7$  support the idea that  $b12$  is preferred over  $a40$ , so



Table 4.11: Performance of profiles in Example 2

Profile	g1	g2	g3	g4	g5	g6	g7
b12	4	0	65	28	12	2	3
b23	10	10	45	23	8	3	4
b34	15	20	40	18	4	4	5
b45	25	30	35	10	0	5	6

Table 4.12: Description of criteria in Example 2.

Criterion	g1	g2	g3	g4	g5	g6	g7
Rank	5	1	2	4	3	7	6
Weight	0.073	0.370	0.228	0.109	0.156	0.020	0.044
Generalised Criterion	5	2	4	3	5	1	1
Type	gain	gain	cost	cost	cost	gain	gain
Indifference Thr.	1	4	1	0	1	0	0
Preference Thr.	2	4	3	3	3	0	0

Table 4.13: Interactions between criteria in Example 2.

Criterion 1	Criterion 2	Interaction type	Coefficient
g6	g7	strengthening	0.03
g1	g3	strengthening	0.02
g4	g5	weakening	-0.05
g2	g3	antagonistic	0.08

Table 4.14: Exemplary Preference Indices in Example 2.

Alt. A	Alt. B	g1	g2	g3	g4	g5	g6	g7	Aggregated Preference
a3	b12	1	1	0	1	1	1	1	0.7678
a9	b34	0	0	1	1	0	1	0	0.3565
a17	b34	0	0	1	1	0.6	0	0	0.4121
a22	b23	1	0	0	1	1	0	0	0.3025
a23	b23	0	0	0	1	0	0	0	0.1085
a26	b23	1	0	0	1	0.7	0	0	0.2648
a27	b12	1	1	0	1	1	0	0	0.6642
b12	a32	0	1	0.5	0.7	0	0	0	0.5601
b12	a40	0	0	1	0	0	1	1	0.3128
b34	a5	1	0	1	0	0	0	1	0.3574
b34	a17	0	1	0	0	0	0	1	0.3637

they should have greater influence for the result. If we calculated this partial result without interaction effect the aggregated preference would be equal approximately to 0.2922, but if we accounted for the interaction, this value would rise up to 0.3128. This growth was implied by the strengthening coefficient equal to 0.03.

For (b34, a5), one illustrated the strengthening effect on criteria g1 and g3. The obtained aggregated preference increase from 0.3446 to 0.3574. The third interaction occurs in comparing alternative a22 and profile b23. The mutual weakening effect decreases the result from 0.3374 to 0.3025. The last interaction which represents the antagonistic effect can be shown observed for a pair (b43, a17). Criterion g2 supports the idea that b43 is preferred over a17, but criterion g3 supports opposite preference direction. This allows to decrease the influence of criterion g2 on the aggregated preference index. As a result, the value of index is decreased from 0.4146 to 0.3637. In general, the number of interactions is not limited and it can occur multiple times in each example. The strengthening and weakening effect exist both at once in first example (a3, b12).

**Outranking flows.** After the preference degrees are calculated, the next step is to computed the outranking flows. In sorting problems, the flows are derived from comparisons against the profiles. These have been presented in Tables 4.15 (for profiles) and 4.16 (for alternatives).

Table 4.15: Outranking flows of profiles in Example 2.

Profile	Positive Flow	Negative Flow
b12	0	1
b23	0.3333	0.6667
b34	0.6667	0.3333
b45	1	0

**Assignments.** In Table 4.17, we present the final assignments of all alternatives to categories derived with the PromSort and FlowSort I methods. Both methods use the Promethee I positive and negative flows. As one can see PromSort returns precise assignments, while FlowSort I delivers imprecise ones.

During calculations, we used PromSort module M17 described in Section 2.6.1 and FlowSort I module M18 described in Section 2.6.2. PromSort required setting two parameters - “cut point” and “assign to a better class”. With setting “cut point” to 0 we are basing a second step assignments on a distance function. With parameter “assign to a better class” set to *true*, when a distance to both potential categories is equal, we will assign alternative to a better category. FlowSort I required us to set only one parameter - “profiles type”. We set it to “boundary”, as we used the boundary profiles in our calculations.

