# Understanding the mechanisms of how poly aluminium chloride inhibits short-chain fatty acids production from anaerobic fermentation of waste activated sludge


# PROMETHEE is Not Quadratic: An $\mathcal{O}(qn\log(n))$ Algorithm

Toon Calders, Dimitri Van Assche

PII: S0305-0483(17)30372-9
DOI: 10.1016/j.omega.2017.04.003
Reference: OME 1771

To appear in: *Omega*

Received date: 28 February 2016
Revised date: 14 April 2017
Accepted date: 16 April 2017

**Highlights**

- New method of computation of PROMETHEE using a sliding window after the alternatives have been sorted.
- Proof of efficiency of this method (complexity reduction from $O(n^2)$ to $O(n.\log(n))$).
- Performance comparison of this method versus the traditional one.

# PROMETHEE is Not Quadratic: An $\mathcal{O}(qn\log(n))$ Algorithm

Toon Calders[a,b], Dimitri Van Assche[b,*]

[a] Department of Mathematics and Computer Science, Universiteit Antwerpen
[b] Computer & Decision Engineering Department (CoDE), Université libre de Bruxelles

**Abstract**

It is generally believed that the preference ranking method PROMETHEE has a quadratic time complexity. In this paper, however, we present an exact algorithm that computes PROMETHEE's net flow scores in time $\mathcal{O}(qn\log(n))$, where $q$ represents the number of criteria and $n$ the number of alternatives. The method is based on first sorting the alternatives after which the unicriterion flow scores of all alternatives can be computed in one scan over the sorted list of alternatives while maintaining a sliding window. This method works with the linear and level criterion preference functions. The algorithm we present is exact and, due to the sub-quadratic time complexity, vastly extends the applicability of the PROMETHEE method. Experiments show that with the new algorithm, PROMETHEE can scale up to millions of tuples.

*Keywords:* Decision Support Systems, PROMETHEE, Multicriteria, Incremental Computing, Multicriteria Decision Analysis

## 1. Introduction

Multi-criteria decision aid (MCDA) [1], that is, the study of simultaneously evaluating possible decisions on multiple conflicting criteria, has been an active research field for over 40 years. A common example in this area is that of buying a new car. When selecting which car to buy, one typically tries to minimize cost and consumption while maximizing performance, comfort, etc. Obviously no real car can be best on all those criteria at the same time. Therefore, the notion of optimal solution necessarily is replaced by that of a compromise solution [2].

Different methods to select a compromise solution from a set of alternatives have been proposed in the literature, which can be divided into 3 main categories [3]: those based upon multi-attribute utility theory (MAUT) [4, 5], outranking methods [6], and interactive methods [7].

*Corresponding author
*Email addresses:* toon.calders@uantwerpen.be (Toon Calders), dvassche@ulb.ac.be (Dimitri Van Assche)

We only discuss the first two methods, as in the paper we only consider fully automatic ranking methods. In the methods based upon MAUT, scores are computed by aggregating, per alternative, its unicriterion utilities with functions that map a value into a preference degree. Most MAUT methods can therefore be computed efficiently, because the utility function is applied to each alternative for each criterion only once.

On the other hand, outranking methods such as ELECTRE [8] and PROMETHEE[1] [9] are based on pairwise comparisons of alternatives. These methods have a wide range of applications [10, 11, 12, 13], yet are often criticized for their high complexity due to pairwise comparisons. Indeed, the straightforward computation of these ranking methods leads to a quadratic complexity in the number of alternatives. As a result, for a small number of alternatives the straightforward implementation of these methods works fine, yet the performance degrades rapidly for an increasing number of alternatives.

In [14], a discussion is made about the performances of outranking methods in the context of geographical information analysis. Outranking methods are considered impractical when a large number of alternatives are involved, but there is no consensus on the meaning of large. According to [15]: "*Outranking techniques [...] require pairwise or global comparisons among alternatives, which is obviously impractical for applications where the number of alternatives/cells in a database range in the tens or hundreds of thousands.*" Furthermore, according to [16], the problem already arises with more than 100 alternatives: "*[...] outranking methods have difficulties dealing with more than a hundred alternatives.*" In [17] it is grossly overgeneralized that the evaluation of a large number of alternatives is an issue of multi-criteria analysis: "*One major drawback of MCA is that it does not allow the comparison of a large number of alternatives. With only a few alternatives to be evaluated, it is almost certain that the best alternative chosen from the set is in fact a sub-optimal solution.*"

In order to tackle these complexity problems, recently approximation methods have been developed to reduce the complexity of the calculation of PROMETHEE II by, for instance, the use of piecewise linear functions to approximate net flow scores of alternatives [18]. In this paper we show, however, that **efficient exact methods exist to compute PROMETHEE flow scores**. We present an exact method to compute flow scores of all alternatives considered for both PROMETHEE I and II, with time complexity $\mathcal{O}(qnlog(n))$ with $q$ the number of criteria and $n$ the number of alternatives. This is clearly a tremendous improvement over the $\mathcal{O}(qn^2)$ complexity of the straightforward computation by iterating over all pairs. Our method works for the two most popular preference functions: the linear[2] and level criterion preference functions (see figure 1 and 2). It requires, for each criterion, that the alternatives are sorted ascending according to this criterion using a standard sorting algorithm such as Merge Sort

[1] Preference Ranking Organization METHod for Enrichment of Evaluations

[2] We use the naming *linear preference function* for the *V-shape with indifference threshold preference function* (type 5 as defined by [9])

or Quick Sort. Subsequently a linear scan is made over the sorted alternatives while maintaining a sliding window. In this way, per criterion one unicriterion flow score is computed and these scores are then combined in the final PROMETHEE flow score. The key observation in determining the complexity of this procedure is that sorting the dataset on a criterion takes time $\mathcal{O}(n \log(n))$ using a standard out-of-the-box sorting algorithm such as merge-sort.

