FACULTY OF SCIENCE AND TECHNOLOGY

MASTER THESIS

Study programme/specialization: Computational Engineering	Spring semester, 2022 Open
Author: Trym Seim Sandbakk	..... (Signature of author)
Course coordinator: Supervisor(s): Reidar B. Bratvold, Aojie Hong	
Thesis title: Bayesian Interactive Decision Support for Multi-Attribute Problems with Even Swaps	
Credits (ECTS): 30	
Keywords: Even Swaps Decision-making Decision analysis Decision support systems Bayesian updating/inference Probable dominance	Pages: 114 + appendix: 37 Stavanger, June 13, 2022

Bayesian interactive decision support for multi-attribute problems with even swaps

by

Trym Sandbakk

MSc Thesis

Presented to the Faculty of Science and Technology

University of Stavanger

2022

Abstract

Even swaps (ES) is a multi-criteria decision-making method introduced by Hammond et al. (1998) that makes it easier for decision makers (DMs) to make trade-offs between the decision criteria. The ES method can be further guided using decision support systems (DSSs) such that it becomes even easier to use the method.

This thesis intends to make a DSS to guide a DM through the ES method and assess how the preferences of the DM can be captured and updated using probabilistic dominance based on Bayesian updating.

Results show that the DSS implemented in this thesis can remove dominated alternatives through absolute dominance and practical dominance. Furthermore, the DM can make ES through the DSS. In addition, the DSS can suggest alternatives that are likely to be dominated, while also suggesting objectives that are close to having equal ranks such that these alternatives and objectives can be used for ES. Finally, changing the coordinate pair and lower and upper limits of the uniform distribution produces different results for the probable dominance such that it becomes easy to see which coordinate pair and limits for the uniform distribution produces the best results for dealing with the preferences of the DM.

Acknowledgments

First, I would like to acknowledge and express my sincere gratitude to my supervisor, Prof. Reidar B. Bratvold, whose suggestions, and discussions during this time have been exceptionally good help in producing this thesis. He has provided valuable supervision and guidance throughout this time, and I really appreciate all the help I received.

My appreciation also goes to my co-supervisor, Prof. Aojie Hong, for his support and guidance. My gratitude goes towards his help in understanding mathematical formulations and insightful discussions.

Last but not least, I would like to thank my family for their support during my Master studies and in life.

Trym Seim Sandbakk

Table of Contents

Abstract ..... **i**
Acknowledgments ..... **ii**
Table of Contents ..... **iii**
List of Figures ..... **v**
List of Tables ..... **vii**
Table of Abbreviations ..... **viii**
1 Introduction **1**
1.1 Problem motivation 1
1.2 Literature Review 2
1.3 Research goals 5
1.4 Thesis structure 5
2 Decision Support Systems **6**
3 Even swaps method **8**
3.1 Flaws in the ES method 19
3.2 Modified ES 21
3.3 Smart-Swaps 24
3.4 Bayesian Smart-Swaps 26
3.4.1 Bayesian interactive DS algorithms 28
3.4.2 Two-attribute example 34
3.4.3 Results from Bayesian Smart-Swaps 36
4 Even Swaps Program **39**
4.1 CYO 52
4.2 LCF 58
4.3 Even Swaps in ESP 66
4.3.1 Example of rows being removed during ES operations 82
4.4 Errors in the use of ESP 85
4.5 Probable dominance example 95
4.6 Discussion of results 102
4.7 Further improvements/additions to the ESP 108

5 Conclusion 109
References 111
Appendices 115
Appendix A 115
A1 Installation Guide 115
A2 User Manual 115
Appendix B 117
B1 ESP complete code 117
B2 Two-attribute example code 148
Appendix C 152

List of Figures

Figure 3.1: Sahid's CT from Hammond et al. (1998) 9
Figure 3.2: Sahid's RT from Hammond et al. (1998) 10
Figure 3.3: Making the even swap from Hammond et al. (1998) 12
Figure 3.4: Miller's CT from Hammond et al. (1998) 14
Figure 3.5: Miller's RT from Hammond et al. (1998) 15
Figure 3.6: Miller's ES 1 from Hammond et al. (1998) 16
Figure 3.7: Miller's ES 2 from Hammond et al. (1998) 17
Figure 3.8: Miller's ES 3 from Hammond et al. (1998) 18
Figure 3.9: Smart-Swaps user interface (Mustajoki and Hämäläinen, 2007) 24
Figure 3.10: Algorithm 1 (Bhattacharjya and Kephart, 2014) 29
Figure 3.11: Algorithm 2 (Bhattacharjya and Kephart, 2014) 31
Figure 3.12: Algorithm 3 (Bhattacharjya and Kephart, 2014) 33
Figure 3.13: Regions of absolute and practical dominance for two-attribute example when w1 ~ Uniform(0, 1) from Bhattacharjya and Kephart (2014) 34
Figure 3.14: Regions of absolute and practical dominance for two-attribute example when w1 ~ Uniform(0.4, 0.6) from Bhattacharjya and Kephart (2014) 35
Figure 3.15: Effect of learning upon the number and type of queries and events (Bhattacharjya and Kephart, 2014) 36
Figure 3.16: Normalized data from figure 3.15 (Bhattacharjya and Kephart, 2014) 37
Figure 3.17: Effect of M and N on the number and type of queries and events (Bhattacharjya and Kephart, 2014) 38
Figure 4.1: Welcome back message for user 40
Figure 4.2: Returning user quits program 40
Figure 4.3: CFD version 2 of the ESP 41
Figure 4.4: CFD version 1 of the ESP 43
Figure 4.5: Rules for using the ESP 44
Figure 4.6: New user elects to change username and password 44
Figure 4.7: Returning user returns to program 45
Figure 4.8: New user is happy with his username and password 45
Figure 4.9: File of usernames and passwords 46
Figure 4.10: Rules for doing ES for new users 47
Figure 4.11: Example CT and RT for new users 48
Figure 4.12: Equal attributes and dominance 49
Figure 4.13: Example of doing ES 50
Figure 4.14: User does not understand the ES process 51
Figure 4.15: User understands the ES process 51
Figure 4.16: Rules for creating CT in ESP 52
Figure 4.17: User enters number of objectives and alternatives 53
Figure 4.18: User enters names for alternatives 53
Figure 4.19: User creates CT 54

Figure 4.20: User is asked to rank string objective 55
Figure 4.21: User is asked for index position for string objective 56
Figure 4.22: User is asked whether rankings should be ascending or not 57
Figure 4.23: Created RT of the created CT 57
Figure 4.24: Example CT in Excel 58
Figure 4.25: Example RT in Excel 58
Figure 4.26: User is asked whether he would like to load a CT or an RT 59
Figure 4.27: User elected to load in an RT 59
Figure 4.28: User elected to load in a CT 60
Figure 4.29: Sahid's CT from Hammond et al. (1998) in the ESP 61
Figure 4.30: More string objectives to be ranked 62
Figure 4.31: Ranking the FY objective 62
Figure 4.32: Ranking the BSD objective 63
Figure 4.33: Index position for ranks 63
Figure 4.34: DF with only numbers 64
Figure 4.35: Finished ranking CT 65
Figure 4.36: Rules for doing ES in the ESP 66
Figure 4.37: Parkway dropped by practical dominance 67
Figure 4.38: Lombard dropped by practical dominance 67
Figure 4.39: Pierpoint dropped by practical dominance 67
Figure 4.40: Example of alternative being dropped by absolute dominance 68
Figure 4.41: DF before and after equality check 69
Figure 4.42: Likely dominance and almost equal ranks 70
Figure 4.43: DF with no almost equal ranks 71
Figure 4.44: No equal ranks for DF 71
Figure 4.45: User chooses alternative 72
Figure 4.46: User quits program instead of selecting an alternative 72
Figure 4.47: User chooses an objective 73
Figure 4.48: User quits program instead of selecting an objective 73
Figure 4.49: User chooses new value for selected objective and alternative 73
Figure 4.50: User selects objective to compensate on 74
Figure 4.51: DF with remaining objectives is displayed 75
Figure 4.52: User quits instead of selecting compensation value 76
Figure 4.53: User is asked for compensation value 76
Figure 4.54: DF after making an even swap 77
Figure 4.55: Alternative dropped after doing ES 78
Figure 4.56: User elects to not create a new file of the final alternative 79
Figure 4.57: User returns to CYO and LCF options 79
Figure 4.58: User elected to create a file of the final alternative 80
Figure 4.59: Final alternative in Excel 81
Figure 4.60: Example DF for removing rows during ES operations 82
Figure 4.61: Dominated alternatives have been removed 82
Figure 4.62: DF after doing ES 83

Figure 4.63: Alternative dropped after ES 83
Figure 4.64: Equal rows dropped 84
Figure 4.65: User does not enter correct option for CYO and LCF 85
Figure 4.66: User enters incorrect name of alternative 86
Figure 4.67: User enters name of objective not in RT 86
Figure 4.68: User enters swap value that is the same as the value already is 87
Figure 4.69: User enters value that is greater than the greatest rank in the RT 87
Figure 4.70: User enters value that is less than lowest value in RT 88
Figure 4.71: User enters non-int for swap value 89
Figure 4.72: User enters same name for compensated objective as swap objective 89
Figure 4.73: User enters the name of Excel file that does not exist or is not in same folder as program 90
Figure 4.74: User enters non-int for number of objectives 90
Figure 4.75: User enters non-int for number of alternatives 91
Figure 4.76: User does not type CT or RT 91
Figure 4.77: User enters non-int for index position 92
Figure 4.78: User enters non-int for rankings 92
Figure 4.79: User enters rank that is less than one 93
Figure 4.80: User enters incorrect type for index position 93
Figure 4.81: User enters incorrect password 94
Figure 4.82: DF with NaNs 94
Figure 4.83: User tries to do ES on DF with NaNs 95
Figure 4.84: Reproduced figure 3.13 of two-attribute example 99
Figure 4.85: Reproduced figure 3.14 of two-attribute example 100
Figure 4.86: Two-attribute example when $x1 = 0.4$ and $x2 = 0.8$ for $U(0, 1)$ 101
Figure 4.87: Two-attribute example when $x1 = 0.3$ and $x2 = 0.7$ for $U(0.2, 0.8)$ 102
Figure 4.88: CFD part 1 of how the ESP could work in a future version 105
Figure 4.89: CFD part 2 of how the ESP could work in a future version 106

List of Tables

Table 1: ES method application examples 3
Table 2: Average number of queries/events with learning turned on and turned off (Bhattacharjya and Kephart, 2014) 39
Table 3: Rules for creating a CT 52

Table of Abbreviations

Abbreviation	Phrase
AI	Artificial Intelligence
AVD	Annual Vacation Days
BS	Benefits
BSD	Business Skills Development
CA	Customer Access (%)
CFD	Control Flow Diagram
CIM	Commute in Minutes
CSV	Comma-Separated Values
CT	Consequences Table
CYO	Create Your Own
DF	Dataframe
DM	Decision Maker
DOS	Disk Operating System
DS	Decision Support
DSS	Decision Support System
ES	Even Swaps
ESP	Even swaps program
ET	Enjoyment
FY	Flexibility
GIS	Geographic Information System

GUI	Graphical User Interface
LCF	Load a Created File
MC	Monthly Cost ($)
ML	Machine learning
MS	Monthly Salary ($)
NaN	Not a Number
NPD	New Product Development
OS	Office Services
OSSF	Office Size (Sq. Feet)
RT	Ranking Table
SAVE	Scientific, Academic, Volunteer, and Educational
UASTAS	Unmanned Aerial Surveillance and Target Acquisition System

Note: To refrain from writing repeated he/she throughout the report, it is assumed that all mentions of he/she is male and therefore he will be used.