After analyzing the results one can see that for all alternatives the assignments obtained with different methods are consistent. For each of our example alternatives the final PromSort assignment was at least as good as FlowSort I pessimistic assignment and at most as good as FlowSort I optimistic assignment. As an example, alternative

Table 4.16: Outranking flows of alternatives in Example 2.

Alternative	Positive Flows	Negative Flows
a1	0.9231	0.0221
a2	0.4686	0.4508
a3	0.4596	0.3244
a4	0.5478	0.3594
a5	0.4594	0.3868
a6	0.5332	0.3933
a7	0.6145	0.2840
a8	0.6930	0.2220
a9	0.5335	0.3718
a10	0.5692	0.2346
a11	0.4675	0.4101
a12	0.4991	0.3287
a13	0.6663	0.1947
a14	0.2410	0.5816
a15	0.2880	0.5402
a16	0.4647	0.3705
a17	0.6642	0.2500
a18	0.4762	0.4416
a19	0.3775	0.4108
a20	0.3547	0.5759
a21	0.3911	0.4414
a22	0.2959	0.5536
a23	0.3050	0.5683
a24	0.1586	0.6289
a25	0.0884	0.8531
a26	0.3754	0.4826
a27	0.2815	0.5545
a28	0.3538	0.4929
a29	0.4306	0.5348
a30	0.6440	0.1697
a31	0.4386	0.3960
a32	0.0623	0.8468
a33	0.4428	0.3551
a34	0.2626	0.5719
a35	0.1310	0.7147
a36	0.0718	0.8813
a37	0.1944	0.6466
a38	0.0940	0.7404
a39	0.1667	0.7313
a40	0.1944	0.6444

Table 4.17: PromSort and FlowSort I alternatives assignments.

Alternatives	Assignments		
	PromSort	FlowSort I	
	Precise	Pessimistic	Optimistic
a1	C4	C4	C4
a2	C3	C3	C3
a3	C3	C3	C4
a4	C3	C3	C3
a5	C3	C3	C3
a6	C3	C3	C3
a7	C3	C3	C4
a8	C4	C4	C4
a9	C3	C3	C3
a10	C3	C3	C4
a11	C3	C3	C3
a12	C3	C3	C4
a13	C3	C3	C4
a14	C2	C2	C3
a15	C2	C2	C3
a16	C3	C3	C3
a17	C3	C3	C4
a18	C3	C3	C3
a19	C3	C3	C3
a20	C3	C3	C3
a21	C3	C3	C3
a22	C2	C2	C3
a23	C2	C2	C3
a24	C2	C2	C3
a25	C2	C2	C2
a26	C3	C3	C3
a27	C2	C2	C3
a28	C3	C3	C3
a29	C3	C3	C3
a30	C3	C3	C4
a31	C3	C3	C3
a32	C2	C2	C2
a33	C3	C3	C3
a34	C2	C2	C3
a35	C2	C2	C2
a36	C2	C2	C2
a37	C2	C2	C3
a38	C2	C2	C2
a39	C2	C2	C2
a40	C2	C2	C3

$a_3$  was assigned with PromSort to category  $C3$ , in FlowSort I pessimistic assignment is the same ( $C3$ ), optimistic assignments is better ( $C4$ ). In some cases, the assignments were identical (for example  $a_1$ ).

Let's now trace the way of assigning alternative  $a_3$  to category  $C3$  in PromSort method. The alternative has positive flow 0.4596 and negative flow 0.3244. Its flows are being compared with profiles flows to get the relations between alternative and each of the profiles. As one can see alternative  $a_3$  is better than profiles  $b_{12}$  and  $b_{23}$  (it has higher positive flow then these profiles and lower negative flow), so alternative  $a_3$  is preferred to them. Profiles  $b_{34}$  and  $b_{45}$  have lower positive flow and higher negative flow, so they are preferred to  $a_3$ . In that case the assignment from the first step of PromSort can be deemed as final for that alternative; it is assigned to category  $C3$ , which is between profiles  $b_{23}$  and  $b_{34}$ .