The paper is organized as follows. In Section 2, we revisit the PROMETHEE ranking method. In Section 3, we introduce our efficient sorting-based approach for calculating the flow scores of PROMETHEE, and establish its time complexity. Finally, in Section 4, we compare the performance of the straightforward quadratic implementation with our new method, clearly illustrating the dramatic performance improvement, scaling up PROMETHEE to applications with millions of alternatives.

## 2. PROMETHEE

In this section, we revisit the definition of the PROMETHEE I and II scoring methods. We refer the interested reader to [9] for a detailed description of PROMETHEE.

Let $A = \{a_1, a_2 ..., a_n\}$ be a set of $n$ alternatives and let $F = \{f_1, f_2 ..., f_q\}$ be a set of $q$ criteria. The evaluation of alternative $a_i$ for criterion $k$ will be denoted by a real value $f_k(a_i)$. We assume, without loss of generality, that a higher value for a criterion is better. For each pair of alternatives, we define $d_k(a_i, a_j)$ as the difference between $a_i$ and $a_j$ on criterion $k$.

$$d_k(a_i, a_j) := f_k(a_i) - f_k(a_j) \ . \tag{1}$$

A preference function, denoted $P_k$, is associated with each criterion $k$. This function transforms the difference $d_k(a_i, a_j)$ between alternatives into a preference degree of $a_i$ over $a_j$ on criterion $k$. Multiple preference functions exist [9], such as the *linear preference function* (Equation 3, Figure 1) and the level criterion preference function (Equation 2, Figure 2).

$$P_k[d_k(a_i, a_j)] = \begin{cases} 0 & \text{if } d_k(a_i, a_j) < q_k \\ \frac{1}{2} & \text{if } q_k \leq d_k(a_i, a_j) \leq p_k \\ 1 & \text{if } p_k < d_k(a_i, a_j) \end{cases} \tag{2}$$

In this paper we will consider the *linear preference function*, which is defined as follows, given an indifference threshold $q_k$ and a preference threshold $p_k$:

$$P_k[d_k(a_i, a_j)] = \begin{cases} 0 & \text{if } d_k(a_i, a_j) < q_k \\ \frac{d_k(a_i, a_j) - q_k}{p_k - q_k} & \text{if } q_k \leq d_k(a_i, a_j) \leq p_k \\ 1 & \text{if } p_k < d_k(a_i, a_j) \end{cases} \tag{3}$$

The results in this paper can easily be extended to the level criterion preference function as well. These two preference functions represent the most popular choices in the literature.

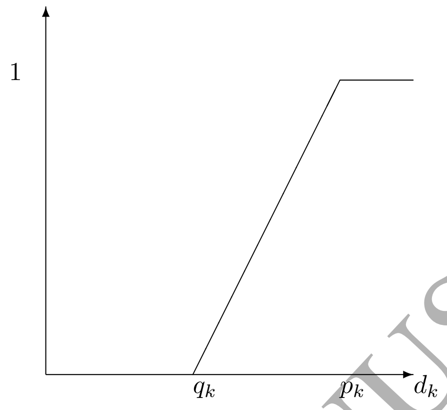

Figure 1: Linear preference function

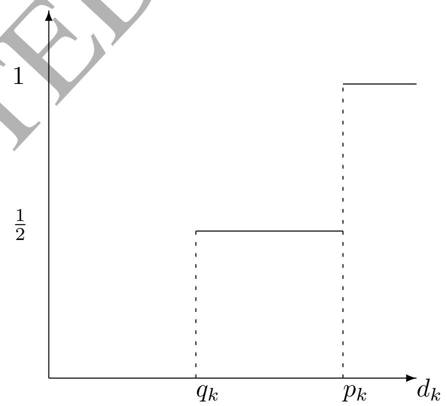

Figure 2: Level criterion preference function

By applying the preference function $P_k$ to the difference $d_k$, we get the unicriterion preference degree $\pi_k$:

$$\pi_k(a_i, a_j) := P_k[d_k(a_i, a_j)] \tag{4}$$

The aggregated preference degree of alternative $a_i$ over $a_j$ is then computed as a weighted sum over the unicriterion preferences using weights $w_k$ associated with each criterion $k$. Weights are assumed to be positive and normalized.

$$\pi(a_i, a_j) := \sum_{k=1}^{q} w_k \pi_k(a_i, a_j) \tag{5}$$

The last step consists in calculating the positive flow score denoted $\phi^+(a_i)$ and the negative flow score denoted $\phi^-(a_i)$ which are combined into the final flow score of $a_i$ as follows:

$$\phi^+(a_i) := \frac{1}{n-1} \sum_{x \in A} \pi(a_i, x) \tag{6}$$

$$\phi^-(a_i) := \frac{1}{n-1} \sum_{x \in A} \pi(x, a_i) \tag{7}$$

$$\phi(a_i) := \phi^+(a_i) - \phi^-(a_i) \tag{8}$$

The PROMETHEE I (partial) ranking is obtained as the intersection of the rankings induced by $\phi^+$ and $\phi^-$. PROMETHEE II gives a complete ranking induced by $\phi$. For an interpretation of the net flow scores, the interested reader is referred to [19].

In this paper we will use the following, equivalent definition of the flow score:

$$\phi_k^+(a_i) := \frac{1}{n-1} \sum_{x \in A} \pi_k(a_i, x) \tag{9}$$

$$\phi_k^-(a_i) := \frac{1}{n-1} \sum_{x \in A} \pi_k(x, a_i) \tag{10}$$

$$\phi_k(a_i) := \phi_k^+(a_i) - \phi_k^-(a_i) \tag{11}$$

$$\phi(a_i) = \sum_{k=1}^{q} w_k \phi_k(a_i) \tag{12}$$

$\phi_k^+(a_i)$ and $\phi_k^-(a_i)$ are respectively called the *unicriterion positive flow score* and *unicriterion negative flow score* of the alternative $a_i$ on criterion $k$.

