ELECTRE and PROMETHEE MCDA methods as reusable software components

Olivier CAILLOUX
École Centrale Paris and Université Libre de Bruxelles
Grande Voie des Vignes, 92295 Châtenay-Malabry Cedex, France, olivier.cailloux@ecp.fr

ABSTRACT. Open source software components implementing features of the ELECTRE and PROMETHEE family of MCDA procedures have been developed at Université Libre de Bruxelles and École Centrale Paris, as part of the Decision Deck project. As these procedures share several computation routines, they have been implemented as a set of independent components which can then be assembled in various ways. Care has been taken to develop complete structures representing the underlying MCDA concepts, thereby providing easy to reuse artifacts. The goal was to make them easy to use for the end-user as well as for a developer usage, i.e. to allow building of more complex functionalities on top of these objects. The components deal with XMCDA conforming files (i.e., files conforming to the XMCDA grammar, a standard published by the Decision Deck consortium) in input and output and can also be helpful for a developer wanting to provide his own software with the ability to read or write such files.
This article describes the architecture of the components and details some of their usage possibilities.

KEYWORDS. XMCDA, ELECTRE, PROMETHEE, diviz, Decision Deck

1 INTRODUCTION

The Decision Deck project aims at collaboratively developing open source software tools implementing Multiple Criteria Decision Aid (MCDA) procedures (Decision Deck Consortium, 2009b). It is driven by the Decision Deck consortium.

As a contribution to this project, Java libraries have been developed collaboratively at Université Libre de Bruxelles and École Centrale Paris. They provide a set of components implementing each a part of a MCDA procedure inside the ELECTRE (Roy, 1991; Figueira et al., 2005) and PROMETHEE (Brans et al., 1984; Brans and Vincke, 1985; Brans and Mareschal, 2005) family. More specifically, they provide implementations of the complete PROMETHEE procedures, and are able to compute the outranking relations proposed by the ELECTRE Iv, II, III procedures (as well as the concordance and discordance indexes), by appropriately assembling the components. The exploitation part of the ELECTRE procedures have not been implemented yet, although it would be easy to add them in the proposed framework. This is done using two libraries.

J-MCDA provides a set of Java objects representing the concepts required for dealing with the considered procedures, e.g. alternatives, criteria, matrix of floats, set of criteria weights, ...

The second library, named J-XMCDA, depends on J-MCDA and provides a library of serialization and deserialization. Provided with an XMCDA conforming file (i.e., a file which conforms to a specific grammar as described below), it is able to transform it into an equivalent set of objects as provided in J-MCDA library; and conversely, to write an XMCDA file when provided with J-MCDA objects.

The main goal was not only to implement a tool able to compute the results of the considered methods, but to provide good basis for further extensibility and reuse by other developers. The developed classes satisfy object oriented "best practices" such as hidden implementation details, contract based methods, reuse of classical and well designed third parties libraries, and functional tests are included.

XMCDA (Decision Deck Consortium, 2009c), as referred to in this document, is an XML schema (W3C Consortium, 2004) proposed by the Decision Deck consortium and specifies a language appropriate to describe data typical of MCDA problems. An XML file (W3C Consortium, 2008) is a computer file conforming to a specific grammar published by the W3C consortium (a well-known standardization body in

the computer science domain). An XML schema is a grammar, more restrictive than the XML grammar, typically used to define a common language for concepts used in a specific domain of knowledge. XMCDA is such an XML schema, defined for the MCDA practitioners and users. It defines a language to use to describe data such as an alternative, a criterion, an evaluation of a given alternative according to a given criterion, etc. It is not restricted to any particular MCDA procedures family.

Note that, formally speaking, the XML standard and other related standards described here above do not only relate to computer "files", but to anything which can be represented as a stream of bits (e.g. a bit stream in a network). This paper however describes, for easiness of understanding, only the "file" case.

This article uses the phrase "XMCDA conforming file", or simply "XMCDA file", to refer to a file which validates against the XMCDA schema.

This paper is organized in two parts. The first one (Section 2) describes how the MCDA procedures have been split into components and show that when put together, they indeed provide solutions conforming to the procedures specifications. The second part (Section 3) describes how these software components may be useful for an end-user and for a developer.

