Mat-2.108, Special assignment in applied mathematics:
Implementing an optimization algorithm for the Smart-Swaps software

Juuso Nissinen 54667E

22nd May 2006
Contents

1 Introduction 3

2 Even Swaps with preference programming 5
  2.1 Bounds for feasible weights and value functions 6
  2.2 Practical dominance 8
  2.3 Solving the practically dominated alternatives with preference programming 10

3 Consistency of a system of inequalities 11
  3.1 The Fourier-Motzkin elimination method 11
  3.2 The implementation of Fourier-Motzkin elimination 13

4 Optimization methods 14
  4.1 Basic graph theory 16
  4.2 Floyd’s algorithm 17
  4.3 The graph theory based optimization algorithm 18
  4.4 The symmetrical issues of the practical dominances 20

5 Implementation and Results 22

6 Discussion and conclusions 25

References 27
1 Introduction

Today, people are faced with increasingly complex problems of making hard decisions. The decision maker (DM) has to typically choose between some decision alternatives, whose desirability to the DM is evaluated on some criteria. Multi-criteria decision analysis (MCDA) aims to help the DM in such situations, by providing clear, structured and consistent ways of making rational decisions.

Numerous MCDA methods and approaches have been developed to aid the DM in different situations, see for example Clemen (1996) or Belton and Stewart (2002). One such method is called Even Swaps (Hammond et al., 1998), in which the alternatives are evaluated by the value that they hold to the DM with respect to attributes, the criteria. The method is based on making value trade-offs, the even swaps, between the attributes.

In an even swap, the consequence of a selected alternative in one attribute is changed to a new value. This change is compensated with an equally valued change of opposite direction in another attribute of that same alternative. The created new alternative may be used instead of the initial one, as they are of equal value to the DM. It is important to note that by changing the outcomes of the alternatives, the DM creates new virtual alternatives that do not actually exist. However, if the process is carried out correctly, these alternatives are of equal preference to the DM and can be used interchangeably with the original ones.

The aim of these swaps is to make attributes irrelevant or alternatives dominated. If all the alternatives have the same consequence on some attribute, this attribute is irrelevant to the problem and can be eliminated. An alternative is dominated, if there exists an another alternative that is at least as good in every attribute and better in at least one. Consequently, such an alternative can be eliminated. Further, Hammond et al. (1998) define an alternative $A$ to be practically dominated, if there exists another alternative $B$ that is better than (or as good as) $A$ in almost all the attributes and only little worse in the other attributes, i.e. $A$ is dominated by $B$ for practical purposes. Similarly to the dominated alternatives, the practically dominated ones may also be eliminated. Even swaps are carried out until there only remains one alternative or one attribute, which naturally leads to choosing the best alternative.

Especially in larger problems (e.g. six or more alternatives/attributes), the process of identifying (practically) dominated alternatives or selecting the
best possible even swaps, can become difficult and confusing to the DM. To facilitate and support the Even Swaps method, a software called Smart-Swaps\textsuperscript{1} has been developed in the Systems Analysis Laboratory. Smart-Swaps provides support for the DM by identifying the dominated alternatives and by suggesting swaps that could make attributes irrelevant or alternatives dominated (Mustajoki and Hämäläinen, 2005). This is carried out with the preference programming model (Arbel, 1989; Salo and Hämäläinen, 1992, 1995).

Smart-Swaps is not intended only to the advanced users with the knowledge of decision analysis, but to everyone who faces demanding multicriteria decisions. The software is designed to be interactive, i.e. it should run in real time by accepting input and producing output to the user. However, as the calculations required by the preference programming are computationally demanding, the user has been obliged to wait as the computing takes place. One such computationally demanding subroutine is the identification of practically dominated alternatives. This task is carried out multiple times during the decision process and may take several seconds, especially with larger problems.

With the preference programming model, the practically dominated alternatives can be identified by solving multiple linear programming (LP) problems. Originally, Smart-Swaps solved these optimization problems with a freely distributed optimization package, the Qopt linear programming solver\textsuperscript{2}, which was not fast enough for a real time calculations. The aim of this study is to find and implement an optimization algorithm that finds the practically dominated alternatives in a shorter time. This would hopefully lead to a smoother and more interactive process for the DM.

The paper is organized as follows. In Section 2, the Even Swaps method, the Smart-Swaps software and the problem of solving the practically dominated alternatives are more closely described. Sections 3 and 4 introduce some possible approaches to solve the optimization problem. In Section 5, the implementation and results of the new algorithm are presented. Finally, Section 6 concludes the paper.

\textsuperscript{1}http://www.smart-swaps.hut.fi
\textsuperscript{2}http://www.isye.gatech.edu/~wcook/qsopt
2 Even Swaps with preference programming

In the Even Swaps method no general information about the relative importance of the attributes nor about the DM’s preferences concerning the attribute outcomes is needed. This enables also the people without mathematical background to use the method. However, in large problems, multiple difficulties are faced. It may be difficult to find even swaps that would either render attributes irrelevant or alternatives dominated. Also, if the alternatives are compared with respect to many attributes, it may be difficult for the DM to find the (practically) dominated alternatives. Further, information about the DM’s preferences acquired from the even swaps made is not utilized in eliciting the DM’s general preferences.

In Smart-Swaps, the preference programming model has been implemented to overcome these difficulties. By monitoring the swaps made by the DM, the software is able to suggest efficient even swaps that could otherwise remain unnoticed by the DM. Smart-Swaps also identifies the (practically) dominated alternatives and suggests these to the DM who may then eliminate them. The mathematical aspects of these procedures are presented in this Section. Special attention is given to defining the concept of practical dominance formally so that an efficient algorithm to identify these dominances can be developed.