FlowSort I derives its results from the separated comparisons of positive and negative flows of alternative and profiles. In our example, alternative  $a_3$  is being comprised with boundary profiles. When comparing positive flows, alternative  $a_3$  with its flow 0.4596 is better then profile  $b_{23}$  (0.3333) and worse then profile  $b_{34}$  (0.6667). Because of that, the alternative  $a_3$  is assigned to category  $C3$  using positive flows. Negative flow for alternative  $a_3$  is 0.3244. It is lower (better) then negative flow of profile  $b_{34}$  and higher (worse) then the negative flow of profile  $b_{45}$ . That means that the negative assignment of alternative  $a_3$  is category  $C4$ . As positive assignment is worse then negative one it will be the pessimistic one, negative assignment will be the optimistic one. Finally, alternative  $a_3$  is assigned to categories [ $C3$ ;  $C4$ ].

Table 4.18: Ordered clustering in Example 2.

Cluster index	Alternatives
0	a1
1	a4, a6, a7, a8, a10, a13, a17, a30
2	a2, a3, a5, a9, a11, a12, a16, a18, a19, a21, a26, a28, a29, a31, a33
3	a14, a15, a20, a22, a23, a24, a27, a34, a37, a39
4	a25, a32, a35, a36, a38, a40

**Ordered Clustering.** We also run the Ordered Clustering (OC) module which assigns the alternatives into five clusters. This method uses the aggregated preference indices computed as a result of comparison alternatives with each other. In contrast to the previously used sorting method, there is no need to define categories and their respective profiles. Instead, it uses different approach which is described in Section 2.8.1. Table 4.18 shows the constructed ordered clusters. Note that the lower the index of a cluster, the better alternatives are contained in it. The OC algorithm finds the first highest preference value, the maximal one is equal to 1 and the minimal is equal to 0. The first cluster (cluster index = 0) has solely one alternative  $a_1$  since it has high preferences (the smallest one is equal to 0.74...) over other alternatives. Incorrect as it may seem, in the example workflow,  $a_1$  is always better than the remaining ones (i.e., there is not any at least as good alternative as  $a_1$ ). As the users of the OC method, we may control

the quantity of groups but not the minimal cardinality of clusters. We can say that alternative  $a_4$  is better than any other alternative aggregated in a group with lower id (there is a transitive relation between clusters) and alternative  $a_4$  is similar or indifferent to any other one assigned to the same cluster.

## 5 Conclusions

In this thesis, we described a large number of basic PROMETHEE methods. We postulated greater flexibility in constructing new approaches in this family by putting together some elementary algorithmic components.

At the stage of defining criteria weights, we considered several methods which can help to reflect the DM's preferences into specific weights using only simple information like ranking of criteria. At the stage of constructing a preference structure, we accounted for different approaches of computing preference indices. Apart from considering those commonly used in the PROMETHEE method (i.e., involving six types of preference functions and three types of thresholds (indifference, preference, sigma)), we expanded PROMETHEE by introducing reinforced preference effect, interactions between criteria, veto effect and accounting for the discordance. We considered the process of creating a positive and negative outranking flows. At the stage of exploitation of preference indices and outranking flows, we considered several algorithms which can be used to ranking alternatives (PROMETHEE I, II, and III), selecting a subset of the most preferred options (PROMETHEE V), resolving sorting problems (PromSort, Promethee Tri, FlowSort), dividing alternatives into a number of groups (Promethee II Ordered Clustering, Promethee Cluster) and for group decision making (FlowSort GDSS, Promethee Group). We investigated also the creation of alternatives profiles from partial preference indices.

We implemented the postulate of flexibility in creating the PROMETHEE methods in practice. We developed 28 universal, highly parameterized and flexible modules realizing some methods, procedures or visualizing data according to them in the spirit of PROMETHEE. The modules are designed to compare alternatives with each other, alternatives with central (characteristic) class profiles and with boundary class profiles. We introduced them into *diviz* platform, which provides full environment for exploiting them. It allows to recreate existing PROMETHEE methods and to create new methods undiscovered by scientists in earlier considerations. Our modules allow to create several hundreds valid combinations which can accurately reflect a particular decision context and all that without any mathematical and programming skills.