2 ARCHITECTURE OF THE PROCEDURES AND COMPONENTS

This section presents the notation used, then describes the procedures with a unified notation, as detailed by Belton and Stewart (Belton and Stewart, 2002), and finally describes how the software components implement the required functionalities.

ELECTRE and PROMETHEE are two families of procedures adopting the outranking approach (Vincke, 1999). They aim to aid the decision maker in a multicriteria decision problem context by providing an outranking relation, or a preference relation in terms of PROMETHEE, which represents the degree to which each alternative outranks (or is preferred to) each other one. This outranking relation is computed using, in the case of ELECTRE, concordance values, describing for each ordered pair of alternative the degree to which the first one can be considered better than the second one, and possibly discordance values using so-called veto thresholds. The discordance is also computed on ordered pairs of alternatives and takes into account the fact that the first one may be so much worse than the second one on some criteria (possibly a minority of them) that it makes the first one outranking the second one impossible. This is described more precisely below.

The outranking, or preference, relations can then be exploited to obtain results in terms of the decision problem, e.g., a ranking of the considered alternatives. The software components currently implement the complete PROMETHEE procedures, and the ELECTRE computations required to obtain the outranking matrixes, but not the ELECTRE exploitation part, which will therefore not be detailed here.

2.1 Notations and assumptions

The methods are described assuming the following context: alternatives from an alternative set A are evaluated on criteria from a given criteria set. F denotes the set of indices representing the considered criteria. The evaluations can be described by a real value. The evaluation of an alternative a on the criterion i is denoted by $z_i(a)$. Where appropriate, the criterion i may be given preference, indifference, and veto thresholds written p_i, q_i, v_i, with $v_i > p_i \geq q_i \geq 0, \forall i \in F$. Each criterion is associated with a weight $w_i \geq 0$ which represents a voting power. The weights are supposedly normalized, i.e., $\sum_{i \in F} w_i = 1$. $|A|$ denotes the number of alternatives. C^* and C^- are two majority levels, with $1 \geq C^* \geq C^- \geq 0$. All criteria are supposedly to be maximized, i.e. a lower value cannot imply a better performance. If a criterion is to be minimized instead (e.g. a cost), it suffices to invert $z_i(a)$ and $z_i(b)$ in the following equations.

The procedures are described only in the case of constant thresholds (only this case is currently imple-

mented).

2.2 Procedures description

2.2.1 ELECTRE Iv outranking relation

ELECTRE Iv builds a binary outranking relation S using a concordance function C and a discordance function D.

$$C(a,b) = \sum_{\{i|z_i(a)\geq z_i(b)\}} \omega_i. \tag{1}$$

$$D(a,b) = \begin{cases} 1 & \Leftrightarrow \exists i | z_i(b) - z_i(a) \geq v_i, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

$$S(a,b) = \begin{cases} 1 & \Leftrightarrow C(a,b) > C^* \text{ and } D(a,b) = 0, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Note that ELECTRE I originally contained another proposition for computing the discordance index. This version, known as the ELECTRE Iv procedure (Figueira et al., 2005), is the one currently implemented.

2.2.2 ELECTRE II outranking relations

ELECTRE II uses the same concordance and discordance indexes than ELECTRE Iv , and two outranking relations are then built.

$$S^-(a,b) = \begin{cases} 1 & \Leftrightarrow C(a,b) > C^- \text{ and } D(a,b) = 0, \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

$$S^*(a,b) = \begin{cases} 1 & \Leftrightarrow C(a,b) > C^* \text{ and } D(a,b) = 0, \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

2.2.3 ELECTRE III outranking relation

ELECTRE III uses fuzzy concordance, discordance and outranking relations.

$$C_i(a,b) = \begin{cases} & 1 \Leftrightarrow z_i(b) - z_i(a) = q_i \text{ and } q_i = p_i, \\ & 1 \Leftrightarrow z_i(b) - z_i(a) < q_i, \\ 1 - \dfrac{(z_i(b) - z_i(a)) - q_i}{p_i - q_i} & \Leftrightarrow q_i \leq z_i(b) - z_i(a) \leq p_i \text{ and } q_i < p_i, \\ & 0 \Leftrightarrow z_i(b) - z_i(a) > p_i. \end{cases} \tag{6}$$