1 Introduction

People face many different decision-making challenges throughout their lives. These challenges can be related to work, school, personal life, or any other such challenges. Some of these decision-making situations are easy to deal with and do not require a lot of thought, e.g., buying ice cream on a sizzling summer day. You would look at the assorted flavors available and choose the one you like the most. On the other hand, some decision-making situations are complex, or the stakes involved are high, which requires a lot of thought, analysis, and possibly making the decision is a group effort with many stakeholders. For example, a corporation wanting to decide on whether they should start selling product A or product B. Product A may be better in some areas, while product B is better in others. Potentially, this situation could determine whether the corporation goes bankrupt or not. In such a situation, if one is better than the other on the chosen decision criteria, then there is no need for further analysis. The analysis must add value to the decision to be useful. If analysis is required, then decision support systems (DSSs) and decision-making methods like even swaps (ES) come into play (for more details on DSSs and ES, see chapters 2 and 3). DSSs can help guide the decision makers (DMs) through their decisions by the DM providing information about the decision to the DSS and receiving results back. With the DSS, the DM is equipped with information to base his decision, whether or not he uses the DSS. ES is a decision-making process where the DM makes value trade-offs based on his objectives and alternatives. ES was introduced in 1998 by Hammond et al. (1998).

1.1 Problem motivation

Making decisions can be complex and difficult based on several factors such as time constraints, limited information about the decision problem, and conflicting stakeholders. Therefore, a DSS

together with the decision-making method ES can help DMs make better decisions, i.e., decisions that are consistent with their values, alternatives, and information. This topic is important because the ES method is old, and it can take some time to learn using. It is important to update the ES method since it is an easy decision-making method to use. It is easy to use because it organizes the alternatives and the objectives in such a manner that makes it easy to look at the big picture of the decision problem. However, it was intended to be used with pen and paper. In today's modern world, this makes the ES method challenging and more time-consuming to use.

Making use of a DSS for the ES method can help DMs go through the ES method more efficiently by having the DSS make suggestions for the DM that he might not discover on his own. By applying Bayesian updating to the ES method, it can be further improved to make the DSS help catch and learn the preferences of the DM to propose swaps that are more aligned with the DM's values (Bhattacharjya and Kephart, 2014).

1.2 Literature Review

ES has been used in many different situations (see Table 1) since it was first introduced by Hammond et al. (1998). There have also been changes and improvements to the ES method such as Mustajoki and Hämäläinen (2004a, 2005, 2007), Dereli and Altun (2012; 2014), and Bhattacharjya and Kephart (2014). Mustajoki and Hämäläinen (2004a, 2005, 2007) have created a preference program called Smart-Swaps. This software makes it easier for DMs to make better decisions by the software asking about the preferences of the DM. More details about Smart-Swaps are in section 3.3. Dereli and Altun (2012; 2014) have modified ES to be used in multi-issue negotiation. More on modified ES is in section 3.2. Bhattacharjya and Kephart (2014)

introduces Bayesian interactive decision support for multi-attribute problems with ES. For more details on this work, see section 3.4.

From the previous work on and use of ES, ES is a decision-making method that has been used many times in terms of decision analysis and among experts in the field. Some works have addressed how it can be improved and made easier to use for non-expert DMs (see e.g., Mustajoki and Hämäläinen, 2007 and Bhattacharjya and Kephart, 2014), however, none of these publications have presented use cases by DMs and instead focus on the theoretical. This could be because it is difficult to find enough people to test the ES method on. However, some studies have done this such as Luo and Bor-Wen (2006), Luo (2008), and Lahtinen and Hämäläinen (2016). Luo and Bor-Wen (2006), and Luo (2008) used ES for enhancing the process of intuition for 11 nurses to see if their decision regarding quitting during the SARS outbreak in 2003 would be different compared to their intuition. Lahtinen and Hämäläinen (2016) tested the path dependence theory on 148 second year engineering students by using the Smart-Swaps software (Mustajoki and Hämäläinen, 2007).

Table 1: ES method application examples

ES method	Application	Source
Modified ES	Multi-issue negotiation	Dereli and Altun (2012; 2014)
Bayesian ES	DSS for multi-attribute problems	Bhattacharjya and Kephart (2014)
ES	Selection of an UASTAS	Hurley and Andrews (2003)
ES	Selection of a construction site	(1999)
ES	Choice analysis in SAVE community choice and impact model	Kask et al. (2011)

DSS-aided ES	Smart-Swaps DSS software	Mustajoki and Hämäläinen (2004a, 2005, 2007)
Modified ES	Customer co-creation in NPD through multi-issue negotiation	Altun et al. (2013)
ES	Path dependence in the ES method	Lahtinen and Hämäläinen (2016)
ES based on prospect theory	Emergency logistics plans	Qin et al. (2021)
ES	Strategy selection in a rural enterprise	Kajanus et al. (2001)
ES	Environmental policy choices	Gregory and Wellman (2001)
ES	Algorithm for multi-criteria sorting problems	Keser (2005)
ES	Enhancing the process of intuition	Luo and Bor-Wen (2006), and Luo (2008)
ES	E-negotiation systems	Wachowicz (2007)
ES	DS software to be used in negotiations	Wachowicz (2010)
ES	Gower plots and decision balls to visualize and rank alternatives	Li and Ma (2008; 2011)
ES	Security requirements engineering	Elahi and Yu (2009)
ES	Multi-criteria clinical DS	Dolan (2010)
ES	Analyzing requirement trade-offs by comparing alternatives	Elahi and Yu (2012)

ES	Decision chains for analyzing multi-criteria choice problems	Podinovski (2016)
ES	Decision-making in continuous choice models when using a GIS	Milutinovic et al. (2018)
ES	Mitigate biases by being simulated computationally	Lahtinen et al. (2020)

1.3 Research goals

The literature review has explained that ES can be used in many different fields for many different purposes. This shows that ES is a versatile decision-making method that is as easy to use. However, it is possible to guide the DM through the ES process with the use of a DSS such that it becomes even easier to use. The goals of this research are described below:

- Creating a DSS of the ES method such that it becomes easier for the DM to use the ES method
- Using a Bayesian approach to capture and learn the preferences of the DM through the DSS similar to Bhattacharjya and Kephart (2014)

1.4 Thesis structure

This thesis contains four chapters. Chapter 2 is about DSSs and how they can improve decision-making, in particular the ES method, while chapter 3 is about ES. This chapter goes in detail on how the ES method works, how it has been applied in different fields, and how it has been modified and improved by using DSSs. Furthermore, in chapter 4, the program is introduced and

discussed. The structure of the program is discussed along with how it compares to previous work, further additions and improvements that could be incorporated into the program, and limitations of the program. Finally, in chapter 5 there is a conclusion to the thesis.

2 Decision Support Systems

People want to make decisions that are consistent with their values, preferences, alternatives, and information. They want to follow the five rules of actional thought in making their decisions, as this is the only way to be logical and consistent in their decision-making. The five rules of actional thought are probability, order, equivalence, substitution, and choice. A DSS can be very useful for guiding the DM through the five rules of actional thought. In the context of this thesis, a DSS is particularly useful for multi-attribute values and preferences, as it can guide the DM through the ES procedure. Most of the time a DSS is a computer program that is used by organizations and businesses, but it can be useful for personal decision-making as well. An example of a DSS applying the ES method is the Smart-Swaps software by Mustajoki and Hämäläinen (2007).

According to Shim et al. (2002) DSSs started out in the late 1970s by the use of disk operating systems (DOS) and UNIX. Since DSSs have been used for such a long time, there has been much improvement in how DSSs work and how they are used. As explained by Shim et al. (2002), classic DSS tool design consists of three major areas:

1. Sophisticated database management capabilities
2. Powerful modeling functions
3. Powerful and simple user interface designs

These three areas define a good DSS. Although these areas were introduced a long time ago, they are still relevant today in designing a good DSS. However, in today's standards, DSSs can be further automized and improved using artificial intelligence (AI) and machine learning (ML). The next paragraph provides a literature review of DSSs to show how they have evolved over time, and that DSSs are tools that can be used in any field by anyone to make better decisions.

Belaid and Razmak (2013) go through how several types of multi-criteria DSSs within different fields work, and how these can be improved in the future. How research about model-driven DSS applications, underlying modeling techniques, delivery mechanisms, and DSS user interface have been implemented in different situations is discussed by Power and Sharda (2007). Turban et al. (2005) gives an overview of DSSs, how they has been applied in different scenarios, how DSSs can assist people in business, how different DSSs have been used in practice, and how DSSs have been affected by the internet. Furthermore, they discuss how DSSs can help DMs become better at decision-making. Razmak and Aouni (2014) discuss elementary notions of multi-criteria decision aid based DSS, evaluate some uses of multi-criteria DSS within some fields of application and organize multi-criteria DSS applications by fields. A DSS to be used for multi-criteria automatic clustering was created by Jabbari et al. (2022). Ensembles and single classifiers for credit scoring and prediction by means of a DSS was proposed by Luo (2020) to be compared with the performance of several other single classifiers. Bhattacharjya and Kephart (2014) use a DSS in their Bayesian Smart-Swaps to guide the DM through the ES process.

DSSs have a wide use for DMs as seen in this section. DSSs give DMs a tool to use to make better decisions. The program described in this thesis in chapter 4 is a type of DSS as it simplifies a decision-making method such that DMs that use this program can make better decisions.

3 Even swaps method

The ES process is a value trade-off decision-making method that makes it easier for DMs to make good decisions that are consistent with their values and preferences, alternatives, and information. It was introduced by Hammond et al. (1998, 1999) influenced by a letter from Benjamin Franklin to Joseph Priestly, where Benjamin Franklin proposed using trade-offs to simplify complexity. The full letter is in appendix C, reprinted from Hammond et al. (1998). According to Hammond et al. (1998) the ES process is a form of negotiating, as it forces the DM to consider the significance of one objective by comparison to another. What this means is that the DM must be willing to make compromises in terms of his objectives, such that he can make an even swap. Furthermore, Hammond et al. (1998) describe the ES process as a consistent tool for conducting trades and a comprehensible basis in which to conduct them. This establishes that ES supports the DMs in a structured and reliable way when they want to make complex decisions that have many alternatives with several objectives. Hammond et al. (1998) go on to show several examples of how the ES process works. They use consequences tables (CT) as illustrated in Figure 3.1. This table makes a grid of the alternatives and objectives, which makes it easy to compare them.