## 3. Sorting-Based Algorithm for PROMETHEE

Our sorting-based algorithm is based on computing the unicriterion positive and negative flow scores $\phi_k^+(a)$ and $\phi_k^-(a)$ for all alternatives $a$ for each criterion $f_k$ separately. The time required for this computation will be shown to be $\mathcal{O}(n\log(n))$ per criterion. Hence, computing all unicriterion positive and negative flow scores for all criteria takes time $\mathcal{O}(qn\log(n))$. These scores can be combined into the final PROMETHEE II flow score in time $\mathcal{O}(qn)$, leading to an overall time complexity of $\mathcal{O}(qn\log(n))$.

Given the symmetry of the expressions for $\phi_k^+(a)$ and $\phi_k^-(a)$, we will only elaborate on the computation of $\phi_k^+(a)$. Indeed, if we substitute $f_k(a)$ by $f'_k(a) = -f_k(a)$, the resulting $\phi'^+_k(a) = \phi_k^-(a)$. This is quite easy to see since for all $k$, $x$, and $a_i$ it holds that $d'_k(a_i, x) = d_k(x, a_i)$, and consequently $\pi'_k(a_i, x) = \pi_k(x, a_i)$.

As such, exactly the same algorithm can be used to compute $\phi_k^+(a)$ for all $a$ as for computing $\phi_k^-(a)$ for all $a$. In the following we only present the algorithm for the linear preference function, as the algorithm for the level criterion preference function is very similar.

To illustrate the sorting-based computation of the unicriterion positive flow function, we use the following running example.

**Example 1.** *Consider the following table listing a number of destinations (the alternatives), together with their characteristics (the criteria) on the basis of which we want to decide where to go on vacation. We consider three criteria: hours of sunshine, price, and historical value, each of which have been marked on a numerical scale of 1 to 10.*

| | Destination | $f_1$ Sunshine | $f_2$ Price | $f_3$ History |
|---|---|---|---|---|
| $a_1$ | *Brussels* | 5 | 6 | 9 |
| $a_2$ | *Paris* | 6 | 4 | 10 |
| $a_3$ | *Blois* | 7 | 9 | 8 |
| $a_4$ | *Berlin* | 7 | 8 | 8 |
| $a_5$ | *Barcelona* | 10 | 7 | 7 |

*Furthermore for all $k = 1, \ldots, 3$, the thresholds $q_k$ and $p_k$ are respectively* 1 *and* 3. *This means that a difference of* 1 *score point or less will be ignored, and a difference of more than* 3 *points will no longer give additional benefit. In between these two extremes, the contribution to the flow score will be linear.*

*With these parameters, the positive unicriterion flow score for criterion $f_1$ for alternative $a_4$ is:*

$$\begin{aligned}\phi_1^+(a_4) &= \frac{1}{4}(\pi_1(a_4,a_1) + \pi_1(a_4,a_2) + \pi_1(a_4,a_3) + \pi_1(a_4,a_4) + \pi_1(a_4,a_5)) \\ &= \frac{1}{4}\left(\frac{1}{2} + 0 + 0 + 0 + 0\right) = \frac{1}{8}\end{aligned}$$

### *3.1. Window of an Alternative*

One of the observations on which our algorithm for the linear preference function is based, is the following: There are three different ways in which an

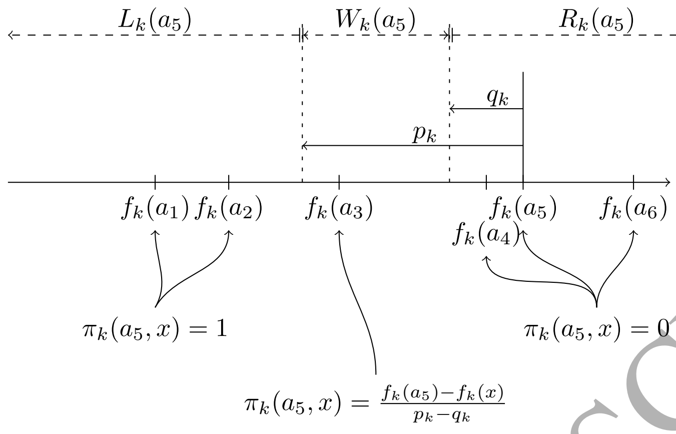


Figure 3: Illustration of the three different cases for the preference degree $\pi_k(a_5, x)$: $x \in L_k(a_5)$, $x \in W_k(a_5)$, and $x \in R_k(a_5)$

alternative $x$ can influence $\phi_k^+(a)$ (see figure 3):

$$\begin{aligned} f_k(x) < f_k(a) - p_k \quad &: \quad \pi_k(a,x) = 1 && (13)\\ f_k(a) - p_k \leq f_k(x) \leq f_k(a) - q_k \quad &: \quad \pi_k(a,x) \text{ linear in } d_k(a,x) && (14)\\ f_k(a) - q_k < f_k(x) \quad &: \quad \pi_k(a,x) = 0 && (15) \end{aligned}$$

Hence, except for a window of length $p_k - q_k$, the influence of an alternative $x$ is a default value that depends on the side of the window on which $x$ falls. We formalize these three regions as follows:

**Definition 1.** *The* window of $a_i$*, denoted* $W_k(a_i)$ *is defined as:*

$$W_k(a_i) := \{x \in A \mid f_k(x) \in [f_k(a_i) - p_k, f_k(a_i) - q_k]\} \; .$$

*We will use* $l_i$*, respectively* $u_i$ *to denote the lower and upper bounds of the interval* $[f_k(a_i) - p_k, f_k(a_i) - q_k]$*. We say that an* alternative $x$ is left of $W_k(a_i)$ *if* $f_k(x) < l_i$*, and that it is right of* $W_k(a_i)$ *if* $f_k(x) > u_i$*. The set of all alternatives left, respectively right of* $a_i$ *will be denoted* $L_k(a_i)$*, respectively* $R_k(a_i)$*.*