While it is possible to describe the C_i function in a more compact way, this notation has been chosen because it allows for an easier comparison with the PROMETHEE preference function.

$$C(a,b) = \sum_{i \in F} \omega_i C_i(a,b). \tag{7}$$

$$D_i(a,b) = \begin{cases} 1 \Leftrightarrow & z_i(b) - z_i(a) \geq v_i, \\ \dfrac{(z_i(b) - z_i(a)) - p_i}{v_i - p_i} \Leftrightarrow & p_i \leq z_i(b) - z_i(a) < v_i, \\ 0 \Leftrightarrow & z_i(b) - z_i(a) < p_i. \end{cases} \tag{8}$$

If v_i is not specified, D_i is always nul.

$$S(a,b) = \begin{cases} & C(a,b) \Leftrightarrow \forall i \in F : D_i(a,b) \leq C(a,b), \\ C(a,b) \displaystyle\prod_{\{i|D_i(a,b)>C(a,b)\}} & \dfrac{1 - D_i(a,b)}{1 - C(a,b)} \text{ otherwise.} \end{cases} \tag{9}$$

2.2.4 PROMETHEE preference function

PROMETHEE (Brans et al., 1984; Brans and Vincke, 1985; Brans et al., 1986) defines a global preference function P by aggregating partial preference functions P_i (defined for each criterion i). Among the six partial preference functions proposed by the PROMETHEE method, the following four have been implemented.

$$P_i^1(a,b) = \begin{cases} 1 \Leftrightarrow z_i(a) - z_i(b) > 0, \\ 0 \Leftrightarrow z_i(a) - z_i(b) \leq 0. \end{cases} \tag{10}$$

$$P_i^2(a,b) = \begin{cases} 1 \Leftrightarrow z_i(a) - z_i(b) > q_i, \\ 0 \Leftrightarrow z_i(a) - z_i(b) \leq q_i. \end{cases} \tag{11}$$

$$P_i^3(a,b) = \begin{cases} 1 \Leftrightarrow z_i(a) - z_i(b) > p_i, \\ \dfrac{z_i(a) - z_i(b)}{p_i} \Leftrightarrow 0 \leq z_i(a) - z_i(b) \leq p_i \text{ and } 0 < p_i, \\ 0 \Leftrightarrow z_i(a) - z_i(b) < 0, \\ 0 \Leftrightarrow z_i(a) - z_i(b) = 0 \text{ and } p_i = 0. \end{cases} \tag{12}$$

$$P_i^5(a,b) = \begin{cases} 1 \Leftrightarrow z_i(a) - z_i(b) > p_i, \\ \dfrac{(z_i(a) - z_i(b)) - q_i}{p_i - q_i} \Leftrightarrow q_i \leq z_i(a) - z_i(b) \leq p_i \text{ and } q_i < p_i, \\ 0 \Leftrightarrow z_i(a) - z_i(b) < q_i, \\ 0 \Leftrightarrow z_i(a) - z_i(b) = q_i \text{ and } p_i = q_i. \end{cases} \tag{13}$$

$$P(a,b) = \sum_{i \in F} \omega_i P_i(a,b), \tag{14}$$

where $P_i = P_i^1, P_i^2, P_i^3$ or P_i^5 depending on the chosen preference function.

2.2.5 PROMETHEE exploitation

PROMETHEE proposes to exploit the preference function P defined above by associating each alternative to a positive (Q^+) and negative (Q^-) flow, thereby defining two rankings over the alternatives.

$$Q^+(a) = \frac{1}{|A| - 1} \sum_{b \in A \setminus \{a\}} P(a,b). \tag{15}$$

$$Q^-(a) = \frac{1}{|A|-1} \sum_{b \in A \setminus \{a\}} P(b,a). \tag{16}$$

PROMETHEE II introduces the net flow concept. The other computations are unchanged.

$$Q(a) = Q^+(a) - Q^-(a). \tag{17}$$

2.3 Software components

The following basic components have been developed, able to produce with different assemblies and parameters the required MCDA procedures.