We reconsidered two real-world problems to present construction of the new PROMETHEE methods and to show how it can help in decision aiding process. In this examples we designed a few not considered earlier variants of PROMETHEE. For these examples we created workflows which can be easy imported into *diviz*. They can be used as a base for researchers to explore their own new PROMETHEE methods or for easy adapting to other decision context.

In this thesis we considered a very wide spectrum of PROMETHEE-based approaches to resolving multiple criteria decision problems, but we have to admit there is still some scope for improvements. However, all our modules are open source and are available for

other researchers. They can extend functionality of these programs to reach the new requirements. Furthermore, there is no any contraindications to create new modules which will cooperate with our modules on *diviz*. We can propose sample improvements, which would be useful in our PROMETHEE framework:

- Adding a module based on Promethee GAIA Plane method, which is a very powerful way of visualizing dependencies between criteria and alternatives [4];
- Adapting all implemented modules to account for the hierarchical structure of criteria;
- Extending the modules to provide results of Monte Carlo simulation involving different sets of weights consistent with some linear constraints instead of just one weight vector which is either provided by the DM or selected with some arbitrary rule, and to imprecise performances of alternatives;
- Adapting robust ordinal regression methods to the context of a wide variety of PROMETHEE method.

## 6 Acknowledgments

The authors thank Sébastien Bigaret from IMT Atlantique for helping us to make our modules available in *diviz* platform.

## References

- [1] Araz, C. and Ozkarahan, I., 2007. Supplier evaluation and management system for strategic sourcing based on a new multicriteria sorting procedure, *Int. J. Production Economics* 106, 585–606.
- [2] Behzadian, M., Kazemzadeh, R.K., Albadvi, A. and Aghdasi, M., 2010. PROMETHEE: A comprehensive literature review on methodologies and applications, *European Journal of Operational Research*, 100(1): 198–215.
- [3] Bouyssou, D. and Perny, P., 1992. Ranking methods based on valued preference relations: A characterization of the net flow method. *European Journal of Operational Research* 60, 61–67.
- [4] Brans, J.-P. and Mareschal, B., 2005. PROMETHEE methods. In Figueira, J., Greco, S., Ehrgott, M., editors, *Multiple Criteria Decision Analysis. State of the Art Surveys*, 163–195. Springer, New York, USA.
- [5] Brans, J.-P., Vincke, Ph. and Mareschal, B., 1986. How to select and how to rank projects: The PROMETHEE method. *European Journal of Operational Research* 24 (2), 228–238.
- [6] Cavalcante, C.A.V. and de Almeida, A.T., 2007. A multi-criteria decision-aiding model using PROMETHEE III for preventive maintenance planning under uncertain conditions. *Journal of Quality in Maintenance Engineering*, Vol. 13 Iss 4 pp. 385 - 397.
- [7] Corrente, S., Greco, S. and Słowiński, R., 2016. Multiple Criteria Hierarchy Process for ELECTRE TRI. *European Journal of Operational Research*, 252(1), 198-199.
- [8] Damart, S., Dias, L.C. and Mousseau, V., 2007. Supporting groups in sorting decisions: Methodology and use of a multi-criteria aggregation/disaggregation DSS. *Decision Support Systems* 43 (2007) 1464–1475
- [9] De Smet, Y., 2013. P2CLUST: An Extension of PROMETHEE II for Multicriteria Ordered Clustering. 2013 IEEE International Conference on Industrial Engineering and Engineering Management, 848-851
- [10] De Smet, Y., Nemery, P. and Selvaraj, R., 2012. An exact algorithm for the multicriteria ordered clustering problem. *Omega* 40, 861–869
- [11] Dias, L., et al., 2002. An aggregation/disaggregation approach to obtain robust conclusions with ELECTRE TRI. *European Journal of Operational Research* 138.2, 332-348.
- [12] D’Avignon, G. and Mareschal, B., 1989. An application of the PROMETHEE and GAIA methods. *Mathematical and Computer Modelling* 12 (10–11), 1393–1400.