The preference programming model is based on the multi-attribute value theory (MAVT) (Keeney and Raiffa, 1976). MAVT assumes that the outcomes of the decision alternatives in each criteria can be assigned a value according to the preferences of the DM. In reference to these values, weighted with the relative importance of each attribute to the DM, the best alternative can be selected.

In MAVT, the overall value of an alternative is composed from its value in respect to every attribute and of the relative weight of each of these attributes. If the alternatives are mutually preferentially independent (Keeney and Raiffa, 1976), additive value function may be used and the overall value of an alternative \( x \) is

\[
v(x) = \sum_{i=1}^{n} w_i v_i(x_i),
\]

where \( n \) is the number of attributes, \( x_i \) is the consequence of \( x \) with respect to attribute \( i \), \( v_i(x_i) \) is its rating on [0, 1] scale, and \( w_i \) is the weight of attribute \( i \). The ratings \( v_i(x_i) \) are assigned so that the best outcome in attribute \( x_i \) is
given the value of 1 and the worst gets 0. Ratings for different attribute levels in between are assigned based on the DM’s preferences, i.e. value function, for more details, see for example Clemen (1996). As a standard assumption, the weights are normalized, so that

$$\sum_{i=1}^{n} w_i = 1. \tag{2}$$

In the Even Swaps process, Preference programming can be used to model the incomplete information about the DM’s preferences, if the following two basic assumptions hold: (i) the DM’s value function is additive, and (ii) the DM is able to provide some information about her preferences. Neither of these assumptions is not considered very restrictive and they both are also commonly accepted.

As the DM is not obliged to state her general preferences in the Even Swaps method, the structure of the feasible set of weights \( w \in S \) is unknown. Also, the ratings, i.e. the shape of the DM’s value function, \( v_i(x_i) \) are unknown. With preference programming, however, we can define general bounds, within which these parameters are assumed to be. Then, the lower bound \( \underline{v}(x) \) for the overall value of an alternative \( x \) is obtained as the minimum of (1) in the region of feasible weights \( S \) and ratings, i.e.

$$\underline{v}(x) = \min_{w \in S} \sum_{i=1}^{n} w_i v_i(x_i), \tag{3}$$

where \( \underline{v}_i(x_i) \) is the lower bound for \( v_i(x_i) \) and \( w = [w_1, \ldots, w_n] \). The upper bound is calculated similarly.

The use of the Even Swaps method explicitly assumes that the there is no uncertainty concerning the consequences of the alternatives and that the DM knows all the consequences prior to making the decision.

2.1 Bounds for feasible weights and value functions

The use of preference programming to model the uncertainty of the DM requires setting initial general bounds for ratios of the attribute weights and for the ratings of the alternatives. Mustajoki and Hämäläinen (2005) assumed three kinds of bounds: (i) A general upper bound for the weight ratios, (ii) exponential value function bounds that define the lower and upper bounds.
for the ratings on each attribute, and (iii) general bounds for the rating differences, i.e., for the slope of the feasible value functions.

Each of these bounds can be initialized with only one general input parameter, so that the same bounds apply for each weight ratio and rating interval in the model. If we set a general upper bound \( r \geq 1, \ r \in \mathbb{R} \) for ratios of the attribute weights, we get constraints of the form

\[
\frac{w_i}{w_j} \leq r_{ij}, \ \forall i, j = 1, \ldots, n, \ i \neq j,
\]

so that \( r_{ij} = r, \ \forall i \neq j \). These constraints, together with the normalization constraint \( (2) \), define the feasible region of weights \( S \). The upper and lower bounds for the ratings \( v_i(x_i) \) are derived from the exponential value functions

\[
\bar{v}_i(x_i) = \frac{a^{x_i^N} - 1}{a - 1}, \ \text{and}
\]

\[
\underline{v}_i(x_i) = \frac{(\frac{1}{a})^{x_i^N} - 1}{(\frac{1}{a}) - 1},
\]

where \( x_i^N = (x_i - x_{i_{\text{min}}})/(x_{i_{\text{max}}} - x_{i_{\text{min}}}) \) is the value of \( x_i \) standardized onto range \([0, 1]\). Variables \( x_{i_{\text{max}}} \) and \( x_{i_{\text{min}}} \) are the maximum and minimum values of \( x_i \) in the initial set of alternatives. Naturally, \( v_i(x_{i_{\text{max}}}) = 1 \) and \( v_i(x_{i_{\text{min}}}) = 0 \). Input parameter \( a \in (0, 1) \) defines the curvature of the functions \((5)\) and \((6)\) so that the closer it is to 1, the closer the functions are to linear function. It is important to note that these functions only determine bounds for the value functions, not the actual functions.

The third parameter \( s \) is used to derive bound for rating differences, but is not discussed in this study, see Mustajoki and Hämäläinen (2005).

When the assumptions made above hold, the changes made in attribute consequences during an even swap do not depend on the consequence levels in other attributes. This allows us to use the information acquired about the equally preferred changes to enhance the quality of the approximation about the attribute weight ratios in \((4)\).