2.3.1 Concordance

The concordance component is able to compute a concordance relation given a set of alternatives, a set of criteria, a performance table, and possibly criteria preference and indifference thresholds. The concordance relation can be the one considered in ELECTRE Iv, II, and III, according to the thresholds used. It is also able to compute a PROMETHEE-style preference relation. The name "Concordance" has been chosen arbitrarily, and both relation styles have been included in this component capability because of their proximity.

The component is able to compute a pairwize concordance or preference (corresponding to the above $C_i(a,b)$ or $P_i(a,b)$), given two thresholds p_i, q_i ($p_i \geq q_i$) and two evaluations $z_i(a), z_i(b)$, and a boolean information indicating whether an ELECTRE style concordance value or a PROMETHEE style preference value is requested. Each criterion is also associated with a boolean information specifying if it is to be maximized or minimized. The component computes the performance difference $z = z_i(a) - z_i(b)$. Then, the special case $z = p_i$ and $z = q_i$ is treated, and a value of 0 is returned if a preference value is asked, 1 if a concordance value must be computed. In the other cases, it considers two new thresholds r_1, r_2 and assigns them as follows: for an ELECTRE computation, $r_1 = -p_i$ and $r_2 = -q_i$; for PROMETHEE, $r_1 = q_i$ and $r_2 = p_i$. Finally, it returns the following value:

$$\begin{cases} 1 \Leftrightarrow z \geq r_2, \\ \dfrac{z - r_1}{r_2 - r_1} \Leftrightarrow r_1 \leq z < r_2, \\ 0 \Leftrightarrow z < r_1. \end{cases}$$

The component also can compute a matrix M of concordance or preference (corresponding to the above $C(a,b)$ or $P(a,b)$). It does so by computing, for each $a, b \in A$:

$$M(a,b) = \sum_{i \in F} w_i * M_i(a,b),$$

where $M_i(a,b)$ is the value returned by the pairwize concordance or preference function described here above (depending on whether a concordance or a preference matrix is being computed).

That matrix represents the ELECTRE III concordance matrix, or PROMETHEE preference functions when the fifth preference function is used. Obtaining the concordance values for ELECTRE Iv and II and for the simpler PROMETHEE preference functions is done by appropriately setting the p_i, q_i threshold values.

Note that $P(a,b) = 1 - C(b,a)$. This fact is used for testing the correctness of the computations in the unit tests.

2.3.2 Discordance

The discordance component can compute a discordance value for a given criterion on a given pair of alternatives, given the applicable preference and veto thresholds p_i, v_i. It computes the discordance as described in the ELECTRE III method. The ELECTRE Iv and II discordance values can be obtained by setting $p_i = v_i$.

More precisely, given a set of alternatives, a set of criteria with their corresponding preference and veto thresholds and binary informations specifying their optimisation direction, together with the evaluations of the alternatives on these criteria, the component computes and returns a set of relations, each one representing the discordance of a specific criterion on the set of alternatives pairs.

2.3.3 Outranking

The outranking component is able to compute a valued outranking relation described in the ELECTRE III method, as well as a binary outranking relation (obtained by "cutting" the ELECTRE III valued outranking relation at a given level). Using this second possibility permits to compute the ELECTRE Iv and II outranking relations. The component must be given a set of alternatives, criteria with applicable thresholds and weights, the evaluations of the alternatives on the criteria, and possibly a cutting threshold.

2.3.4 Flows

The PROMETHEE flows (positive flow, negative flow, net flow) can be computed with the flows component. It requires the same inputs as the concordance and outranking component, plus an information specifying which kind of flow is asked, and outputs a set of alternatives together with their flow values. Structures ranking the alternatives according to these scores are also available in the library.

3 USAGE PATTERNS

3.1 Components for the end-user

The end-user may use simple components, e.g., he may simply want to compute the concordance relation of ELECTRE III; he also can bind together different components in his own fashion and create his own MCDA procedure, for example to investigate "what-if" scenarios. The binding is made possible because of the inclusion of the components provided in J-MCDA in a software called diviz.