- [13] Figueira, J., De Smet, Y. and Brans, J.P., 2004. MCDA methods for sorting and clustering problems: Promethee TRI and Promethee CLUSTER. Université Libre de Bruxelles, Service de Mathématiques de la Gestion, Working Paper 2004/02
- [14] Figueira, J., Greco, S. and Roy, B., 2009. ELECTRE methods with interaction between criteria: An extension of the concordance index. *European Journal of Operational Research* 199, 478-495.
- [15] Figueira, J. and Roy, B., 2002. Determining the weights of criteria in the ELECTRE type methods with a revised Simos' procedure, *European Journal of Operational Research* 139, 317-326.
- [16] Halouani, N., Chabchoub, H. and Martel, J.-M., 2009. PROMETHEE-MD-2T method for project selection, *European Journal of Operational Research* 195 841–849
- [17] Hayez, Q., De Smet, Y. and Bonney, J., 2012. D-Sight: A New Decision Making Software to Address Multi-Criteria Problems, *International Journal of Decision Support System Technology (IJDSST)* 4(4)
- [18] Hu, Y.C. and Chen, C.J., 2011. A PROMETHEE-based classification method using concordance and discordance relations and its application to bankruptcy prediction. *Information Sciences* 181, 4959-4962.
- [19] Huth, A., Drechsler, M. and Kohler, P., 2005. Using multicriteria decision analysis and a forest growth model to assess impacts of tree harvesting in Dipterocarp lowland rain forests. *Forest Ecology and Management* 207, 215–232.
- [20] Kadziński, M. and Michalski, M., 2016. Scoring procedures for multiple criteria decision aiding with robust and stochastic ordinal regression. *Computers & Operations Research* 71, 54–70.
- [21] Lolli, F., Ishizaka, A., Gamberini, R., Rimini, B. and Messori, M., Appl. 2015. FlowSort-GDSS-A novel group multi-criteria decision support system for sorting problems with application to FMEA. *Expert Syst.*, 42, 6342–6349.
- [22] Macharis, C., Brans, J.-P. and Mareschal, B., 1998. The GDSS PROMETHEE Procedure. *Journal of Decision Systems*, Vol. 7-SI/1998, 283-307.
- [23] Mareschal, B. and De Smet, Y. 2009. Visual PROMETHEE: Developments of the PROMETHEE & GAIA multicriteria decision aid methods. In 2009 IEEE International Conference on Industrial Engineering and Engineering Management, 1646-1649
- [24] Mavrotas, G., Diakoulaki, D. and Caloghirou, Y., 2006. Project prioritization under policy restrictions. A combination of MCDA with 0–1 programming. *European Journal of Operational Research* 171, 296–308.
- [25] Meyer, P. and Bigaret, S., 2012. diviz: a software for modeling, processing and sharing algorithmic workflows in MCDA. *Intelligent Decision Technologies* 6, 283–296.

- [26] Nemery, P. and Lamboray, C., 2007. FlowSort: a flow-based sorting method with limiting or central profiles, *Top* 16(1), 90-113.
- [27] Roberts, R. and Goodwin, P., 2002. Weight approximations in multi-attribute decision models. *Journal of Multi-Criteria Decision Analysis*, 11 (6), pp. 291-303.
- [28] Roszkowska, E., 2013. Rank Ordering Criteria Weighting Methods – a Comparative Overview, *Wydawnictwo Uniwersytetu w Białymstoku, Optimum. Studia ekonomiczne*, Nr 5 (65), 21-22.
- [29] Roy, B. and Skalka, J., 1984. ELECTRE IS : Aspects méthodologiques et guide d'utilisation. Tech. rep., Document du LAMSADE N° 30, Université Paris-Dauphine, Paris.
- [30] Roy, B. and Słowiński, R., 2008. Handling effects of reinforced preference and counter-veto in credibility of outranking. *European Journal of Operational Research* 188, 185–190.
- [31] Zheng, J., Cailloux, O. and Mousseau, V., 2011. Constrained Multicriteria Sorting Method Applied to Portfolio Selection. *Algorithmic Decision Theory*, New Brunswick, New Jersey, United States. Springer, 6992, pp.331-343, Lecture Notes in Computer Science.