Assume that the DM makes an even swap, where the change in the consequence of the alternative \( x \) in the attribute \( i \) is compensated with a change in attribute \( j \). Together with the assumptions of equal valued even swaps and the additive model, we get
\begin{equation}
\begin{aligned}
w_i v_i(x_i) + w_j v_j(x_j) &= w_i v_i(x'_i) + w_j v_j(x'_j) \\
w_i (v_i(x'_i) - v_i(x_i)) &= w_j (v_j(x'_j) - v_j(x'_j)).
\end{aligned}
\end{equation}

As the ratings \(v_i\) and \(v_j\) were limited with the bounds derived from the exponential functions in equations (5) and (6), we can obtain a new upper bound for the weight ratio \(w_i/w_j\), as the maximum feasible ratio of the differences

\begin{equation}
\frac{w_i}{w_j} \leq \max \left( \frac{v_j(x_j) - v_j(x'_j)}{v_i(x'_i) - v_i(x_i)} \right).
\end{equation}

By substituting the the upper (\(\overline{v}_i\) and \(\overline{v}_j\)) and lower (\(\underline{v}_i\) and \(\underline{v}_j\)) bounds of the ratings and by assuming that \(v_j(x_j) - v_j(x'_j) > 0\) and \(v_i(x'_i) - v_i(x_i) > 0\), we get

\begin{equation}
\frac{w_i}{w_j} \leq \left( \frac{\overline{v}_j(x'_j) - \underline{v}_j(x'_j)}{\overline{v}_i(x'_i) - \overline{v}_i(x_i)} \right).
\end{equation}

Now we can change the value of \(r_{ij}\) in (4), which was initially set to the same value for every \(i,j\), correspondently. Further, we can readily derive equivalent equation to the (9), for the case were \(v_j(x_j) - v_j(x'_j) < 0\) and \(v_i(x'_i) - v_i(x_i) < 0\).

### 2.2 Practical dominance

As stated earlier, Hammond et al. (1998) defined practically dominated alternatives to be alternatives that are in practice not as good as some other alternative(s). However, as this definition is not mathematical in nature, we can define it more formally with the concept of the pairwise dominance (Salo and Hämäläinen, 1992; Weber, 1987). If the incomplete information concerning the preferences of the DM is modeled with preference programming and an alternative is pairwise dominated by another alternative, Mustajoki and Hämäläinen (2005) define it to be practically dominated.

In models with incomplete information, the pairwise dominance concept is used to analyze relations between the alternatives. Alternative \(x\) is dominated by alternative \(y\) in a pairwise sense, if the overall value of alternative \(y\) is at least as good as the value of \(x\) in the set of the feasible weights \(S\), i.e. if
\[
\min_{w \in S} \sum_{i=1}^{n} w_i [v_i(y_i) - \overline{v}_i(x_i)] \geq 0.
\]  

(10)

where \( S \) is the feasible region of the weights, \( \overline{v}_i(y_i) \) is the upper bound for \( v_i(y_i) \) and \( \underline{v}_i(x_i) \) is the lower bound for \( v_i(x_i) \) calculated from (3). To sum up the discussion in the previous Sections, the feasible set for weights \( S \) is subject to the constraints

\[
\frac{w_i}{w_j} \leq r, \ \forall i, j = 1, \ldots, n, \ i \neq j
\]

(11)

\[
w_i > 0, \ \forall i = 1, \ldots, n
\]

(12)

\[
\sum_{i=1}^{n} w_i = 1.
\]

(13)

We can readily see that this problem is of the form

\[
\min \ c^T x \geq 0
\]

s.t \( Ax \leq b \),

(14)

where \( A \) is a matrix of order \( m \times n \), \( b, x, c \in \mathbb{R}^n \) and \( a_{ij} \in \mathbb{R} \). Because both the target function and the constraints are linear, this is a LP problem.

The inequality in the target function of the problem (10) somewhat simplifies the solution process compared to the standard LP problem presented in (14). If the minimum is non-negative, we know that \( y \) dominates \( x \). However, if we can find a single point where \( \sum_{i=1}^{n} w_i [\underline{v}_i(y_i) - \overline{v}_i(x_i)] \leq 0 \), we do not need to solve the optimum value, as we know that it must be under zero and thus alternative \( y \) can not dominate \( x \) in a pairwise sense.

If an alternative is pairwise dominated, it means that there are no feasible weights for which it would be the best alternative (yielding the highest overall value). If we assume that the alternatives are mutually preferentially independent, it is clear that an alternative that is pairwise dominated is pairwise dominated also after any amount of swaps because the overall value of an alternative remains constant through the process.
2.3 Solving the practically dominated alternatives with preference programming

In practice, when the DM makes a swap with the Smart-Swaps, the system scans through all the alternatives seeking practically dominated alternatives. This requires that we solve the problem (10) for each possible pair of alternatives. If we have \( k \in \mathbb{N} \) alternatives, it is easy to show that we have to solve \( k(k - 1) \) problems. An interesting characteristic of these problems is that they all have the same feasible set \( S \) but different target functions.

In practice software does not automatically remove the alternatives found to be practically dominated. It simply suggests these to the DM, who in the end decides whether they should be eliminated. As Smart-Swaps is designed to be an interactive support tool, it is essential that the practically dominated alternatives are solved and presented to the DM as fast as possible once she has completed a new even swap. This requires that the algorithm used to solve the LP problems should be as efficient as possible.

Because the LP problems are extremely common in practice, there has been an extensive interest to solve them. Dantzig in 1947, was the first to formulate an algorithm, the simplex, to effectively solve LP problems. Published by Dantzig (1963), the simplex is still today the most used algorithm to solve moderate size problems. The Qsopt solver implemented originally in the Smart-Swaps to solve the practical dominances used also a variant of the simplex. As the Qsopt algorithm was not fast enough to allow a real time process for the DM, a new approach was needed in our case.

Rios Insua and French (1991) suggest that a LP problem of the type (14) can be tackled in two distinct analytical ways:

- By examining the consistency of a system of inequalities.
- Through mathematical programming formulations that find the minimum value of the target function.