Diviz (Decision Deck Consortium, 2009a; Bigaret and Meyer, 2010) is another open source software part of the Decision Deck project. It provides a platform where users can graphically bind together components produced by different developers and thereby easily produce their own MCDA procedures or any type of workflow allowing to obtain a result by chaining (in the sense of a mathematical composition of functions) different transformations on some input data. The main requirement to enable participation of a component into this platform is that it takes as input, and produces, XMCDA files (Decision Deck Consortium, 2009c; Bisdorff et al., 2009). Thanks to this language the components share, diviz is able to use the output of one component as input to another one. The libraries provided comply with this requirement and have been made available for the diviz platform (as illustrated on Figure 1), together with specific documentation and appropriate packaging.

3.2 Components for the developer

Because the developed components does not simply compute the mathematical expressions proposed by the methods, but rather builds a whole set of easy to manipulate structures to represent the underlying concepts, the developer can easily use these libraries as a basis to develop other components. For example, he may want to provide another possibility of computing a discordance relation. He (or the end-user, through diviz) may then build a different assembly of components using his own ones together with existing ones, and use

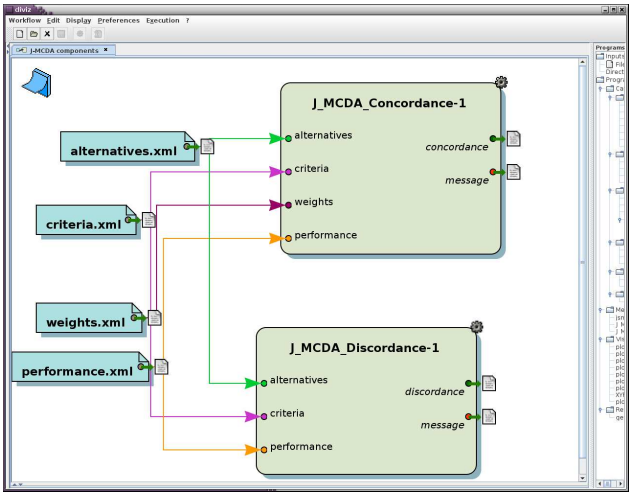

Figure 1: Two components provided by J-MCDA used in diviz.

J-XMCDA to parse the input and generate the output. He may also which to develop other MCDA concepts as Java objects, for example to support other MCDA procedures families. In such a case, he can easily build on top of J-XMCDA supplementary parsers able to convert between his own conceptual objects and XMCDA files.

The J-XMCDA library indeed makes use of a tool (Apache XMLBeans) assigning automatically generated Java classes equivalent to the concepts found in an XMCDA schema, and hence able to build Java objects representing the data found in any XMCDA file. It is then only a matter of traversing these objects and equating their contained informations to the simpler J-MCDA objects. This represents a loss of information compared to what can be stored in an XMCDA file because the XMCDA schema is not restricted to any family of MCDA procedures as J-MCDA is, the advantage being that the J-MCDA objects are much simpler to manage and descriptive enough when dealing only with outranking procedures.

The libraries feature functional tests (written in JUnit (Massol and Husted, 2003), the well known Java unit testing framework), giving the developer the ability to confidently refactor parts of the code for a better design should the need arise. The functional tests are based on XMCDA files featuring a specific example case. This example file and its associated Java classes and objects could be re-used by any developer or researcher to test other MCDA procedures and compare the results, if using software which, like diviz, uses XMCDA as a file format. The tests also have a documenting role, i.e. they provide working examples of code usage.

Another possible usage would be as an help for a developer to write file format converters, thanks to the J-XMCDA library. One way to write a software able to convert between XMCDA files and a software's proprietary format is first, to convert between the XMCDA file and a set of Java objects (representing, in an agnostic format, the equivalent MCDA data), and second, to be able to convert between these objects and the second format.

The first part of this job being implemented by the J-XMCDA and J-MCDA libraries, only the part specific to the legacy format must be dealt with by the developer. An example implementation of such a functionality is included in the library as a converter between the XMCDA and IRIS formats (IRIS (Dias and Mousseau, 2003) is a software for inferring ELECTRE TRI parameters given examples of alternative assignments into categories).

The software package, documentation, source code can be found on the Decision Deck website (Decision

Deck Consortium, 2010).