SAHID'S CONSEQUENCES TABLE

Objectives	Alternatives				
	Job A	Job B	Job C	Job D	Job E
MONTHLY SALARY ($)	2000	2400	1800	1900	2200
FLEXIBILITY	moderate	low	high	moderate	none
BUSINESS SKILLS DEVELOPMENT	computer	people management, computer	operations, computer	organization	time management, multitasking
ANNUAL VACATION DAYS	14	12	10	15	12
BENEFITS	health, dental, retirement	health, dental	health	health, retirement	health, dental
ENJOYMENT	great	good	good	great	boring

Figure 3.1: Sahid's CT from Hammond et al. (1998)

The first step in the ES process is to create a ranking table (RT) of the CT, as shown in Figure 3.2. The RT transposes the absolute values in the CT to relative values for each group and translates any qualitative attribute range to quantitative rankings, e.g., the highest salary is ranked as 1, and so on. The ranking goes from one to five, as there are five alternatives with one being the highest. For the quantitative objectives, the ranking is just whichever is highest, while for qualitative objectives, it requires more thought as it is based on what is more important to the DM (Hammond et al., 1998).

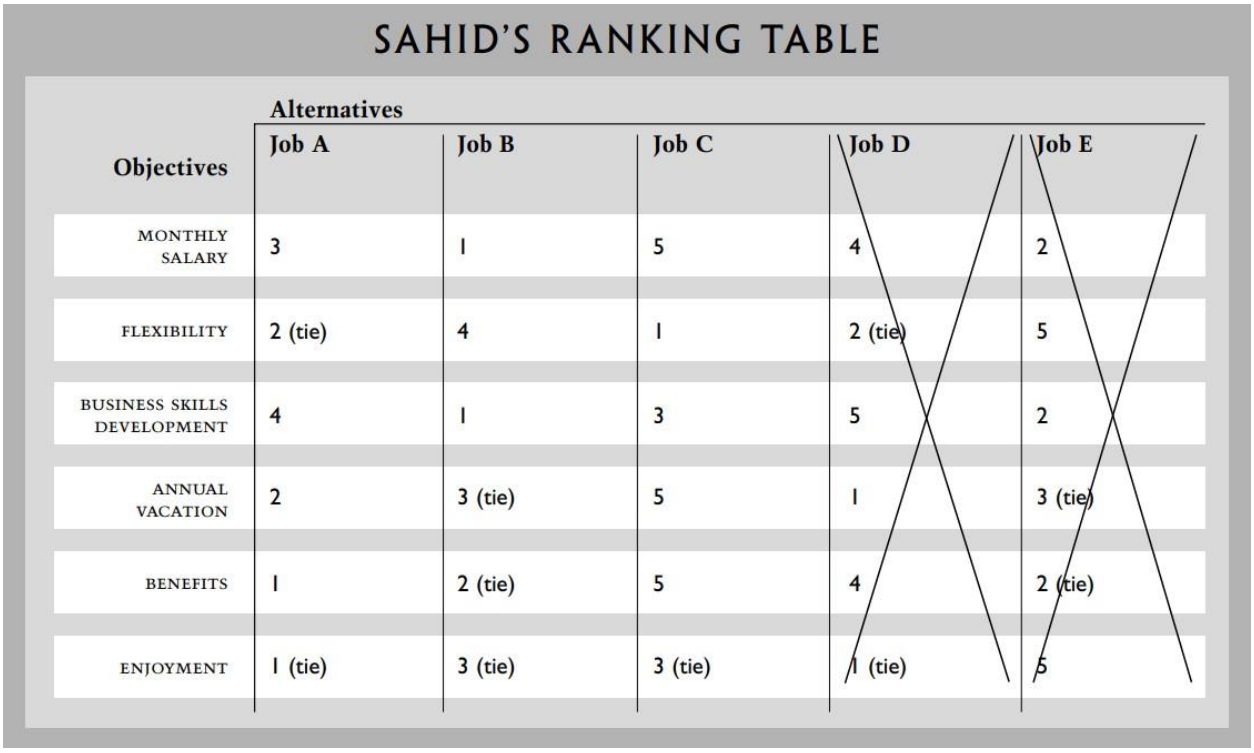


Figure 3.2: Sahid's RT from Hammond et al. (1998)

From the RT in Figure 3.2 Alternative D and Alternative E are crossed out, as explained by Hammond et al. (1998) since Alternative B absolutely dominates Alternative E by being better or the same on all the objectives. In the same sense, Alternative A practically dominates Alternative D, as Alternative A is better or the same on all, but one objective. Consequently, Alternative D and Alternative E are eliminated from consideration. The next step is to make ES by comparing the rest of the objectives and looking at what can be given up making the objectives equivalent. When objectives are equivalent, they are removed from the table. For example, by making the monthly salary (MS) objective equivalent, one would have to increase or decrease one of the other objectives such as flexibility (FY) to remove MS. The objectives you would make ES on must be for the same alternative such that you cannot make a swap for one objective for one alternative with another objective with another alternative. For example, to make a swap on MS

and alternative B, you would compensate on another objective for alternative B. Another thing to consider when dealing with the swaps is that you need to make consistent swaps such as making a swap where an objective can easily become equivalent. Furthermore, the swaps should be such that increasing one objective would require a decrease on another objective, and vice versa. From the RT in Figure 3.2, the enjoyment (ET) objective is equal for alternatives B and C such that only one swap is necessary on alternative A to make this alternative equivalent. Next, you compare the alternatives again to see if there is any dominance or practical dominance. The process is then repeated until a decision is made.

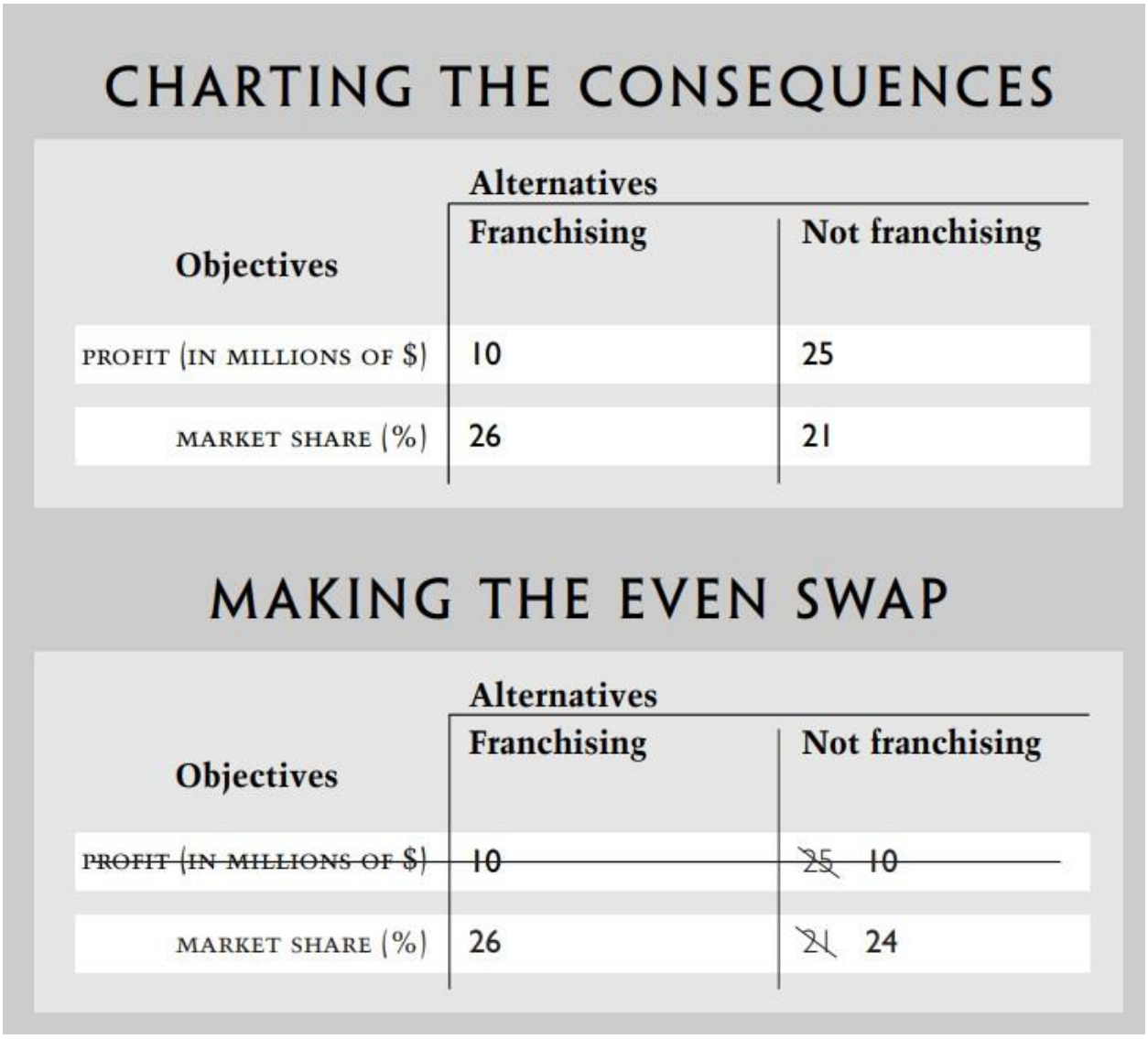


Figure 3.3: Making the even swap from Hammond et al. (1998)

Another example by Hammond et al. (1998) shows how the ES method works in practice. Looking at Figure 3.3, the "Profit" objective is eliminated from consideration since an even swap has been applied. The "Not franchising" alternative is adjusted to make the alternatives equivalent for the "Profit" objective by adjusting the "Market Share" objective. Considering this example, it is evident that ES is a substantial value trade-off decision-making process that makes it easier for DMs to make better decisions, as ES is easy to apply to any decision-making

context. Whether it is a big decision involving multiple stakeholders or a personal decision, ES can be applied to come to a final alternative. ES can be further guided using a DSS as is the case with Smart-Swaps (Mustajoki and Hämäläinen, 2005, 2007). For more details on Smart-Swaps see section 3.3. This does not mean that it does not take time to learn the ES method. It is recommended for inexperienced DMs who do not have much experience with DSSs and different decision-making methods to consider spending some time reading up on and learning the method before trying to apply it.

Like any other method, ES is not without its own flaws or inconsistencies (see section 3.1 for more on this). Once the DM has learned the ES method and is familiar with using it, the ES method can become a practical tool for making better decisions, either privately or in business. Other decision-making methods could be more aligned with the DM based on his experience in using decision-making methods and the type of decision-making problem he is facing. It is important for the DM to analyze the different methods he is considering to check how the different methods work.

A final example by Hammond et al. (1998) shows how the ES process works in making a decision in selecting the location of an office. The CT for this example is shown in Figure 3.4 below.

MILLER'S CONSEQUENCES TABLE

Objectives	Alternatives				
	Parkway	Lombard	Baranov	Montana	Pierpoint
COMMUTE IN MINUTES	45	25	20	25	30
CUSTOMER ACCESS (%)	50	80	70	85	75
OFFICE SERVICES	A	B	C	A	C
OFFICE SIZE (SQUARE FEET)	800	700	500	950	700
MONTHLY COST ($)	1850	1700	1500	1900	1750

Figure 3.4: Miller's CT from Hammond et al. (1998)

From this example, Figure 3.4 shows that Miller has five alternatives for office locations and five objectives that are important to consider for each alternative. Most of these objectives are quantitative, while the Office Services (OS) objective is qualitative. The next step is to create an RT as in the example of choosing a job in Sahid's example (see Figure 3.5).