In this paper, we will first examine the consistency approach and after that the optimization methods, for finding a new more efficient way to solve the LP problems (14) and thus identify the practical dominances.
3 Consistency of a system of inequalities

A system of linear inequalities, where \( A \in \mathbb{R}^{m \times n}, \ b, x \in \mathbb{R}^n \), and

\[
Ax \leq b,
\]

is said to be consistent, if and only if there exist \( x \in \mathbb{R}^n \) that satisfies it. When we convert the LP problem (14) to the form

\[
\begin{align*}
\min \ & \ z \\
\text{s.t.} \ & \ c^T x \leq z \\
& \ Ax \leq b,
\end{align*}
\]

we see that the problem of solving whether the minimum value of the target function in the equation (14) is lower than zero, is analogous to finding whether the system (16) is consistent when we put \( z = 0 \).

A common algorithm to determine the consistency of a system of inequalities is the Fourier-Motzkin elimination (FME). According to Melachrinoudis and Liu (2002), Fourier first proposed this method in 1826 and Motzkin reintroduced it in 1936. The approach used in this paper relies heavily to the version of Melachrinoudis and Liu (2002), where further clarifications and proofs may be found.

For problems, where each inequality(constraint) involves only two variables i.e the \( i \)th row of \( A \) has at maximum only two non-zero-elements, very efficient algorithms have been developed in literature. For example, Megiddo (1983) has developed an algorithm that runs in polynomial time \( O(mn^3 \log m) \), where \( m \) is the number of inequalities and \( n \) is the dimension of the problem. Unfortunately, we have the first inequality in (16) of the form \( c^T x \leq 0 \), in which \( c \) has more than two non-zero elements. Thus these methods were of no use in solving the problems of the form (16).

3.1 The Fourier-Motzkin elimination method

In FME, the amount of variables is reduced by the expense of having more inequalities. In every subsequent iteration, a single variable is eliminated until it is easy to determine the consistency of the system. In practice, this can result quickly in a very large number of inequalities. The needed operations,
however, are very basic (e.g. adding and dividing real numbers) and thus the method should perform reasonably well on small problems (Melachrinoudis and Liu, 2002).

We can write the system of inequalities (15) in an open form

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & \cdots & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \leq \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}. \quad (17)$$

The elimination of an (arbitrary) variable $x_n$ begins with sorting the constraining inequalities in (17) to three distinct sets $I_1, I_2$ and $I_3$ according to the coefficient $a_{in}$ of $x_n$.

$$a_{in} > 0 \quad \Rightarrow \quad i \in I_1$$
$$a_{in} < 0 \quad \Rightarrow \quad i \in I_2$$
$$a_{in} = 0 \quad \Rightarrow \quad i \in I_3 \quad (18)$$

We divide the equations, for which $i \in I_1, I_2$, with the absolute value of coefficient $|a_{in}|$. As a result we get the following inequalities, which are equivalent with those of the original problem

$$\frac{x_n}{\sum_{j=1}^{m-1} a_{ij} x_j - b_i} \leq \frac{b_i - \sum_{j=1}^{m-1} a_{ij} x_j}{a_{in}} \quad \text{if } i \in I_1$$
$$\frac{b_i - \sum_{j=1}^{m-1} a_{ij} x_j}{a_{in}} \leq x_n \quad \text{if } i \in I_2$$
$$\sum_{j=1}^{m-1} a_{ij} x_j \leq b_i \quad \text{if } i \in I_3. \quad (19)$$

The new system of inequalities, without the variable $x_n$, is formed by generating a new constraint from every possible pair of the inequalities in the two sets $I_1$ and $I_2$. Because we have $|a_{in}| \neq 0$ for $i \in I_1, I_2$, all the inequalities are well defined and we can eliminate the variable $x_n$. For if $C, D, x_n \in \mathbb{R}$, the following applies

$$\begin{cases} C \leq x_n \\ x_n \leq D \end{cases} \iff C \leq D. \quad (20)$$

The new set of inequalities is finally formed by adding the constraints that belong to the group $I_3$. These are the original constraints that did not include the eliminated variable $x_n$. It easy to see that the number of inequalities grows by
\[ \Delta m = (m_1 \ast m_2) - m_1 - m_2, \]

where \( m_i \) is the number of inequalities in \( I_i \). The described method produces a system of inequalities with one less variable that is consistent \textit{if and only if} the original system is consistent. For a proof, see Melachrinoudis and Liu (2002).

The procedure may be repeated as many times as it is necessary. Usually this means eliminating all variables so that we have only comparisons of real values. This remaining system, and hence the original system, is consistent if and only if all these comparisons are true. For example, the system

\[\begin{align*}
-5 & \leq 2 \\
0 & \leq -2
\end{align*}\]  

(21)

is inconsistent while the system

\[\begin{align*}
-5 & \leq 2 \\
0 & \leq 2
\end{align*}\]  

(22)

is consistent.

3.2 The implementation of Fourier-Motzkin elimination

To test the performance of FME method, a Matlab m-file was written. In addition to the algorithm of the previous Section, a row sorting algorithm was implemented to reduce the number of inequalities. After the algorithm has formed the new constraints, the coefficients \( \mathbf{a}_i \) of every possible pair of inequalities in equations (19) are compared and if they prove to be same, the less binding constraint is discarded. Once again, the implementation was adapted from Melachrinoudis and Liu (2002).

The Matlab tests were carried out with problems of the form (16). The parameters \( r_{ij} \) in the pairwise weight comparisons were given value \( r_{ij} = 5, \forall i, j \) and the coefficients of the target function were randomized.

The method appeared to be too slow for our purposes. Already the very preliminary tests showed that the amount of inequalities would rise very high. With six attributes the method produced tens of thousands inequalities in the final stage of the algorithm. As the amount of inequalities grows exponentially, with seven or more attributes this number becomes too high. The
algorithm goes through and organizes all these inequalities multiple times in a single iteration round, so it is clear that the amount of operations required and thus the computing time to eliminate one variable rises very quickly. When the results showed calculation times of several seconds for problems with six attributes, the FME and also other consistency based algorithms were ruled out as a solution.