**Example 2.** *We continue Example 1. We illustrate the concept of window by looking at the first criterion "Sunshine." The alternatives are already ordered according to increasing score for "Sunshine." The window* $W_1(a_1)$ *includes all alternatives with a temperature value that is in the interval* $[5-3, 5-1] = [2, 4]$*. Hence* $W_1(a_1) = \emptyset$*,* $L_1(a_1) = \emptyset$*, and* $R_1(a_1) = \{a_1, a_2, a_3, a_4, a_5\}$*. The other* *windows are as follows:*

| $a_i$ | $[l_i, u_i]$ | $L_1(a_i)$ | $W_1(a_i)$ | $R_1(a_i)$ |
|---|---|---|---|---|
| $a_1$ | $[2,4]$ | $\emptyset$ | $\emptyset$ | $\{a_1,a_2,a_3,a_4,a_5\}$ |
| $a_2$ | $[3,5]$ | $\emptyset$ | $\{a_1\}$ | $\{a_2,a_3,a_4,a_5\}$ |
| $a_3$ | $[4,6]$ | $\emptyset$ | $\{a_1,a_2\}$ | $\{a_3,a_4,a_5\}$ |
| $a_4$ | $[4,6]$ | $\emptyset$ | $\{a_1,a_2\}$ | $\{a_3,a_4,a_5\}$ |
| $a_5$ | $[7,9]$ | $\{a_1,a_2\}$ | $\{a_3,a_4\}$ | $\{a_5\}$ |

Using these definitions, we obtain the following equality:

$$\begin{aligned}\phi_k^+(a) &= \frac{1}{n-1}\left(\sum_{x\in L_k(a)} 1 + \sum_{x\in W_k(a)} \frac{(f_k(a)-f_k(x))-q_k}{p_k-q_k} + \sum_{x\in R_k(a)} 0\right) \\ &= \frac{1}{n-1}\left(|L_k(a)| + |W_k(a)| \times \frac{f_k(a)-q_k}{p_k-q_k} - \frac{\sum_{x\in W_k(a)} f_k(x)}{p_k-q_k}\right) \quad (16)\end{aligned}$$

Hence, in order to compute $\phi_k^+(a)$ for all $a$ efficiently, it is essential to be able to quickly compute $|W_k(a)|$, $|L_k(a)|$, and the sum $S_k(a) := \sum_{x\in W_k(a)} f_k(x)$ for all alternatives $a$. This can be achieved by a sorting-based method that incrementally computes $|W_k(a_i)|$, $|L_k(a_i)|$ and $S_k(a_i)$, as we show next.

*3.2. Incremental Computation of the Windows $W_k(a)$*

Let $a_1, a_2, \ldots, a_n$ be the sequence of alternatives, ordered in ascending order of their $k$-th criterion; that is, for all $i = 1\ldots n-1$, $f_k(a_i) \leq f_k(a_{i+1})$. After ordering the alternatives, it holds that if $i < j$, $W_k(a_i)$ will be "more to the left" than $W_k(a_j)$ in the sense that $l_i \leq l_j$ and $u_i \leq u_j$. The following lemma formalizes this observation. We exploit this fact for computing the windows $W_k(a_i)$ incrementally.

**Lemma 1.** *Let $A = \{a_1, \ldots, a_n\}$ be the set of alternatives ordered in ascending order w.r.t. the $k$-th criterion; i.e., $f_k(a_i) \leq f_k(a_{i+1})$ for all $i = 1\ldots n-1$. Let $x \in A$, $i < j \in [1, n]$. If $x \in W_k(a_i) \cap W_k(a_j)$, then for all $i \leq \ell \leq j$, it holds that $x \in W_k(a_\ell)$. Furthermore, if both $a_r$ and $a_s$ are in $W_k(a_i)$, then also for all $r \leq \ell \leq s$, $a_\ell \in W_k(a_i)$.*

*Proof.* $x \in W_k(a_i) \cap W_k(a_j)$ implies that $f_k(x) \in [l_i, u_i] \cap [l_j, u_j]$. Let now $i < \ell < j$, then we have the following inequalities:

$$\begin{aligned} l_\ell &= f_k(a_\ell) - p_k && (17)\\ &\leq f_k(a_j) - p_k && (18)\\ &= l_j && (19)\\ &\leq f_k(x) && (20)\\ &\leq u_i && (21)\\ &= f_k(a_i) - q_k && (22)\\ &\leq f_k(a_\ell) - q_k && (23)\\ &= u_\ell && (24)\end{aligned}$$

Hence, $l_\ell \le f_k(x) \le u_\ell$ and thus $x \in W_k(a_\ell)$. (17), (19), (22), and (24) follow from the definition of $W_k$ and the lower and upper bounds of that window. (18) and (23) follow from $i < \ell < j$ and the alternatives being ordered ascending with respect to $f_k$; hence $f_k(a_i) \le f_k(a_\ell) \le f_k(a_j)$. (20) and (21) follow from $x \in W_k(a_i) \cap W_k(a_j) = [l_i, u_i] \cap [l_j, u_j]$. Because $f_k(a_i) \le f_k(a_j)$, $l_i \le l_j$ and $u_i \le u_j$, from which we derive that $[l_i, u_i] \cap [l_j, u_j] = [l_j, u_i]$.

We prove the second statement using similar techniques as follows. Let $a_r$ and $a_s$ with $r \le s$ be in $W_k(a_i)$, and $r \le \ell \le s$.

$$\begin{aligned} l_i &\le f_k(a_r) && (25)\\ &\le f_k(a_\ell) && (26)\\ &\le f_k(a_s) && (27)\\ &\le u_i && (28) \end{aligned}$$

Hence $a_\ell \in W_k(a_i)$. (25) and (28) follow from $a_r, a_s \in W_k(a_i)$ and (26) and (27) from $r \le \ell \le s$ and hence $f_k(a_r) \le f_k(a_\ell) \le f_k(a_s)$. □

**Example 3.** *Figure 4 illustrates the observation for the dataset of Example 1 and the first criterion.*

*It is clear that $l_1 \le l_2 \le l_3 \le l_4 \le l_5$ and $u_1 \le u_2 \le u_3 \le u_4 \le u_5$, and for instance, because $a_1$ is in $W_1(a_2) \cap W_1(a_4)$, $a_1$ is also in $W_1(a_3)$.*