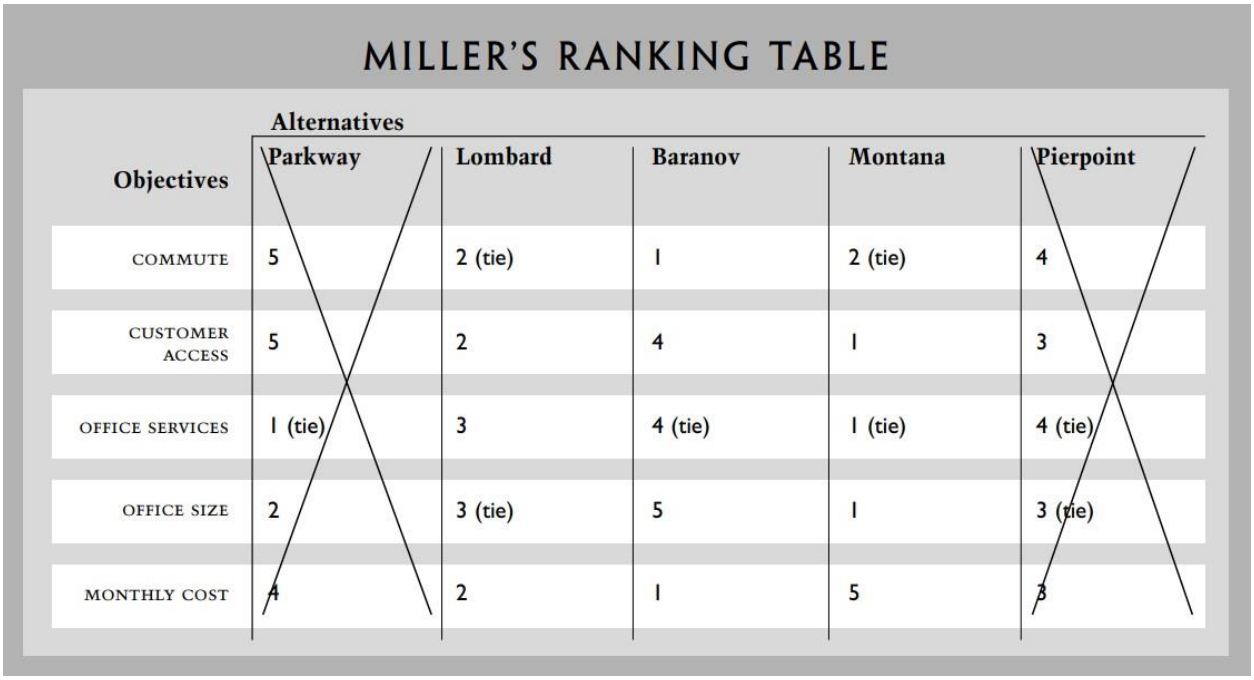
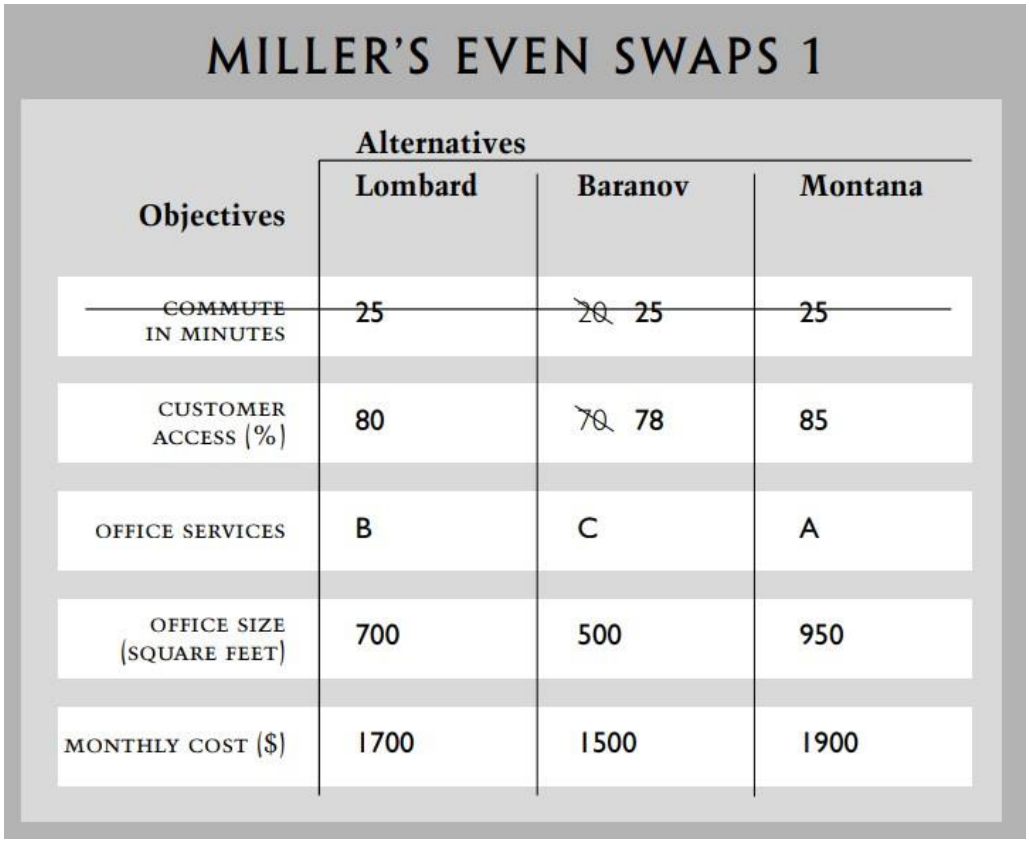


Figure 3.5: Miller's RT from Hammond et al. (1998)

Figure 3.5 shows the RT for Miller's office location decision problem. The next step is to look for absolute and practical dominance as explained in Sahid's example. Looking at the ranking table, "Lombard" absolutely dominates "Pierpoint" by being better or the same on all objectives. Furthermore, "Montana" practically dominates "Parkway" by being better or the same on all, but one objective. Now that the dominated alternatives have been removed, Miller is left with the "Lombard," "Baranov," and "Montana" alternatives as can be seen in Figure 3.6. From here, it is time to apply ES, as there are no more alternatives that dominate any other alternative.



MILLER'S EVEN SWAPS 1

Objectives	Alternatives		
	Lombard	Baranov	Montana
~~COMMUTE~~ IN MINUTES	25	~~20~~ 25	25
CUSTOMER ACCESS (%)	80	~~70~~ 78	85
OFFICE SERVICES	B	C	A
OFFICE SIZE (SQUARE FEET)	700	500	950
MONTHLY COST ($)	1700	1500	1900

Figure 3.6: Miller's ES 1 from Hammond et al. (1998)

Figure 3.6 displays the first even swap that is being applied. From the CT in Figure 3.6, the objective Commute in minutes (CIM) is similar for the remaining alternatives. Therefore, it makes sense to use ES here. The "Baranov" alternative is lower than the other alternatives for the CIM objective such that making this objective equivalent; it can be removed from consideration. This requires a change in one of the other objectives such as "Customer access (%)" (CA). The DM decides that a change from 70 to 78 for this objective is equivalent to making the CIM objective the same. After doing the even swap, the DM looks for more dominance to remove more alternatives. In this case, there is not any dominance, such that more ES are required to reach a decision.

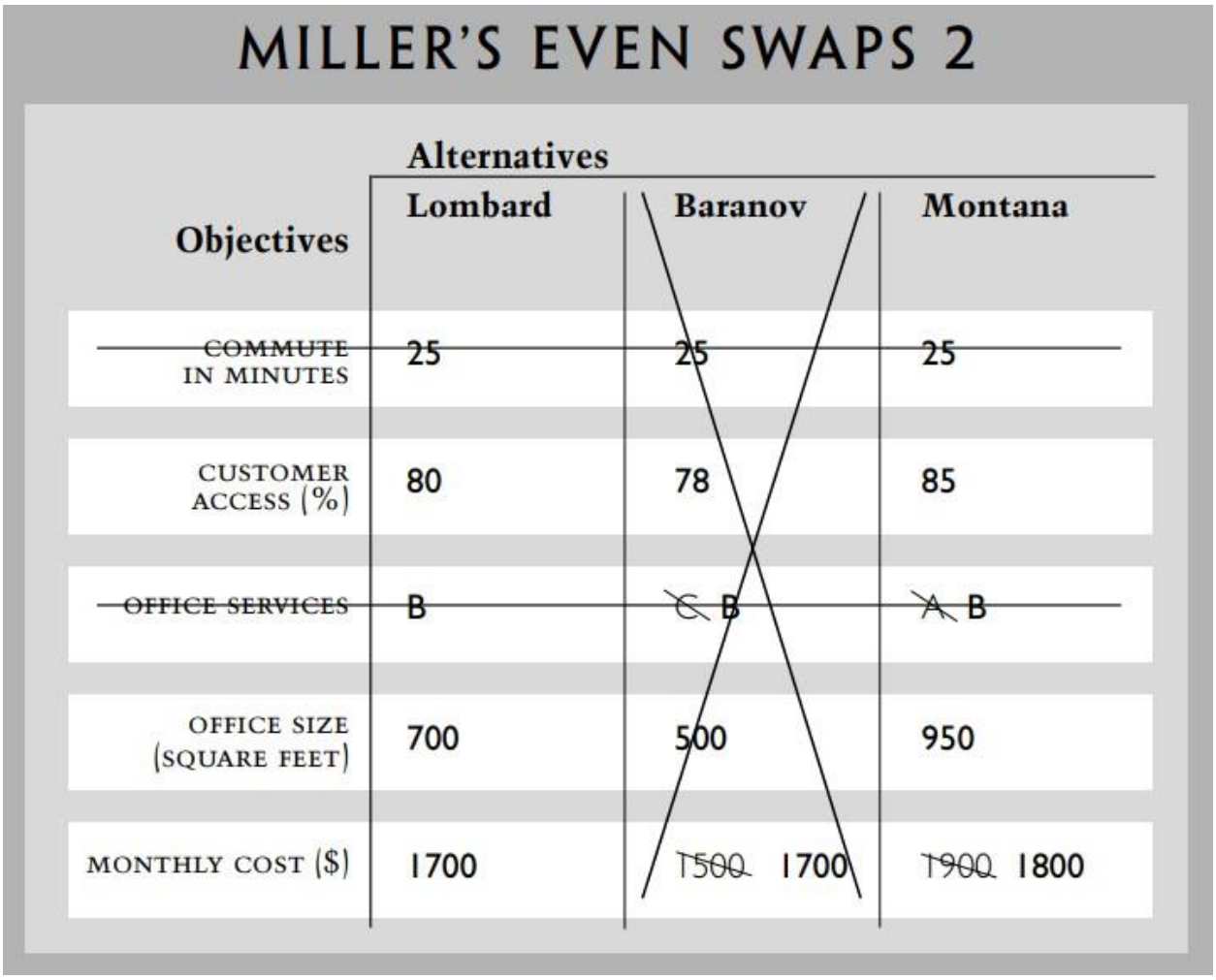


Figure 3.7: Miller's ES 2 from Hammond et al. (1998)

Figure 3.7 illustrates the next even swap. This swap considers the OS objective and the "Monthly cost ($)" (MC) objective. Both the "Baranov" and the "Montana" alternative needs to be considered for this, as the even swap is trying to make this equivalent on the OS objective. The same principle applies for this, such that the DM makes an adjustment on the MC objective. This makes the OS objective equivalent, and it can be removed. Once the even swap has been completed, dominance needs to be checked for. From the swap, "Lombard" dominates "Baranov" by being better or the same on all objectives (displaying absolute dominance) such

that “Baranov” can be removed. It is not possible to remove one of the remaining alternatives such that more ES are necessary.