4 Optimization methods

As the methods based on the consistency of a linear system appeared too slow, the attention was directed to the algorithms based on mathematical programming. Since Dantzig (1963), there has been extensive research that has improved the original simplex algorithm and proposed completely new ways to tackle such problems through the emergence of convex theory and interior point methods.

The focus of the research on LP problems has, however, been more on the larger problems and solvability issues. The problem we are facing with practical dominance is theoretically easy to solve even with the most basic simplex algorithm. The challenge is that we have to solve a considerable amount of such problems. There are efficient commercial optimization packages such as Xpress\textsuperscript{3} but the idea that Smart-Swaps should be free for anyone to use naturally ruled these out. The search for a new algorithm focused mainly on alternative methods that would complement the special structure of the problem presented in Section 2.

As the basic linear theory states, the extreme values for linear optimization problem are always found at the extreme points of the feasible set, called the polyhedron. Thus, if the set of extreme points is known, the optimum value for the problem is found by solving the optimal value in this set, which is considerably easier. For a complete presentation of the linear theory, see for example Bertsimas and Tsitsiklis (1997).

An approach based on this theory seemed very promising, as the feasible set of weights \( w \in S \) in (10) was the same for all the pairs of alternatives. In this way we have to calculate the feasible set only once and then find the minimum values for different target functions, matching every pair of alternatives.

It can readily be shown that the problem of practical dominance given in the

\footnote{\url{http://www.dashoptimization.com}}
equation (10), has $2^n - 2$ extreme points in the initial state, when all weight ratios are assumed to have the same value for $r_{ij}$. For weight $w_i$, $i \in S$, we had

$$\frac{w_i}{w_j} \leq r_{ij} \forall j \neq i$$  \hspace{1cm} (23)

$$\sum_{S} w_i = 1.$$  \hspace{1cm} (24)

Now at an extreme point each variable $w_i$ is either at the higher or lower level, as implicated in the above inequalities with the equality signs. With $n$ variables, we have $2^n$ possible extreme combinations. Naturally, all the variables cannot be at the upper or lower value at the same time, hence we have to subtract these two options. This gives us exactly $2^n - 2$ extreme points. This number applies only to the symmetrical case presented in equations (11)-(13). As the DM makes even swaps, the weight ratio constraints change and thus also the amount of extreme points changes. These changes should not, however, be dramatical, and the amount of extreme points, although it may grow, should stay in the same magnitude. As the DM should include only the relevant attributes to the Even Swaps process, we can expect that the amount of attributes seldom rises above fifteen. This means that the amount of extreme points of $S$ should stay in tens of thousands, an amount that modern computers can easily handle.

As we are only interested about the sign of the minimum, i.e. is there practical dominance, we do not need to know the exact value of the minimum. This means that if we find one extreme point where the target function gets negative value, we do not need to continue the calculations to find out if there exist a better minimum.

The search for a new algorithm was narrowed to the algorithms that find the extreme points of a polyhedron. To summarize the above discussion, the reasons were:

- The existent solution that was not fast enough, was based on simplex.
- All the pairwise comparisons of alternatives of the form (10) have the same feasible region.
- The exact value of the optimum is not needed, only its sign.
Matheiss and Rubin (1980) have identified and compared various algorithms for finding all the extreme points of a polyhedron. As a result, they found the algorithm of Manas and Nedoma (1968) as one of the fastest. Salo (1990), however, presented an algorithm that is considerably faster than that of Manas and Nedoma. This algorithm is based on graph theoretic framework but is suitable only for problems where all the constraints are pairwise comparison between the variables. However, this is exactly the structure of our problem and thus Salo’s algorithm was chosen as the most promising candidate for further study.

4.1 Basic graph theory

Let there be $n$ factors $u_1, \ldots, u_n$ that are being compared with respect to a given criterion. In our problem, these are the attribute weights. A directed graph $G = (V, E)$ is defined as the set of vertices $V = \{u_i|i = 1, \ldots, n\}$, each corresponding to an attribute weight. For each inequality $w_i/w_j \leq a_{ij}$ an edge is defined as $(u_j, u_i) \in E$, where $E$ is the set of edges. Every edge $(u_j, u_i) \in E$, is assigned with a cost $\omega(u_j, u_i) = a_{ij}$. These are the basic notations used in this discussion, for the other part, we follow common graph theoretic notation found for example in Jungnickel (2002).

**Example.** For the problem of practical dominance with three attributes, in which all the constraints are in their initial, i.e. symmetrical form, we have

$$\min \sum_{i=1}^{n} w_i[\omega_i(x_i) - \bar{\omega}_i(y_i)]$$

s.t

$$w_1/w_2 \leq r$$

$$w_2/w_1 \leq r$$

$$w_1/w_3 \leq r$$

$$w_3/w_1 \leq r$$

$$w_2/w_3 \leq r$$

$$w_3/w_2 \leq r$$

$$\sum_{i} w_i = 1$$

$$w_i > 0, \ \forall i.$$ (25)

A corresponding graph $G = (V, E)$ consists of the vertex set $V = \{u_1, u_2, u_3\}$ and the edge set $E = \{(u_1, u_2), (u_2, u_1), (u_1, u_3), (u_3, u_1), (u_2, u_3), (u_3, u_2)\}$. Each $u_i$ represent now a weight $w_i$ in the equation (25), while for every edge $(u_j, u_i), \ i \neq j$ a cost is assigned $\omega(u_j, u_i) = r$. Resulting graph is illustrated in Figure 1.
In a graph, a path is a trail from one vertex to another so that each vertex is visited only once. We denote the set of paths connecting \( u_{k_1} \) and \( u_{k_q} \), where 1 and \( q \) are the first and the \( q \):th vertices in the path, as \( P_{k_1 k_q} \). For a path \( p = u_{k_1}, \ldots, u_{k_q} \in P_{k_1 k_q} \), we define the intensity function as

\[
\lambda(p) = \omega(u_{k_1}, u_{k_2}) \omega(u_{k_2}, u_{k_3}) \ldots \omega(u_{k_{q-1}}, u_{k_q}).
\]  

(26)

If there exist a path \( P \), the minimum intensity \( \delta(u_i, u_j) \) is defined to be the minimum of \( \lambda \) in (26), i.e

\[
\delta(u_i, u_j) = \min_{p \in P_{i,j}} \lambda(p).
\]  

(27)

The problem of finding the minimum intensities is very close to the problem of finding the shortest path between two vertices in a given graph. Thus Salo (1990) used the Floyd's algorithm to calculate the minimum intensities.