4 PERSPECTIVES

The J-MCDA library currently only includes those concepts appropriate to build the procedures described in this document (as well as a few matrix objects and operators). This could be extended by including other concepts such as categories (in a sorting perspective), assignment examples, kernel extraction from an outranking relation, etc. This would allow to include other MCDA procedures and add exploitation components (of an outranking relation, e.g.). XMCDA serialization and de-serialization (i.e., converting between XMCDA files and objects) functionalities could easily be added for these new Java objects thanks to the use of XMLBeans (described in Section 3.2).

Other MCDA procedures could be split into small components and implemented piece by piece, as has been done here for some of the ELECTRE and PROMETHEE procedures.

Care must be exercised however not to extend too much the scope of the J-MCDA library. Wanting it to be able to manage every existing MCDA procedures would make it heavyweight and very difficult to use. It would probably be a better option to create other, similar libraries, for each distinct MCDA procedures "family".

How to trace (probably not really existing) borders between these families, and especially splitting the procedures into components such that they can be best re-used to form a variety of existing and yet-to-be-invented MCDA procedures is an open question to the whole MCDA research community.

5 ACKNOWLEDGMENTS

Thanks are due to Yves De Smet for having initiated this work and to Vincent Mousseau for his interest in this project, and for their comments on early versions of this paper.

REFERENCES

Belton, V. and Stewart, T. (2002). *Multiple Criteria Decision Analysis: An Integrated Approach*. Kluwer Academic, Dordrecht.

Bigaret, S. and Meyer, P. (2010). diviz: an MCDA workflow design, execution and sharing tool. In *Proceedings of the Uncertainty and Robustness in Planning and Decision Making 2010 Conference*.

Bisdorff, R., Meyer, P., and Veneziano, T. (2009). XMCDA : a standard XML encoding of MCDA data. In *EURO XXIII : European conference on Operational Research*, pages 53–53.

Brans, J. and Mareschal, B. (2005). PROMETHEE methods. In Figueira, J., Greco, S., and Ehrgott, M., editors, *Multiple Criteria Decision Analysis: State of the Art Surveys*, pages 163–196. Springer Verlag.

Brans, J., Mareschal, B., and Vincke, P. (1984). PROMETHEE: a new family of outranking methods in multicriteria analysis. In Brans, J., editor, *Operational Research, IFORS 84*, page 477–490.

Brans, J. and Vincke, P. (1985). A preference ranking organization method. *Management Science*, 31(6):647–656.

Brans, J. P., Vincke, P., and Mareschal, B. (1986). How to select and how to rank projects: the PROMETHEE method. *European Journal of Operational Research*, 24:228–238.

Decision Deck Consortium (2009a). The diviz software. `http://www.decision-deck.org/diviz/`.

Decision Deck Consortium (2009b). Strategic manifesto of the Decision Deck project. http://www.decision-deck.org/tiki-index.php?page=Manifesto.

Decision Deck Consortium (2009c). XMCDA schema version 2.0.0. http://www.decision-deck.org/xmcda/.

Decision Deck Consortium (2010). Decision Deck website. http://www.decision-deck.org/.

Dias, L. and Mousseau, V. (2003). IRIS: a DSS for multiple criteria sorting problems. *Journal of Multi-Criteria Decision Analysis*, 12:285–298.

Figueira, J., Mousseau, V., and Roy, B. (2005). ELECTRE methods. In Figueira, J., Greco, S., and Ehrgott, M., editors, *Multiple Criteria Decision Analysis: State of the Art Surveys*, pages 133–162. Springer Verlag.

Massol, V. and Husted, T. (2003). *JUnit in Action*. Manning Publications.

Roy, B. (1991). The outranking approach and the foundations of ELECTRE methods. *Theory and Decision*, 31:49–73.

Vincke, P. (1999). Outranking approach. In Gal, T., Stewart, T., and Hanne, T., editors, *Multicriteria Decision Making, Advances in MCDM Models, Algorithms, Theory and Applications*. Kluwer Academic.

W3C Consortium (2004). XML schema part 0: Primer. http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/.

W3C Consortium (2008). Extensible Markup Language (XML) 1.0 (fifth edition). http://www.w3.org/TR/2008/REC-xml-20081126/.