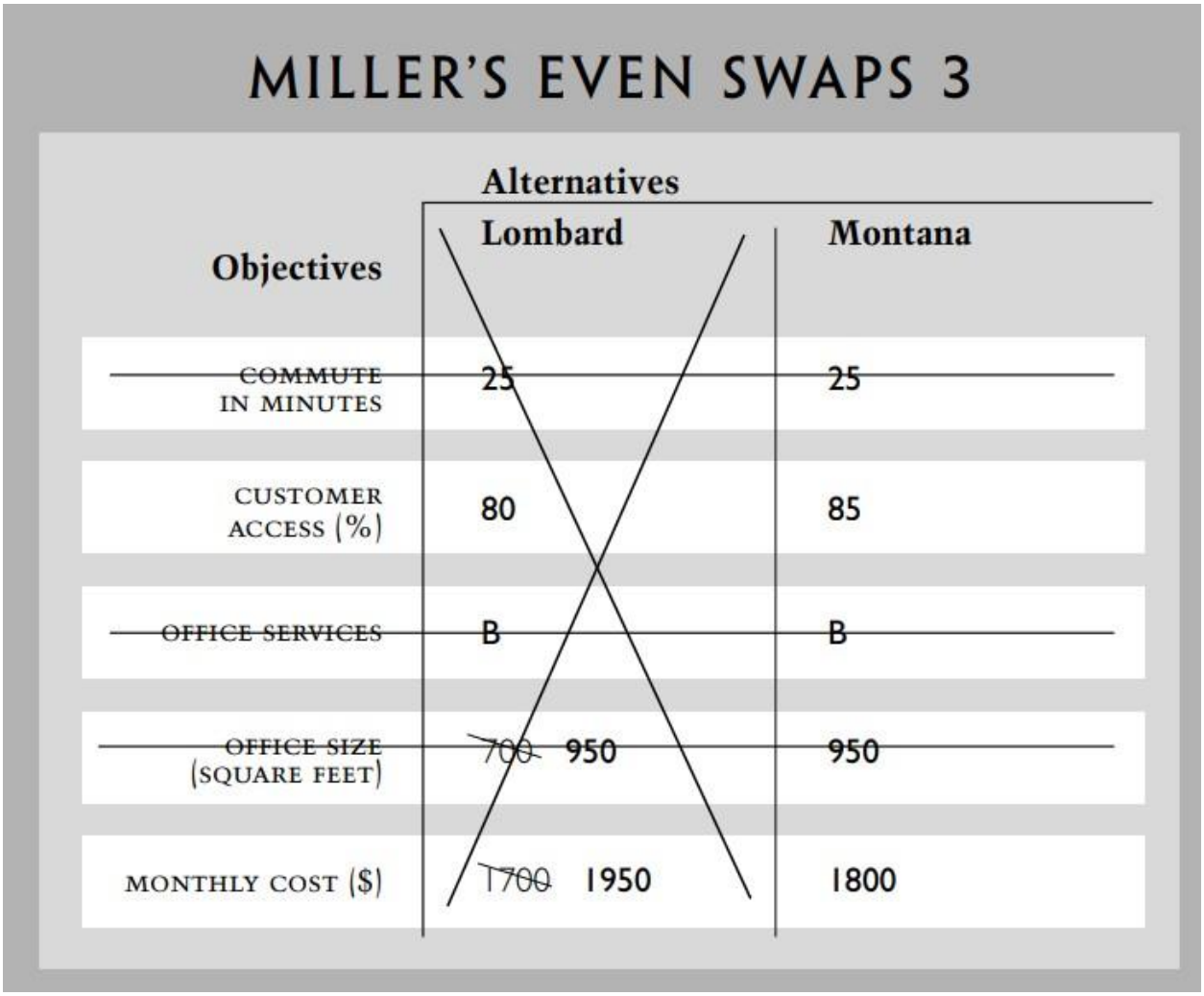


Figure 3.8: Miller's ES 3 from Hammond et al. (1998)

The final two alternatives, “Lombard” and “Montana” are shown in Figure 3.8. The final even swap concerns the “Office size (square feet)” (OSSF) and MC objectives. An even swap is being applied to make the OSSF objective equivalent by adjusting the value of the MC objective for the “Lombard” alternative. Thus, the only objectives that are left to consider are CA and MC. Looking at these objectives for the two final alternatives, “Montana” absolutely dominates

"Lombard" by being better on both objectives such that "Montana" is the clear, final choice. After going through the process of creating a CT and an RT, looking for dominance (absolute and practical), and doing the ES, this process is a valuable tool for DMs to make better decisions as it gives a clear, concise picture of the alternatives and objectives to consider.

3.1 Flaws in the ES method

As seen in the examples above, the ES method is easy to use and a valuable tool for making good decisions. However, there are some insufficiencies in the ES method. As explained by Li and Ma (2008):

1) Only the most preferred alternative is found. In an actual decision environment, the DM may also want to know the second or the third preferred alternatives.
2) Some trade-offs of criteria values, as specified by the DM, may not be consistent with each other. Current methods have no mechanism to check the consistency of these trade-offs.
3) The similarities among alternatives are not taken into account. Actually, the DM does not only want to know what the best option is but also the differences (or similarities) among alternatives.

There are two key causes for the listed insufficiencies in the ES method. The first cause is that according to the trade-off values defined by the DM, there is no way to present differences (or similarities) amid alternatives. A presentation of this nature among the alternatives with different trade-off values can help the DM to see the variances. The second cause is that according to the ES made by the DM, the insufficiencies may not rank alternatives consistently (Li and Ma, 2008).

An example of trade-offs of criteria values that are not consistent with each other is when a DM could be choosing a hotel for a vacation. The DM has his preferences for what type of hotel it should be and where it should be located. In this example, the DM has the choice of choosing between three hotels, namely hotel A, hotel B, and hotel C. The DM prefers hotel A to hotel B because of its luxury, hotel B to hotel C because of its location, and hotel C to hotel A because of its price. This means the DM does not have consistent trade-offs of criteria values, as the DM has different trade-offs for the different hotels.

Regarding the first inconsistency from the above list, it is a valid point that the ES method is unable to deliver in its purest form. By using a DSS for ES, this flaw can be addressed such that the DM may be able to know the other preferred alternatives. However, there might not always be other preferred alternatives, and thus this flaw might not be as bad as it seems. In addition, another solution to this inconsistency is that one could also eliminate the winner of the first round of the ES and then repeat the process once more to determine the second-best solution, and so on for as many top picks as required.

The second inconsistency can also be addressed by a DSS. A DSS can guide the DM through the ES process and can check the consistencies of the trade-offs. Nevertheless, this can be difficult for a DSS to do as well, as dealing with consistencies of the DM is challenging. There is not much help to get from a DSS if the DM decides to change his criteria values for trade-offs in the middle of the process. However, by using a DSS, this uncertainty would likely be decreased, and thus this inconsistency would be easier to deal with.

Finally, the third and final inconsistency about similarities (or differences) can be addressed with a DSS as well. A DSS can see what the similarities are between alternatives, especially if the alternatives are in an RT. Adding to this solution could be to decide rules for the most likely

candidate swaps prior to running the process. If such a rule set is established, it is then also important to recognize that swaps are not necessarily linear, so that trading for instance one object with two others does not always imply that one would trade two for four, but rather three for five, and so on.

Looking at these inconsistencies, a DSS can be the solution to address them to make it easier for the DM to use the ES process. The DSS might not be able to deliver solutions that are completely without these flaws, but at least it will be better for the DM. Furthermore, as seen from the insufficiencies in the ES method, the ES method is not without its flaws, just as any other method may be flawed. Therefore, it is important to always keep trying to improve and update the method. There are several ways of doing this. Bhattacharjya and Kephart (2014) have created a DSS based on ES that uses a Bayesian approach. Mustajoki and Hämäläinen (2007) have created a DSS that uses preference programming to guide the DM in the ES method. Dereli and Altun (2012) have modified the ES method to be used in multi-issue negotiation.

3.2 Modified ES

A closer look at modified ES by Dereli and Altun (2012; 2014) shows how the ES method can be adjusted for specific use cases such as multi-issue negotiation. The main steps of the modified ES method as explained by Dereli and Altun (2012):

1. **Problem initialization**

This step follows the initial step from Hammond et al. (1998), which is to create a CT (see Figure 3.1 and Figure 3.4) or a decision matrix where there are N alternatives to be assessed on M criteria.

2. **Determine “bargainable” issue**

The DM decides on which issue he would like or is able to bargain on. For example, the DM could be willing to decide on price. Then, all the ES in this method will be performed on the issue that the DM has decided on.

3. **Determine a bargaining range for bargainable issue**

The DM determines a bargaining range for the bargainable issue. This range considers the alternatives which remain in the CT.

4. **Eliminate "dominated" alternatives**

This follows from the dominance introduced by Hammond et al. (1998). However, it does not consider practical dominance such that all dominated alternatives are considered for ES.

5. **Eliminate "irrelevant" issues**

The irrelevant issues are the ones that have equal consequence for all alternative options such that the rows of irrelevant issues can be eliminated from the CT.

6. **Is there still any relevant unbargainable issue? If yes, then go step 7, Else, go to step 9**

There is only one thing to consider in this step, and that is whether there are any relevant unbargainable issues left or not. If there are, you go to step 7, otherwise, you go to step 9 and you are done.

7. **Determine alternatives and issue to perform "even swap" on them**

The DM goes through the CT to look for alternatives where one is better on some issues, while the other is better on other issues. Then, the issue, which has different consequence on these alternatives is selected to perform an even swap on them.

8. **Determine the required change and perform swap. Then, go to step 4**

The DM considers what change in one issue would compensate for a change in another issue. This follows from the even swap step described by Hammond et al. (1998).

9. **The most preferred alternatives found**

The DM continues to go through this process of eliminating alternatives and irrelevant issues until the only thing remaining is the bargainable issue. If there are any remaining bargainable issues, the DM must consider the bargainable range.

10. **Determine acceptable and unacceptable areas**

From step 9, only the bargainable issue(s) remains. The DM's preference is reflected by the change concerning the issue values. Area-limits for each lasting alternative in the CT can be detailed by the calculated difference. Calculated difference is used to find the limit of unacceptable area by subtracting the calculated difference from the issue value in the original CT. The limit of acceptable area is calculated by subtracting the predefined bargaining range from the unacceptable area limit. For an example of how modified ES is used, see Dereli and Altun (2012).

Now that the modified ES has been established, this method is not too different from the original ES method. The biggest difference between these methods is that there is a "bargaining" range for the modified ES and that acceptable and unacceptable areas are being applied to the remaining issue in which you could bargain on. Modified ES has been used with multi-issue negotiation, but it seems like a versatile method that would be easy to apply to other decision-making settings as well.