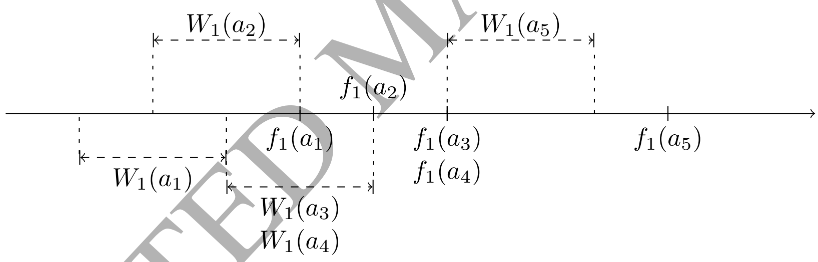


Figure 4: windows $W_1$ for each alternative $a_i$ on the first criterion from the example

From the lemma it is clear that if we construct the windows in order from $W_k(a_1)$ to $W_k(a_n)$, every time an element $x$ leaves a window, we will never have to reconsider it; that is, if $x \in W_k(a_i)$, but $x \notin W_k(a_{i+1})$, then for all $j = i+1 \ldots n$, $x \in L_k(a_j)$. This gives rise to the following incremental algorithm: when computing $W_k(a_i)$ from $W_k(a_{i-1})$, we first shift the leftmost border of the window; we increment the index $\lambda$ until $\lambda = n + 1$ or $f_k(a_\lambda) \ge l_i$, followed by shifting the rightmost border of the window by incrementing index $\upsilon$ until $\upsilon = n$ or $f_k(a_{\upsilon+1}) > u_i$. The resulting algorithm is given as Algorithm 1.

**Example 4.** *We continue our running example 2 and show how the subsequent windows can be computed for the criterion* Sunshine. *First we order the alternatives in ascending order according to this criterion, which gives us the list*

*Brussels - Paris - Blois - Berlin - Barcelona. $L$ and $W$ are initialized to $\emptyset$ and $R = A$. We start by constructing the window for Brussels. The value for the criterion in Brussels is $5$, and hence the interval is $[2, 4]$. Since there are no elements in $W$ yet, we can skip the test for removing elements from $W$. We test the first element of $R$, being Brussels, and find out that it does not fall into the window. Hence it remains in $R$ and we have completed the computation for Brussels. Then we go to the next element, Paris. Paris has $6$ hours of sunshine, so the window boundaries become $[3, 5]$. Again, since $W$ is empty, we do not need to remove elements from $W$. This time, however, the first element Brussels of $R$ does fall into the window and hence Brussels is removed from $R$ and added to $W$. The second element Paris does not fall into the window and thus remains in $W$. Then we compute the window for Blois. The first element of the current $W$ (Brussels) remains in $W$ and the first element of $R$ (Paris) is added to $W$ and removed from $R$. The next element in $R$, Blois, stays in $R$. As the lower and upper bounds for Berlin are the same as for Blois, nothing changes and we can go to the computation of the windows of the last alternative Barcelona. For Barcelona, the first and second element of $W$ are expelled as they do not fall within the interval $[7, 9]$. The first two elements of the current $R$ (Blois, Berlin), however, do and are entered into $W$ and removed from $R$. In this way we get subsequently exactly the windows we got in Example 2.*

The key observation in determining the complexity of this procedure is that sorting the dataset on criterion $q_k$ is executed once and takes time $\mathcal{O}(n\log(n))$ using standard sorting algorithms such as merge-sort. Furthermore, lines $9, 10$ are executed only if the test $f_k(W[1]) < l$ succeeds, which is at most $n+1$ times over *all* incremental computations, because every time the condition is true an element is removed from $W$, $W$ is always a subset of $A$, and every element removed from $W$ will never enter $W$ again. For the lines $13-18$ exactly the same argumentation holds; every time the test succeeds, an element is removed from $R$ and elements removed from $R$ do never enter $R$ again. All the other steps of the algorithm are executed either exactly once (before for-loop), or exactly $n$ times (inside the for-loop). As such the time complexity of the algorithm is $\mathcal{O}(n\log(n) + n) = \mathcal{O}(n\log(n))$.

### 3.3. Incremental computation of $S_k(a) = \sum_{x\in W_k(a)} f_k(x)$

For computing the sum $S_k(a_i)$ from $S_k(a_{i-1})$, it suffices to look at the differences between $S_k(a_i)$ and $S_k(a_{i-1})$:

$$S_k(a_i) - S_k(a_{i-1}) = \sum_{x\in W_k(a_i)} f_k(x) - \sum_{x\in W_k(a_{i-1})} f_k(x) \tag{29}$$

$$= \sum_{x\in W_k(a_i)\setminus W_k(a_{i-1})} f_k(x) - \sum_{x\in W_k(a_{i-1})\setminus W_k(a_i)} f_k(x) \tag{30}$$

$$= \sum_{x\in W_k(a_i)\cap R_k(a_{i-1})} f_k(x) - \sum_{x\in W_k(a_{i-1})\cap L_k(a_i)} f_k(x) \tag{31}$$

**Input**: Set of alternatives $A$ of size $N$, criterion $f_k$ and thresholds $p_k, q_k$.
**Output**: List of cardinalities $(|W_k(a_i)|, |R_k(a_i)|)$, $i = 1 \dots N$

1 $sort(A, f_k)$ ; `// Sort` $A$ `ascending w.r.t.` $f_k$
2 $W \leftarrow []$ ; `// Initialize` $W$ `to an empty list`
3 $R \leftarrow sorted(A, f_k)$ ; `// List` $R$ `is` $A$ `sorted ascending w.r.t.` $f_k$
4 $card_L = 0$ ; $card_W = 0$ ;
5 **for** $i$=*1...N* **do**
6     $a \leftarrow A[i]$ ; `// Alternative for which we are computing windows`
7     $l \leftarrow f_k(a) - p_k$; $u \leftarrow f_k(a) - q_k$;
8     **while** $|W| \geq 1$ *and* $f_k(W[1]) < l$ **do**
9         remove first element from $W$ ; `// First element leaves` $W$
10         $card_W \leftarrow card_W - 1$ ; $card_L \leftarrow card_L + 1$ ;
11     **end**
12     **while** $|R| \geq 1$ *and* $f_k(R[1]) < u$ **do**
13         $x \leftarrow R[1]$ ; remove $x$ from $R$ ; `// Element leaves` $R$
14         **if** $f_k(x) \geq l$ **then**
15             append $x$ at the end of $W$ ; `// Element enters` $W$
16             $card_W \leftarrow card_W + 1$ ;
17         **else**
18             $card_L \leftarrow card_L + 1$ ;
19         **end**
20     **end**
21     **output** $(card_W, card_L)$ ;
22 **end**