### 4.2 Floyd’s algorithm

Floyd’s algorithm was originally developed to calculate the shortest path between all the pairs of vertices in a given graph \( G \). The algorithm, however, can be modified to calculate the minimum intensities, if certain special assumptions hold (see Salo, 1990). The algorithm is quite simple and has two phases.

In the first phase of the algorithm, we create a matrix \( \Delta \) having entries \( \delta_{ij} \) for the minimum intensity between vertices \( i \) and \( j \). These values are initially given the values of the edges joining the vertices \( i \) and \( j \). If these do not exist, the initial value of \( \delta_{ij} \) is set to be some large enough integer \( M \), so that it will
disappear as the algorithm advances. As was shown, in our network related to the problem of practical dominance, every pair of vertices is connected, so only entries $\delta_{ii}$ are assigned values of $M$.

In the second phase, the algorithm goes through every pair of vertices, so that it tries to connect the two with a path through all the other vertices. The minimum is saved on every iteration and subsequent intensities are compared to this value. As a result, we get the minimum intensities.

### 4.3 The graph theory based optimization algorithm

For graph $G = (V, E)$, we define a subgraph $G^+ = (V, E^+)$, where $E^+ \subset E$, so that $G^+$ is acyclic and connected, i.e., a spanning tree of $G$. By a standard result in graph theory $|E^+| = |V| - 1$. For more detailed analysis of spanning trees, consult e.g., Jungnickel (2002).

Given the graph $G^+$ with the above properties, we define the function $\gamma : V \times V \mapsto (0, \infty)$ as follows. For any adjacent pair of vertices $u_i, u_j$ in $G^+$, put

$$\gamma(u_i, u_j) = \begin{cases} \omega(u_j, u_i), & (u_j, u_i) \in E^+ \\ \frac{1}{\gamma(u_i, u_j)}, & (u_i, u_j) \in E^+ \end{cases}$$

(28)

Because $E^+$ is acyclic, only one of the edges $(u_i, u_j), (u_j, u_i) \in E^+$ so that $\gamma$ is well defined. We define $\gamma_i = \sum_{\{u_j \in V\}} \gamma(u_j, u_i)$ and put

$$\omega^i_j = \begin{cases} \frac{1}{\gamma_i}, & j = i \\ \frac{1}{\gamma_{i, u_j}}, & j \neq i, u_j \in V \\ 0, & u_j \notin V \end{cases}$$

(29)

When we define $w_i = (w^i_1, \ldots, w^i_n)$ for all $u_k, u_l \in V^+$, it can be shown that we have $w_k = w_l$. Hence we define $w^+ = w_i$, $\forall u_i \in G^+$.

Salo (1990) showed that for any spanning tree $G^+$ that has smaller or equal minimum intensities between any two pair of vertices than the original network, $w^+ \in \text{ext } S$. This means that the weights calculated from the $\gamma$-function (29) belong to the set of extreme points of the original LP problem (10). More formally, if $\Delta_G$ is the minimum intensities matrix for the graph $G$ and $\gamma_{ij}$ are calculated as in (28) for some spanning tree $G \in G^+$, then
\( w^+ \in \text{ext} \ S, \text{iff } \gamma(u_j, u_i) \leq \delta(u_i, u_j), \ \forall u_i, u_j \in V \ i \neq j \)  \hspace{1cm} (30)

For the example defined earlier, we have the minimum intensities matrix

\[
\Delta_G = \begin{bmatrix}
    r^2 & r & r \\
    r & r^2 & r \\
    r & r & r^2
\end{bmatrix}
\]  \hspace{1cm} (31)

For this example an (arbitrary) spanning tree \( G^+ \) is presented in Figure 2. We see that the graph is connected and acyclic, hence well defined. If we define matrix \( \Gamma \), so that \( \Gamma_{ij} = \gamma(u_i, u_j) \), where the values for \( \gamma(u_i, u_j) \) can be calculated from equation (28), we get

\[
\Gamma_G = \begin{bmatrix}
    1 & r & 1 \\
    \frac{1}{r} & 1 & \frac{1}{r} \\
    1 & r & 1
\end{bmatrix}
\]  \hspace{1cm} (32)

![Figure 2](image_url)

Figure 2: An example of a spanning tree that produces feasible \( w^+ \)

We see that \( \Gamma_{ji} \leq \Delta_{ij} \ \forall i, j \) so we have \( w^+ \in \text{ext} \ S \). The graph presented in Figure 2 represents the situation, where the weighting vector \( w^+ = [w_1 \ w_2 \ w_3] \) satisfies the condition \( w_2 = rw_1 = rw_3 \). As result, we can compute the corresponding extreme value from equation (29), which gives \( w^+ = [\frac{1}{2+r}w_1 \ \frac{r}{2+r}w_2 \ \frac{r}{2+r}w_3] \).