3.3 Smart-Swaps

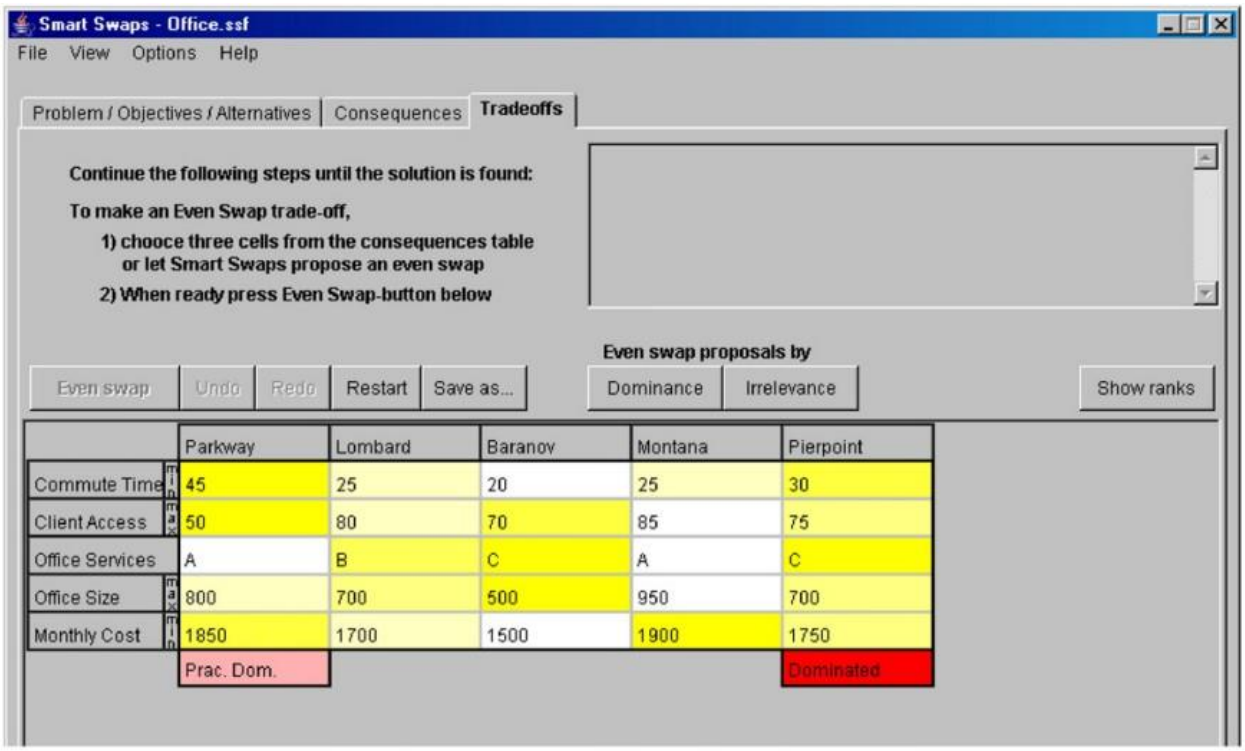


Figure 3.9: Smart-Swaps user interface (Mustajoki and Hämäläinen, 2007)

Mustajoki and Hämäläinen (2007) introduce a computer program they have called Smart-Swaps as a DSS using the ES process. Figure 3.9 shows the user interface of the Smart-Swaps program. Mustajoki and Hämäläinen (2007) have based their Smart-Swaps program on the method introduced by Hammond et al. (1998). According to Mustajoki and Hämäläinen (2007), the intention of Smart-Swaps is to deliver assistance for arranging the problem, prompting preferences, and investigating the outcomes so that the technical concerns can be taken care of by the computer, while the DM concentrates on the essential problem. This illustrates that Smart-Swaps is a versatile program enabling the DM in making better choices as shown by the program's ability to prompt preferences for the DM in such a way that it becomes simple for the DM to use this software.

The DM can set their own scale for the attributes such that it can be either discrete or continuous. This makes it possible for the DM's preference of the consequences to increase non-monotonically if they wish to do so (Mustajoki and Hämäläinen, 2007). This further demonstrates that the DM can make the scales according to what their decision problem is such that the Smart-Swaps program is a resourceful tool for DMs to use. There is a process log to keep track of all the ES the DM has made, and all the times an objective is eliminated or alternative dominated, and the state of the CT after each swap (Mustajoki and Hämäläinen, 2007). This process log makes it easy for DMs to keep track of all the action going on when they are going through the ES process.

Smart-Swaps lets the user know which alternatives can become dominated and which attributes can become irrelevant when the user selects cells to do the swap in (Mustajoki and Hämäläinen, 2007). This displays how easy it is for the DM to understand how the Smart-Swaps program works, enabling the DM to focus on the decision rather than how to go through the way of using the ES method. The software informs the DM about which way the attribute should be in (increased/decreased) and informs the DM if the swap is in the wrong direction such that the DM is able to redefine the swap (Mustajoki and Hämäläinen, 2007). The software suggests how a DM should deal with the swaps he is considering. This way of informing the DM if the swaps he makes are in the wrong direction makes it easier for the DM to understand how the ES process works, and how he is supposed to make swaps.

Furthermore, the model (Smart-Swaps) becomes more exact when the innovative preference data attained from the specified swaps is applied to originate innovative tighter constrictions in the model throughout the ES progression (Mustajoki and Hämäläinen, 2007). This indicates that the model updates its preferences and improves the decision-making progress for the DM as he goes

along throughout his process of deciding. Consequently, the DM should use the software several times and vary his input slightly each time to see how different the results become.

Comprehensive data for all the stages of the procedure is delivered by the procedural direction, which is carried out with support services in Smart-Swaps (Mustajoki and Hämäläinen, 2007). The DM can see what swaps he has made, which alternatives have been dropped by dominance, and how he went through the process for the problem. This makes Smart-Swaps a sophisticated tool for DMs to consider when going through the ES method. However, the Smart-Swaps software was developed from 2004-2007 such that this software is old and outdated. The authors of the software have not been maintaining it, and it requires Java to run in browsers (Mustajoki and Hämäläinen, 2004b). Most browsers today do not support Java or enabling it can be challenging. Therefore, this makes the Smart-Swaps software challenging to use in today's browsers. It is better to look at what the Smart-Swaps software aims to do and develop new software that works with modern solutions as is done in chapter 4 with the even swaps program (ESP).

3.4 Bayesian Smart-Swaps

Bhattacharjya and Kephart (2014) investigate a DSS that uses Bayesian methods to update the beliefs of the DM interactively. They start by introducing the ES process and what work has already been done with improving the process by guiding it with DSSs. As described by Bhattacharjya and Kephart (2014), a DM decides among N alternatives, where each alternative has M attributes. This follows from most decision-making contexts as it is common for a DM to deal with a number of alternatives and attributes.

Bhattacharjya and Kephart (2014) define an even swap that makes the consequences of two alternatives identical beside a characteristic as an equalizing even swap, i.e., a swap where the rank of one alternative is changed to be equal to the rank of another alternative. Being able to recognize when an equalizing even swap can happen makes it easier for DMs to make swaps, since the DMs can look for alternatives where the ranks are almost equal.

Bhattacharjya and Kephart (2014) compare absolute and probable dominance and conclude that their modest model of a two-attribute example shows that probable dominance is more useful in practice when a couple of alternatives are selected at random. This makes probable dominance a useful tool to capture and learn the preferences of a DM through the ES process. For more details on the two-attribute example see section 3.4.2.

Bhattacharjya and Kephart (2014) conclude that they have been able to show through experiments that one can effectively learn about the DM's preferences during a single session to guide them quickly to a final choice. Based on these experiments, the Bayesian interactive DSS works to help DMs make better decisions based on their preferences.

The system does not need to have a complete picture of the DM's preferences to find the optimal alternative for a particular decision. Therefore, it is beneficial to use such methods to reduce the elicitation burden and potential inaccuracies, as people are highly susceptible to cognitive biases (Bhattacharjya and Kephart, 2014). This illustrates that a Bayesian interactive DSS can help guide DMs to better decisions, but that they need to be aware of their cognitive biases to try to avoid being biased in making decisions. ES is particularly useful for DMs who either find it difficult to answer questions about their trade-offs in terms of weight ratios, or who need to view/consider the alternatives to construct their preferences (Bhattacharjya and Kephart, 2014). This makes ES a valuable tool for inexperienced DMs and experienced DMs and by using a DSS

to guide the DM through the process, ES can be even easier to use in all types of different decision-making contexts.

According to Bhattacharjya and Kephart (2014), before reaching the conference room, absolutely dominated alternatives would likely be dropped. However, when a DM or DMs have many different alternatives to consider with several objectives, it can be difficult to spot absolute dominance and therefore a DSS or program can be helpful for the DM(s) to use where the program removes absolutely dominated alternatives. This makes it easier for the DM(s) to focus his/their energy on making ES on the remaining alternatives.

3.4.1 Bayesian interactive DS algorithms

Bhattacharjya and Kephart (2014) suggested three different algorithms that use interactive decision support queries.

Algorithm 1 Practical Dominance Query

Input: N alternatives, threshold p_T, prior $p(\boldsymbol{w})$
Initialize $p_D^{max} = 0$
for each pair of vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ **do**
  Compute $p_{\boldsymbol{xy}}$ from equation (5)
  if $p_{\boldsymbol{xy}} \geq \max(p_T, p_D^{max})$ **then**
    Store pair $\boldsymbol{x}, \boldsymbol{y}$; $p_D^{max} = p_{\boldsymbol{xy}}$
  end if
end for
if $p_D^{max} \neq 0$ **then**
  Recommend potential practical dominance for $\boldsymbol{x}, \boldsymbol{y}$, inquiring whether $\boldsymbol{x} \succeq \boldsymbol{y}$
  Update $p(\boldsymbol{w})$ in accordance with DM's response, using equation (6)
else
  There is no candidate pair
end if

Figure 3.10: Algorithm 1 (Bhattacharjya and Kephart, 2014)

Algorithm 1 is the most important in terms of adding functionality to the ES method (Figure 3.10). The algorithm uses equation (*1*), which is equation 5 from Bhattacharjya and Kephart (2014) to calculate probable dominance or the probability that alternative ***x*** dominates ***y*** based on whether the DM prefers an alternative to another according to the system's beliefs (Bhattacharjya and Kephart, 2014). This algorithm can be used independently of the others in terms of just calculating probable dominance without considering the ES method. However, the other algorithms are necessary for the ES method with probable dominance.

$$p_{\boldsymbol{xy}} = \int_{\boldsymbol{w}} \left(\sum_{i=1}^{M} w_i [v_i(x_i) - v_i(y_i)] \geq 0 \right) p(\boldsymbol{w}) d\boldsymbol{w}$$

(1)

This probable dominance is compared with the max of an initialized value, p_D^{max} and a threshold value, p_T. The initialized value is updated if the probable dominance value is greater than the max of the initialized value and the threshold value. The threshold value is set by the DM before the start of the problem. This value determines how easy it is for real DMs to provide an answer for the queries (Bhattacharjya and Kephart, 2014). Then, if the initialized value is non-zero, the algorithm recommends potential practical dominance for the DM about the alternative pair $\boldsymbol{xy}$ inquiring whether the DM prefers $\boldsymbol{x}$ over $\boldsymbol{y}$. This is used to update the prior information the system has about the DM using equation (*2*), which is equation 6 from Bhattacharjya and Kephart (2014). Equation (*2*) result in an updated distribution over the DM's weights. This algorithm is used in accordance with the other algorithms to let the DM go through the ES process.