**Algorithm 1:** Compute $W_k(a_i)$ and $L_k(a_i)$ for all $a_i$.

In other words, when incrementally computing the window $W_k(a_i)$ from $W_k(a_{i-1})$, we should only keep track of the alternatives leaving $W_k(a_{i-1})$ (step 9 of Algorithm 1), and those entering $W_k(a_i)$ (step 16 of Algorithm 1).

**Example 5.** *We continue example 4. When computing $W_k(a_5)$ from $W_k(a_4)$, we removed $a_1$ and $a_2$ from $W$ and added $a_3$ and $a_4$. Therefore, to get $S_k(a_5)$ from $S_k(a_4)$ we need to subtract the contributions from $a_1$ and $a_2$, and add the contributions of $a_3$ and $a_4$. Hence,*

$$S_k(a_5) - S_k(a_4) \;=\; f_k(a_3) + f_k(a_4) - f_k(a_1) - f_k(a_2) \;\;.$$

### *3.4. Complete Algorithm*

By combining the incremental computation of the windows and that of the sums, we obtain Algorithm 2 to compute the unicriterion positive flow scores of a set of alternatives. In this algorithm we consider the use of queue data structures to store the window $W$ and the set $R$ of alternatives to the right. These structures act as a *LIFO* (Last In First Out) store. We retrieve elements

**Input**: Set of alternatives $A$ of size $n$, criterion $f_k$ and thresholds $p_k, q_k$.
**Output**: $\phi^+(a_i)$ for all $i = 1 \dots n$

1 $sort(A, f_k)$ ; `// Sort` $A$ `ascending w.r.t.` $f_k$
2 $W \leftarrow []$ ; `// Initialize` $W$ `to an empty list`
3 $R \leftarrow sorted(A, f_k)$ ; `// List` $R$ `is` $A$ `sorted ascending w.r.t.` $f_k$
4 $card_L = 0$ ; $card_W = 0$ ; $S = 0$ ;
5 **for** $i$=$1 \dots n$ **do**
6   $a \leftarrow A[i]$ ; `// Alternative for which we compute flow`
7   $l \leftarrow f_k(a) - p_k$; $u \leftarrow f_k(a) - q_k$;
8   **while** $|W| \geq 1$ *and* $f_k(W[1]) < l$ **do**
9     $x \leftarrow W[1]$ ;
10     remove $x$ from $W$ ; `// Remove first element` $W$
11     $card_W \leftarrow card_W - 1$ ; $card_L \leftarrow card_L + 1$ ;
12     $S \leftarrow S - f_k(x)$ ; `// Remove contribution of` $x$
13   **end**
14   **while** $|R| \geq 1$ *and* $f_k(R[1]) < u$ **do**
15     $x \leftarrow R[1]$ ;
16     remove $x$ from $R$ ; `// Remove first element` $R$
17     **if** $f_k(x) \geq l$ **then**
18       append $x$ at the end of $W$ ; `// Element enters` $W$
19       $card_W \leftarrow card_W + 1$ ;
20       $S \leftarrow S + f_k(x)$ ; `// Add contribution of` $x$
21     **else**
22       $card_L \leftarrow card_L + 1$ ;
23     **end**
24   **end**
25   $\phi^+(a) \leftarrow \frac{1}{n-1}\left(card_L + card_W \times \frac{f_k(a)-q_k}{p_k-q_k} - \frac{S}{p_k-q_k}\right)$ ; `// Eq. (16)`
26 **end**

**Algorithm 2:** Sorting-Based Computation of the Uni-Criterion Positive Flow Score

from the front of the queue, and add items to the end of the queue. The pseudocode is very similar to that of 1, but now with the incremental computation of the sum inserted in those places where elements move in and out of the window $W$, and with the computation of the positive flow score based on Equation (16).

**Example 6.** *We illustrate Algorithm 2 by tracing the values of the main variables at critical points in the program; that is: before executing the first while loop (line 8), in between the two while loops (line 14), and when the positive flow of the alternative under consideration is finally computed (line 25).*

*When processing the first alternative, none of the alternatives moves from* $R$ *to* $W$ *and hence all cardinalities and the sum remain* 0 *resulting in a positive flow of* 0*:*

| $i$ | *line* | $a$ | $l$ | $u$ | $f_k(a)$ | $W$ | $R$ | $\|L\|$ | $\|W\|$ | $S$ |
|---|---|---|---|---|---|---|---|---|---|---|
| *1* | *8* | *Bruss.* | *2* | *4* | 5 | $\emptyset$ | $[a_1, a_2, a_3, a_4, a_5]$ | *0* | *0* | *0* |
| | *14* | | | | | $\emptyset$ | $[a_1, a_2, a_3, a_4, a_5]$ | *0* | *0* | *0* |
| | *25* | | | | | $\emptyset$ | $[a_1, a_2, a_3, a_4, a_5]$ | *0* | *0* | *0* |
| | | $\phi^+(a_1) = \frac{1}{4}(0 + 0 - 0) = 0$ | | | | | | | | |