The spanning tree presented in Figure 3 does not produce a feasible weight \( w^+ \in S \), as from (28) we get

\[
\Gamma_G = \begin{bmatrix}
    1 & r & r^2 \\
    \frac{1}{r} & 1 & \frac{1}{r} \\
    \frac{1}{r^2} & r & 1
\end{bmatrix}
\]  \hspace{1cm} (33)
which does not satisfy the condition $\Gamma_{ji} \leq \Delta_{ij} \forall i, j$, where the matrix $\Delta$ is
given in (31).

\[ \begin{array}{c}
\text{u}_1 \\
\text{u}_2 \\
\text{u}_3
\end{array} \]

\[ \begin{array}{c}
r \\
r
\end{array} \]

Figure 3: An example of a spanning tree that does not produce a feasible $w^+$

Salo (1990) showed that if we form all feasible spanning trees $G^+ \in G$ and
calculate the corresponding $w^+ \in \text{ext}S$, we get all $w \in \text{ext}S$.

4.4 The symmetrical issues of the practical dominances

Due to the very symmetric nature of the problem (25), an unexpected
problem was faced. Although we get all the feasible extreme points $w \in \text{ext}S$ by
calculating the feasible spanning trees and the corresponding weights, the
algorithm is greatly hindered by the fact that the correspondence is not one
to one, but there can be multiple well defined spanning trees that correspond
to the same $w^+$.

For example, consider the two spanning trees given in Figure 4. They both
represent the situation, where $u_1 = u_3 = ru_2 = ru_4$. Both of the two trees
are feasible in the sense of (30), so that the algorithm goes through both of
these and finds the value for $w^+$ thus twice with double work.

\[ \begin{array}{c}
\text{u}_1 \\
\text{u}_2 \\
\text{u}_3 \\
\text{u}_4
\end{array} \]

Figure 4: An example of a situation, where two spanning trees correspond to
the same feasible $w^+$

20
With the problems that were solved in practice, the amount of duplicate points found was very large. For example, in a problem with seven alternatives, i.e. \( u \in \mathbb{R}^7 \), the original algorithm produced 33614 points, although there actually is \( 2^7 - 2 = 126 \) distinct points. Apparently the algorithm did much extra work in this way.

One should, however, note that as the Even Swaps process advances, the constraints of the form \( w_i/w_j \leq r_{ij} \) are updated with constraints \( w_i/w_j \leq s_{ij} \), where \( s_{ij} \) is obtained from the equation (8). This way, the problem quickly loses its symmetric properties, which it had in the beginning of the process, when all the attribute intervals were initialized similarly.

As a solution to the problem, the symmetric fractional constraints were all little perturbed from their original value of \( r \). This was done by adding a small positive real number \( \delta \in \mathbb{R} \) to each \( r \). The original constraint can be presented in a comparison matrix \( R \), by assigning each constraint of the form \( w_i/w_j \leq r \) an entry \( R_{ij} = r \). For the example presented in Figure 1, we get

\[
R = \begin{bmatrix}
1 & r & r \\
r & 1 & r \\
r & r & 1
\end{bmatrix}.
\]  

(34)

To create consistent results, a deterministic perturbation of the form

\[
R_{ij}^* = R_{ij} + \frac{0.000001}{i + j}
\]

(35)

was introduced. The use of a perturbation of this kind can be justified with the following points:

- The perturbation used is very small and should seldom affect the results.
- The feasible region is enlarged. This means that the program will not suggest any non-existent practically dominated alternatives, but instead some very close situations may become unnoticed. Additionally, as the program only suggest the practical dominances, the DM has to self decide whether to eliminate alternatives that are found (practically) dominated.
- The perturbation introduced is deterministic, so it will produce consistent results every time.
As a result, the computing times of the algorithm were decreased significantly due to the decreased amount of duplicate points found. We can see this readily from the figure 4. If we changed the cost of every arc with a little perturbation, the two spanning trees, should they still be feasible, would not produce anymore the same value for $w^+$. 

Various alternative perturbations were also tested, but none of them was found to produce smaller number of spanning trees than the presented one. In fact, the algorithm of Salo seemed to have an innate minimum bound for the number of spanning trees that could be achieved with perturbing the symmetrical constraints. Unfortunately this number was still considerably larger than the actual number of the extreme points, e.g. With ten variables 48620 compared to 1022.

With completely randomized perturbation, for instance, the algorithm produced exactly the same amount of spanning trees than with the presented deterministic one. The number of spanning trees could not be affected by increasing the relative perturbation from the symmetrical values. In all empirical tests, the results were distorted with much smaller perturbation than what was required to decrease the number of found spanning trees.

5 Implementation and Results

The algorithm described above, and fully presented in Salo (1990) was implemented with Java to run with the existing Smart-Swaps software package, also coded with Java.

The optimization routine was already in the original version designed to run independently in its own class, separated from the rest of the program. As the execution of the main program progresses to the point where the practical dominance are needed, the main class calls the optimization routine with the comparison matrix $R$ defined earlier. After the calculations, the $Q_{opt}$ optimization routine returns the dominance matrix of the size $n \times n$, where $n$ is the number of alternatives. In the dominance matrix $D$ entry $D_{ij}$ is marked one if alternative $x_i$ practically dominates $x_j$ and zero otherwise.

With the optimization this way separated from the rest of the program, it was easy to replace the existing algorithm with the new one. Also, because the data exchange between the classes is just few numbers in matrices, the interface of the new algorithm was easily matched to the existing one.

The graph based algorithm was compared to the existing $Q_{opt}$ solver in
Table 1: The computation times (ms) for the Qopt, perturbated and the original graph based algorithm in various scenarios. The number of found spanning trees (extreme points) is also presented.