$$\sum_{i=1}^{M} w_i [v_i(x_i) - v_i(y_i)] \geq (\leq)\ 0$$

(2)

Algorithm 2 Even Swap Query

Input: N alternatives, swap response noise δ, prior $p(\boldsymbol{w})$
Set threshold $p_T = 0$ and find alternative pair $\boldsymbol{x}, \boldsymbol{y}$ from Algorithm 1
Initialize $p_S^{max} = 0$
for each pair of attributes i in $N(\boldsymbol{x}, \boldsymbol{y})$ and j in $D(\boldsymbol{x}, \boldsymbol{y})$ **do**
  Compute p_S from equation (8)
  if $p_S \geq p_S^{max}$ **then**
    Store pair i, j; $p_S^{max} = p_S$
  end if
end for
Recommend the swap $s(x_i \rightarrow y_i, x_j \rightarrow x'_j)$
if Response is x'_j **then**
  Update $p(\boldsymbol{w})$ with conditions from equation (9)
else if DM declares swap is infeasible **then**
  Recommend conjugate swap $s(x_j \rightarrow y_j, x_i \rightarrow x'_i)$
  Update $p(\boldsymbol{w})$ using equation (9), after swapping i and j
end if

Figure 3.11: Algorithm 2 (Bhattacharjya and Kephart, 2014)

Algorithm 2 recommends ES to the DM by using the information from algorithm 1 (Figure 3.11). A similar initialized value to the one in algorithm 1 is used in algorithm 2. Equation (*3*), which is equation 8 from Bhattacharjya and Kephart (2014) is used to calculate the probability that a swap will decrease the non-dominated set. Either the algorithm proposes ES or if the DM does not agree with this swap by declaring it infeasible, the algorithm proposes conjugate swaps.

According to Bhattacharjya and Kephart (2014), an even swap is defined as $s(x_i \to y_i, x_j \to x_j')$ and conjugate swaps are defined as $s(x_j \to y_j, x_i \to x_i')$ such that either the even swap or the conjugate swap must be feasible since either the change is on x_i or on x_j (for more details on this, see Bhattacharjya and Kephart, 2014). Algorithm 2 will always provide an even swap (or conjugate swap) that the DM can do based on his preferences.

$$p_S = P(x_j' \geq y_j) = P\left(v_j(x_j') \geq v_j(y_j)\right)$$

$$= \int_{\boldsymbol{w}} \left(\sum_{k=i,j} w_k [\, v_k(x_k) - v_k(y_k)] \geq 0 \right) p(\boldsymbol{w}) d\boldsymbol{w}$$

(3)

Finally, algorithm 3 incorporates algorithm 1 and 2 to remove dominated alternatives and recommend even swaps for the DM (see Figure 3.12). According to Bhattacharjya and Kephart (2014), their third algorithm ends when the ideal alternative is exposed after recognizing complete dominance and identical features, endorsing practical dominance when it is assured enough, and endorsing an equalizing even swap grounded on a dominance attentive heuristic. This demonstrates that the third algorithm can help guide DMs through the ES process in a way that makes it easier for the DMs to deal with their preferences, as they could be uncertain about their preferences. For more results from Bhattacharjya and Kephart (2014), see section 3.4.3.

Algorithm 3 Bayesian Smart Swaps

Input: N alternatives, threshold p_T, swap response noise δ, prior $p(\boldsymbol{w})$
while more than 1 solution and 1 active attribute remain in table **do**
  Remove absolutely dominated solutions, if any
  Mark any attributes with equal consequences across alternatives as inactive, if any
  Identify potential practical dominance using Algorithm 1
  if practical dominance detected **then**
    Recommend it and update $p(\boldsymbol{w})$ from response
  else
    Recommend an even swap using Algorithm 2 and update $p(\boldsymbol{w})$ from response
  end if
end while
if single attribute remains **then**
  Find the optimal alternative $\boldsymbol{x}$
end if
Return $\boldsymbol{x}$

Figure 3.12: Algorithm 3 (Bhattacharjya and Kephart, 2014)

3.4.2 Two-attribute example

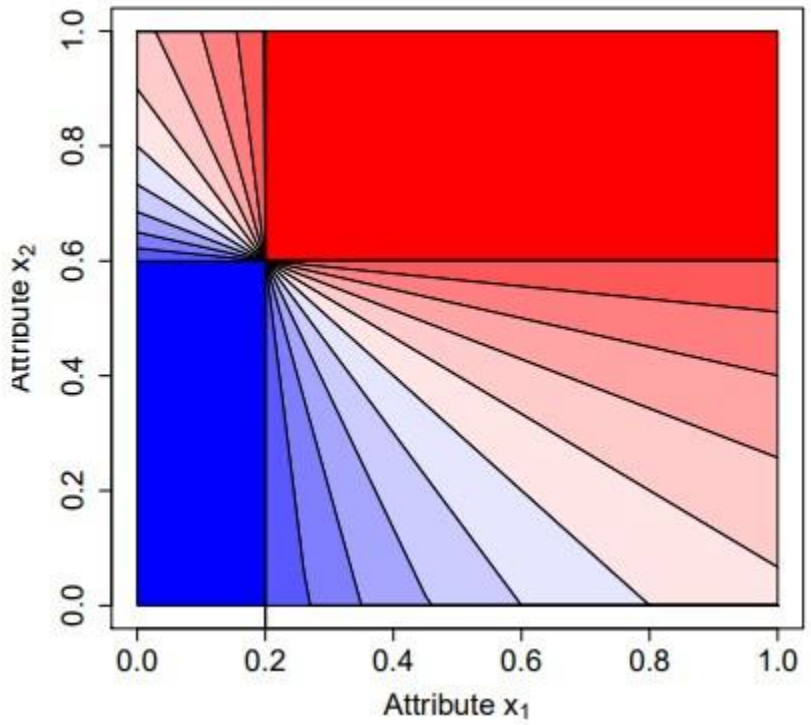


Figure 3.13: Regions of absolute and practical dominance for two-attribute example when w_1 ~ Uniform(0, 1) from Bhattacharjya and Kephart (2014)

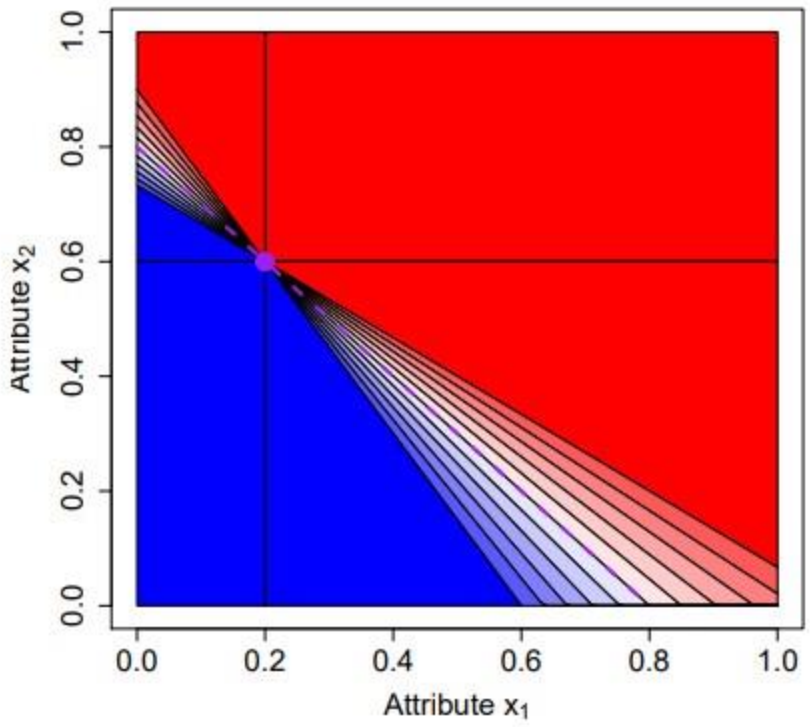


Figure 3.14: Regions of absolute and practical dominance for two-attribute example when w_1 ~ Uniform(0.4, 0.6) from Bhattacharjya and Kephart (2014)

The regions of absolute and practical dominance shown in Figure 3.13 and Figure 3.14 is with respect to a chosen alternative $\boldsymbol{x} = (0.2, 0.6)$. This implies that there are two attributes such that M = 2. The dark red color represents alternatives that absolutely dominate ***x***, while the dark blue color represents alternatives that ***x*** absolutely dominates. The lighter shaded region between the dark red and dark blue regions represents potential practical dominance determined by probable dominance and is the uncertainty between ***x*** dominating or being dominated. Figure 3.13 has a smaller light-shaded region compared to Figure 3.14. As stated by Bhattacharjya and Kephart (2014), this may be because the system has learned from the situation in Figure 3.13 to make a better prediction. Since the uncertainty region is reduced in Figure 3.14, it highlights the beliefs

about the system Bhattacharjya and Kephart (2014) made such that the system can be more confident in suggesting potential practical dominance to the DM.

3.4.3 Results from Bayesian Smart-Swaps

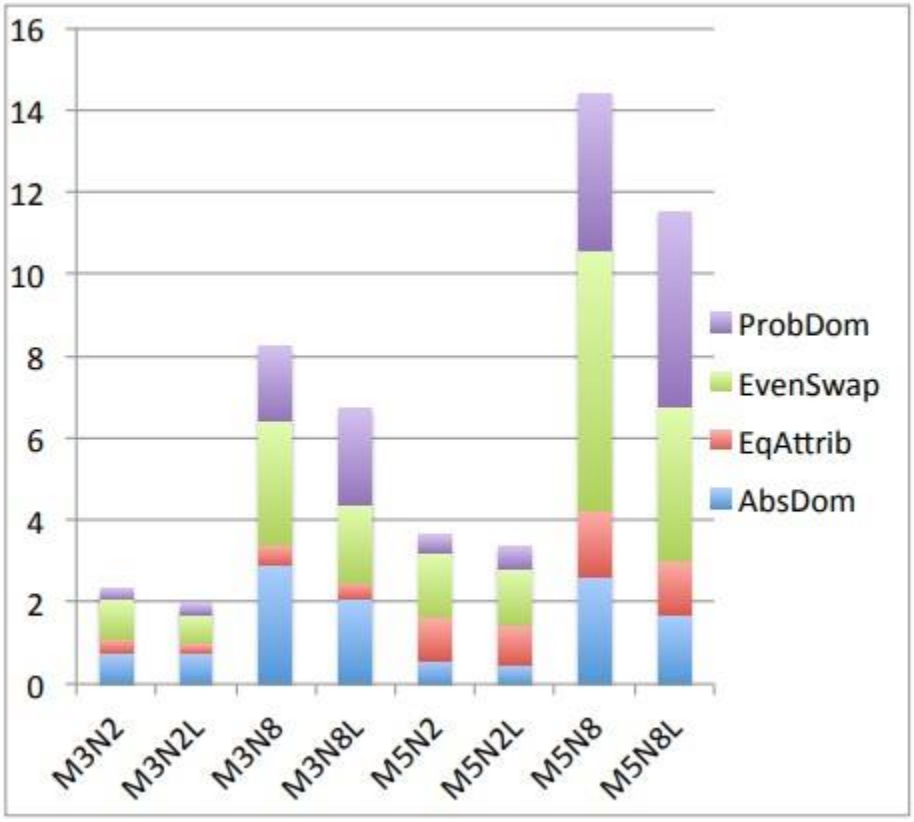


Figure 3.15: Effect of learning upon the number and type of queries and events (Bhattacharjya and Kephart, 2014)

Figure 3.15 shows the number and type of queries and events after using the effect of learning upon them (Bhattacharjya and Kephart, 2014). Learning (L) turned on and off for M = {3, 5} and N = {2, 8}. The average number of probable dominance and even swap queries per scenario, as well as the average number of absolute dominance and equal attribute events is recorded (Bhattacharjya and Kephart, 2014). From Figure 3.15, the absolute dominance (AbsDom) events

have been reduced when learning is turned on. The other events increase as M and N increase, which is to be expected since it is more likely that attributes can become equal, and that ES are required to get to a final alternative.