*When processing the second alternative, $a_1$ moves from $R$ to $W$. The cardinality of $W$ and the sum are updated accordingly. The positive flow for $a_2$ is computed:*

| $i$ | *line* | $a$ | $l$ | $u$ | $f_k(a)$ | $W$ | $R$ | $\|L\|$ | $\|W\|$ | $S$ |
|---|---|---|---|---|---|---|---|---|---|---|
| *2* | *8* | *Paris* | *3* | *5* | 6 | $\emptyset$ | $[a_1, a_2, a_3, a_4, a_5]$ | *0* | *0* | *0* |
| | *14* | | | | | $\emptyset$ | $[a_1, a_2, a_3, a_4, a_5]$ | *0* | *0* | *0* |
| | *25* | | | | | $[a_1]$ | $[a_2, a_3, a_4, a_5]$ | *0* | *1* | *5* |
| | | $\phi^+(a_2) = \frac{1}{4}(0 + 1 \times \frac{6-1}{3-1} - \frac{5}{3-1}) = 0$ | | | | | | | | |

*When processing the third alternative, $a_1$ remains in $W$ and $a_2$ moves from $R$ to $W$. The cardinality of $W$ and the sum are updated accordingly. The positive flow for $a_3$ is computed:*

| $i$ | *line* | $a$ | $l$ | $u$ | $f_k(a)$ | $W$ | $R$ | $\|L\|$ | $\|W\|$ | $S$ |
|---|---|---|---|---|---|---|---|---|---|---|
| *3* | *8* | *Blois* | *4* | *6* | *7* | $[a_1]$ | $[a_2, a_3, a_4, a_5]$ | *0* | *1* | *5* |
| | *14* | | | | | $[a_1]$ | $[a_2, a_3, a_4, a_5]$ | *0* | *1* | *5* |
| | *25* | | | | | $[a_1, a_2]$ | $[a_3, a_4, a_5]$ | *0* | *2* | *11* |
| | | $\phi^+(a_3) = \frac{1}{4}(0 + 2 \times \frac{7-1}{3-1} - \frac{11}{3-1}) = \frac{1}{8}$ | | | | | | | | |

*During the processing of alternative $a_4$ nothing changes since this alternative has exactly the same value for the criterion and hence also the same flow value $\frac{1}{8}$. Hence we move directly to the computation for the last alternative. Here alternatives $a_1$ and $a_2$ move from $W$ to $L$ and $a_3$ and $a_4$ move from $R$ to $W$:*

| $i$ | *line* | $a$ | $l$ | $u$ | $f_k(a)$ | $W$ | $R$ | $\|L\|$ | $\|W\|$ | $S$ |
|---|---|---|---|---|---|---|---|---|---|---|
| *3* | *8* | *Bcn* | *7* | *9* | *10* | $[a_1, a_2]$ | $[a_3, a_4, a_5]$ | *0* | *2* | *11* |
| | *14* | | | | | $\emptyset$ | $[a_3, a_4, a_5]$ | *2* | *0* | *0* |
| | *25* | | | | | $[a_3, a_4]$ | $[a_5]$ | *2* | *2* | *14* |
| | | $\phi^+(a_5) = \frac{1}{4}(2 + 2 \times \frac{10-1}{3-1} - \frac{14}{3-1}) = 1$ | | | | | | | | |

## 4. Empirical validation

We compare the performance of our sorting-based algorithm, that we will call *Sorting Based PROMETHEE* (SBP), with that of the straightforward algorithm based on iterating over all pairs of alternatives[3]. Even though from the theoretical analysis it is crystal clear that the $\mathcal{O}(qn\log(n))$ sorting-based method will outperform the $\mathcal{O}(qn^2)$ standard method, we want to illustrate with these experiments how large the actual difference is and to what new problem sizes PROMETHEE II can be scaled. We do not make any comparison of the ranking produced since the sorting-based method is exact and hence the

[3]All code used in the comparisons in this section is avaiable at: https://github.com/SomeULB/prometheenlogn

flow scores and ranking produced is always exact. The tests have been executed on a computer with an Intel Core 2 Quad 2.0 GHz and 4 Gb of RAM. The data we use is unicriterion and was generated using a random uniform distribution of values to have as many different values as possible.

Figure 5 shows runtime versus number of alternatives for the standard implementation of PROMETHEE, and for the new sorting-based implementation. The time to compute the unicriterion net flow scores is shown on the Y axis and the number of alternatives to be ranked on the X axis. Note that the scale on the X axes (number of alternatives) is vastly different because the standard method was not able to scale to large sizes. To illustrate the difference, the rightmost point in the left figure shows that the standard evaluation method for PROMETHEE for 25000 alternatives takes *over half an hour* while for the sorting-based method this point is on the far left in the graph and takes *less than half a second!*

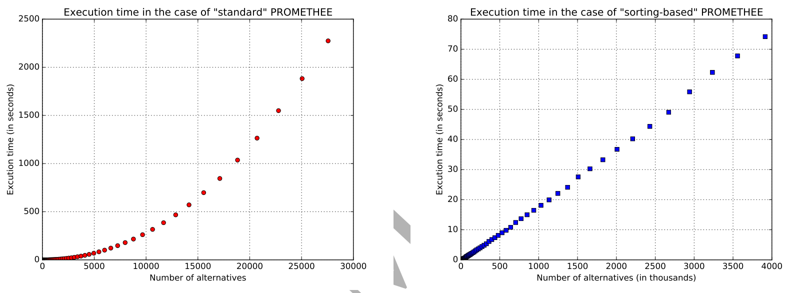


Figure 5: Execution time with respect to the number of alternatives in the case of “standard” PROMETHEE (left) versu the new sorting-based PROMETHEE (right). Note the difference in scale of the graphs.

Table 1 illustrates the computation time’s increase in function of the number of alternatives. Every next line the number of alternatives considered is multiplied by 2. The table includes the computation times required to compute the unicriterion net flow scores of those sets of alternatives. For standard PROMETHEE, the execution time increases by a factor 4, as expected due to the complexity in $\mathcal{O}(n^2)$. For the incremental version, however, it increases by a factor only slightly larger than 2.

For a small number of alternatives, both methods perform satisfactorily, but when the number increases the difference between them increases rapidly. Indeed, for more than $2^{16}$ alternatives, the standard version needs more than 1 hour to compute all the scores while the incremental one requires less than a second!