<table>
<thead>
<tr>
<th></th>
<th>Qopt</th>
<th>Pert</th>
<th>Graph</th>
<th>Spanning trees</th>
<th>Pert/Graph(Correct)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>random</td>
<td>1156</td>
<td>60</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td></td>
<td>symmetric</td>
<td>1182</td>
<td>59</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>random</td>
<td>2362</td>
<td>111</td>
<td>127</td>
<td>6/6 (6)</td>
</tr>
<tr>
<td></td>
<td>symmetric</td>
<td>2344</td>
<td>144</td>
<td>1070</td>
<td>924/33614 (126)</td>
</tr>
<tr>
<td>10</td>
<td>random</td>
<td>4640</td>
<td>3960</td>
<td>3465</td>
<td>48620/Many (1022)</td>
</tr>
<tr>
<td></td>
<td>symmetric</td>
<td>4587</td>
<td>14283</td>
<td>No answer</td>
<td></td>
</tr>
</tbody>
</table>

Because of the symmetry issues in the early stages of the Even Swaps process, the algorithms were tested with two different comparison matrices: With randomized constraints, where $r_{ij}$ was uniformly distributed between $[1, 20]$ and with the symmetric scenario, where all the constraint were limited with the same value of $r_{ij} = 5, \forall i \neq j$. In the last column, the number of feasible spanning trees (extreme points) found with each graph based algorithm is displayed with the actual number of extreme points. The number of spanning trees were calculated only for symmetric scenario, as with the randomized constraints the number of feasible extreme points naturally varies.

As a result, the graph based algorithm was found to be significantly faster than the initial one with problems dealing with less than ten alternatives. Apparently the fixed cost of initializing the Qopt optimization routine is high, for even solving a small problem with tree alternatives takes approximately one second, whereas the graph based algorithms finds all the extreme points and evaluates the target value in each of these in less than 0.1 second.

The effects of the perturbation are also clearly seen from Table 1. With three alternatives there was no difference, but with seven and ten alternatives the perturbed algorithm worked clearly faster and was able to find the solution in the symmetric case with ten variables. It should be noted, that the differ-
ence between perturbed and graph algorithm in randomized scenario is most probably caused by the small sample size (10) and would probably disappear with a larger sample.

Without the perturbation, the graph based algorithm was not able to solve the problem with ten (or more) symmetrical constraints, and thus the Java program was terminated. This was probably due memory restrictions, as the number of spanning trees stored to the memory grew too large.

The reason for the faster performance of the perturbed algorithm can also be seen in the table with much smaller number of found extreme points. However, the number of found (duplicate) extreme points was still much larger than the actual number of \(2^n - 2\) for \(n\) variables. Other alternative perturbations were also tested, but none yielded better results. The reasons for this are unknown and are a subject for further study.

The Qsopt algorithm solved the randomized and symmetrical problems in almost equivalent times. With the graph based algorithm, as already stated, there was a considerable difference in computing times especially concerning the larger problems. This is interesting, because as the problem gets more complicated, i.e., loses its symmetrical aspects, the algorithm speeds up instead of slowing down, because it has to go through a large number of duplicate extreme points.

With the presented results it was decided to use the perturbed graph based algorithm with cases where there are less than ten variables. The original Qsopt package was still left to the software to solve problems with ten or more alternatives. It is noteworthy that although the graph based algorithm is faster in the randomized scenario with ten variables, its poor performance with the symmetric scenarios leads to its rejection with larger problems.

It should also be noted that the perturbated algorithm gave exactly the correct answers in all the test runs and additionally in the preliminary tests with the actual software.
6 Discussion and conclusions

The LP problems are generally considered easy to solve. The basic theory has existed for a decades and is well reviewed and accepted by the scientific community. However, as linear optimization problems rise again and again in engineering and economic applications in very specialized forms, there are still challenges and work to be done to achieve even better results. While the simplex with its many variants is the basic and reliable algorithm to solve the LP problems, there are situations, where alternative methods may provide far better results.

With the problem of practical dominance, this was exactly the case. The problem had such special characteristics that a specialized algorithm clearly dominated the established simplex. The implemented algorithm is based on graph theory, which provides powerful tools to deal with pairwise comparisons. In comparison to simplex and interior point methods, which find one feasible solutions and try to improve it, the solution implemented in this paper takes advantage of the similar structure of the problems that have to be solved in each iteration of the Even Swaps method.

The algorithm finds all the extreme points of the feasible polyhedron and then solves the optimization problem by finding out which of these points yields the highest result. This approach suits the problem of finding the practical dominance extremely well, as the feasible set in all the pairwise comparisons of the alternatives is the same while only minimized target function changes. It should be further noted that the absolute value of minimum was not needed, just the information whether it is positive.

Although no statistical testing was done, it is clear that the results with new algorithm were significantly better. For example with seven variables, the running time was only 1/20 of what it used to be. In the new implementation of Smart-Swaps, this enables near real-time computations which greatly smoothen the process for the DM.

Further, the implemented algorithm held the very peculiar feature that it ran faster when the symmetry of the problem was decreased. This was because of many duplicate answers found and was remedied by adding little deterministic perturbation to the symmetric structure. The added perturbation was very small in reference to the original data, so no distortion in results was found.

Despite the added perturbation, the algorithm still found many duplicate extreme points, and thus specially problems with many attributes, ten or
more, were not solved effectively. Luckily, the problems of this size are rare, as the DM should be able to narrow the decision to only concern really relevant alternatives. The new algorithm was implemented in such way that it solves the problem concerning nine or less variables whereas the original Qsopt-package still solves problem larger than that.

The added perturbation greatly lessened the negative effect of duplicate extreme points but did not remove it. Noteworthy, the amount of duplicate points could not be affected by increasing the relative perturbation, as the results became distorted long before the number of duplicate points began to decrease, as the perturbation was increased. The reasons for this are unknown and were not covered in this study. These could be interesting topics for further research.
References