## 5. Conclusion

In this paper, we have presented a method that reduces the computation time complexity of PROMETHEE in the case of the linear preference function from

| # alternatives | Standard PROMETHEE | Incremental PROMETHEE |
|---|---|---|
| 2 | 3e-05 | 5e-05 |
| 4 | 7e-05 | 6e-05 |
| 8 | 0.00021 | 0.0001 |
| 16 | 0.00078 | 0.00017 |
| 32 | 0.003 | 0.00032 |
| 64 | 0.012 | 0.0006 |
| 128 | 0.047 | 0.0012 |
| 256 | 0.19 | 0.0024 |
| 512 | 0.75 | 0.0048 |
| 1024 | 2.98 | 0.0097 |
| 2048 | 11.88 | 0.02 |
| 4096 | 47.55 | 0.04 |
| 8192 | 190 | 0.08 |
| 16384 | 775 | 0.17 |
| 32768 | 3216 | 0.35 |
| 65536 | 12876 | 0.74 |
| 131072 | NA | 1.56 |
| 262144 | NA | 3.34 |
| 524288 | NA | 7.04 |
| 1048576 | NA | 14.7 |
| 2097152 | NA | 30.5 |
| 4194304 | NA | 64.2 |

Table 1: Execution time in seconds for the standard and incremental PROMETHEE.

$\mathcal{O}(qn^2)$ to $\mathcal{O}(qn \log(n))$. The method is based on an incremental computation after first sorting the alternatives for each criterion. With this new algorithm SBP we have presented, the complexity problem is solved for PROMETHEE; it can now be applied to large sets of alternatives requiring only a small amount of time. The bottom line is: if the dataset can be sorted, PROMETHEE can be applied.

This opens up questions on the interpretation of unicriterion flow scores. Indeed, we have a formulation of $\phi^+(a_i)$ in function of $\phi^+(a_{i-1})$. We can see that with the linear preference function, the difference in scores between $a_i$ and $a_{i-1}$ only depends on the alternatives that were in the windows of $a_i$ and $a_{i-1}$. This could be explored in future research.

Although we have developed our algorithm for the linear preference function, it can easily be extended to the level criterion preference function. The method, however, does not work for the Gaussian preference function. Hence, developing a similarly improved version of PROMETHEE with the Gaussian preference function remains future work as well.

**References**


[1] S. Greco, J. Figueira, M. Ehrgott (Eds.), Multiple Criteria Decision Analysis: State of the Art Surveys, International Series in Operations Research & Management Science, Springer, Berlin, 2016.

[2] B. Roy, P. Vincke, Multicriteria analysis: survey and new directions, European Journal of Operational Research 8 (3) (1981) 207–218.

[3] B. Roy, Paradigms and challenges, in: J. Figueira, S. Greco, M. Ehrogott (Eds.), Multiple criteria decision analysis: State of the art surveys, Springer, 2005, pp. 3–24.

[4] R. Keeney, H. Raiffa, Decisions with multiple objectives: Preferences and value tradeoffs, J. Wiley, New York, 1976.

[5] J. Dyer, MAUT — multiattribute utility theory, in: J. Figueira, S. Greco, M. Ehrogott (Eds.), Multiple Criteria Decision Analysis: State of the Art Surveys, Springer, 2005, pp. 265–292.

[6] J. Siskos, G. Wäscher, H.-M. Winkels, Outranking approaches versus MAUT in MCDM, European Journal of Operational Research 16 (2) (1984) 270–271.

[7] P. Korhonen, Interactive methods, in: J. Figueira, S. Greco, M. Ehrogott (Eds.), Multiple Criteria Decision Analysis: State of the Art Surveys, Springer, 2005, pp. 641–661.

[8] J. R. Figueira, S. Greco, B. Roy, R. Słowiński, An overview of ELECTRE methods and their recent extensions, Journal of Multi-Criteria Decision Analysis 20 (1-2) (2013) 61–85.

[9] J.-P. Brans, B. Mareschal, PROMETHEE methods, in: J. Figueira, S. Greco, M. Ehrogott (Eds.), Multiple Criteria Decision Analysis: State of the Art Surveys, Springer, 2005, pp. 163–186.

[10] M. Behzadian, R. B. Kazemzadeh, A. Albadvi, M. Aghdasi, PROMETHEE: A comprehensive literature review on methodologies and applications, European Journal of Operational Research 200 (1) (2010) 198–215.

[11] S. Corrente, J. R. Figueira, S. Greco, The SMAA-PROMETHEE method, European Journal of Operational Research 239 (2) (2014) 514–522.

[12] M. J. Beynon, P. Wells, The lean improvement of the chemical emissions of motor vehicles based on preference ranking: A PROMETHEE uncertainty analysis, Omega 36 (3) (2008) 384–394.

[13] K. Govindan, M. Kadziński, R. Sivakumar, Application of a novel PROMETHEE-based method for construction of a group compromise ranking to prioritization of green suppliers in food supply chain, OmegaTo Appear.

[14] O. Marinoni, A discussion on the computational limitations of outranking methods for land-use suitability assessment, International Journal of Geographical Information Science 20 (1) (2006) 69–87.

[15] J. M. Pereira, L. Duckstein, A multiple criteria decision-making approach to GIS-based land suitability evaluation, International Journal of Geographical Information Science 7 (5) (1993) 407–424.

[16] F. Joerin, M. Thériault, A. Musy, Using GIS and outranking multicriteria analysis for land-use suitability assessment, International Journal of Geographical information science 15 (2) (2001) 153–174.

[17] J. C. Aerts, G. B. Heuvelink, Using simulated annealing for resource allocation, International Journal of Geographical Information Science 16 (6) (2002) 571–587.

[18] S. Eppe, Y. De Smet, Approximating PROMETHEE II's net flow scores by piecewise linear value functions, European journal of operational research 233 (3) (2014) 651–659.

[19] B. Mareschal, Y. De Smet, P. Nemery, Rank reversal in the PROMETHEE II method: some new results, in: Industrial Engineering and Engineering Management, 2008. IEEM 2008. IEEE International Conference on, IEEE, 2008, pp. 959–963.