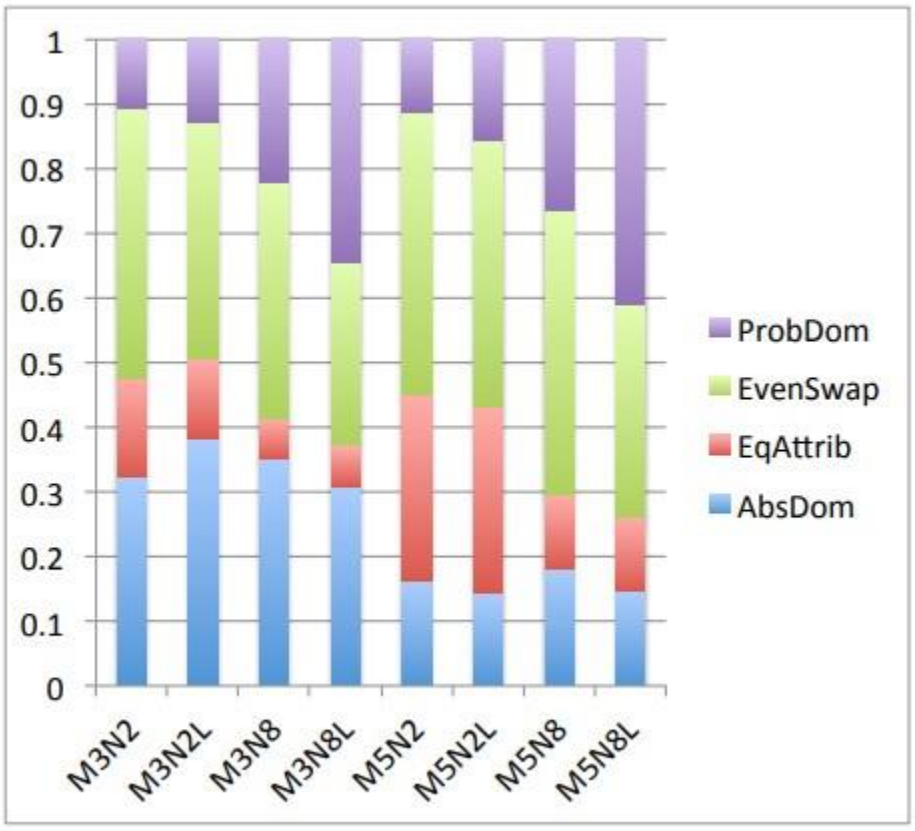


Figure 3.16: Normalized data from figure 3.15 (Bhattacharjya and Kephart, 2014)

Figure 3.16 displays the same data as in Figure 3.15 only normalized. This makes it easier to see the relevant contributions from each of the events and queries for this number of attributes and alternatives.

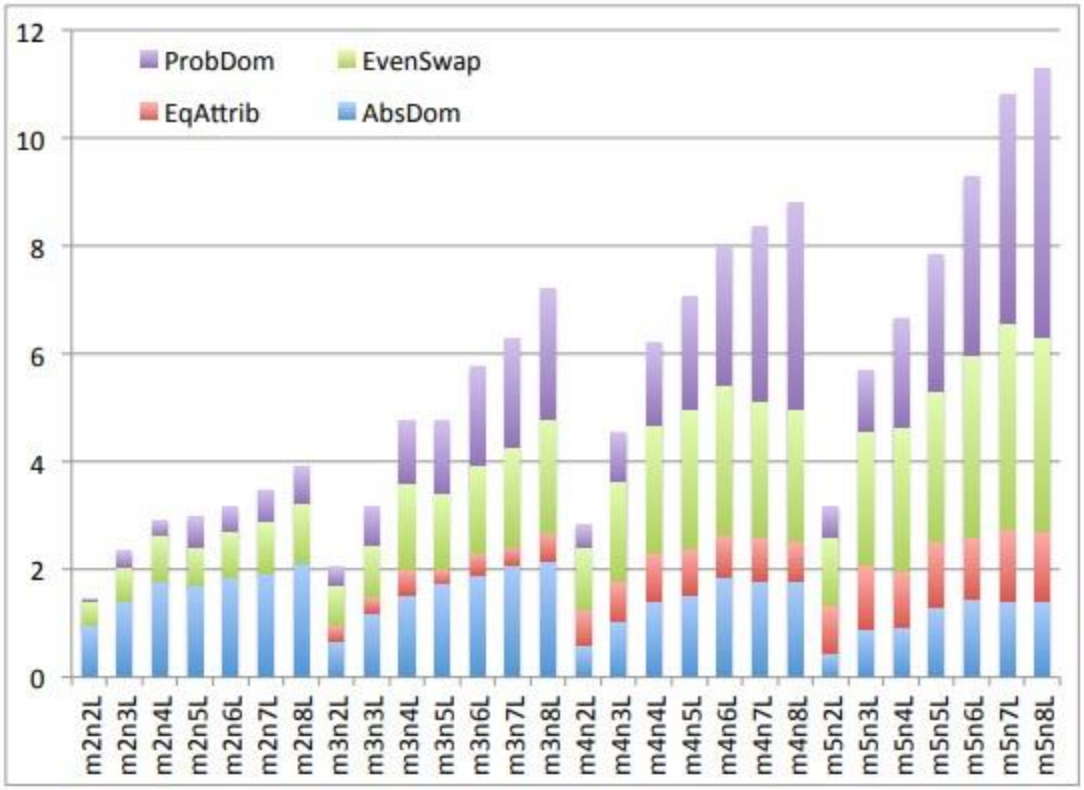


Figure 3.17: Effect of M and N on the number and type of queries and events (Bhattacharjya and Kephart, 2014)

Figure 3.17 exhibits similar data to Figure 3.15 just with M = {2, 3, 4, 5, 6, 7, 8} and N = {2, 3, 4, 5, 6, 7, 8}, and learning turned on. For M = 2, there are no equal attribute events. As M and N increase so do the number of events and queries. However, this is not the case for AbsDom events as they have a slight decrease as M and N increases. This makes sense since it is less likely that an alternative will absolutely dominate another when M and N increase. From Table 2, the average number of queries and events with and without learning is recorded. With learning turned on, the average number of queries and events decreased. However, when N is a small number, the difference in learning turned off and on is small. It is only when there are many alternatives that learning has an effect.

Table 2: Average number of queries/events with learning turned on and turned off (Bhattacharjya and Kephart, 2014)

Number of attributes M, solutions N	Average number of queries/events without learning	Average number of queries/events with learning
M, N = 3, 2	2.33 ± 0.11	1.96 ± 0.11
M, N = 3, 8	8.22 ± 0.34	6.7 ± 0.23
M, N = 5, 2	3.63 ± 0.27	3.35 ± 0.24
M, N = 5, 8	14.37 ± 0.57	11.51 ± 0.41

4 Even Swaps Program

The ESP is based on the work of Hammond et al. (1998, 1999) and inspired by the work of Mustajoki and Hämäläinen (2005, 2007). It is programmed in Python. Python is an easy programming language to learn, and it makes it easy to deal with the CT and the RT in the ES method, as these can be in the dataframe (DF) format, which makes it easy to read and manipulate. This program is supposed to be a tool for DMs to make it easier to go through the ES method. Furthermore, the ESP is also inspired by the work of Bhattacharjya and Kephart (2014). They have introduced Bayesian Smart-Swaps, which is a further improvement of the work of Mustajoki and Hämäläinen (2005, 2007). This Bayesian Smart-Swaps method uses the concept of probable dominance in the form of Bayesian updating to capture the probabilities of the preferences of the DM to propose swaps that are aligned with the preferences of the DM. A probable dominance example can be found in section 4.5, while more details on Bayesian Smart-Swaps are in section 3.4.

The first step in the ESP is to let the user/ DM sign in with his username and password if he is a returning user, otherwise he will select his username and password to start using the ESP. Once the user has signed in, he will be able to use one of two options, namely “Create Your Own (CYO) or “Load Created File” (LCF). In the CYO option, the user can create his own CT and RT, and in the LCF option, he can load a file he has created with either a CT or an RT in xlsx (Excel) format (more details on CYO and LCF in sections 4.1 and 4.2). However, if the user is new, he will be able to go through the ES process first to get familiar with how to use it.

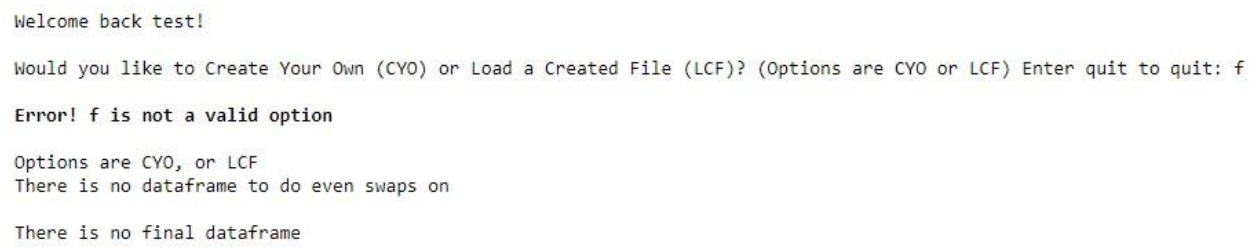


Figure 4.1: Welcome back message for user

Figure 4.1 shows the output of the ESP when the user is a returning user. As can be seen in the figure if the user types anything but the two options: CYO or LCF, he will get an error message. For more details on error messages in the ESP, see section 4.4. There is also a message describing what the user’s options are. There will be no final DF of the ES operations, as none of the available options were selected.

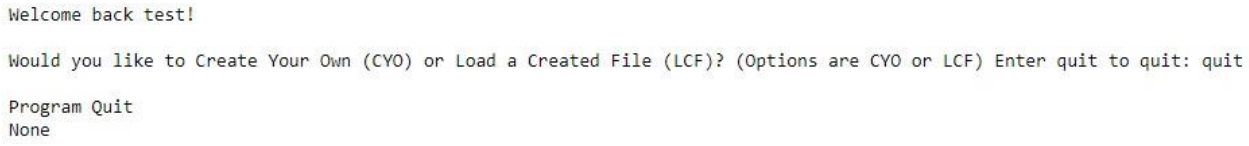


Figure 4.2: Returning user quits program

Once the user has selected one of the options, he will be able to continue with the program or as shown in Figure 4.2, he may type quit to exit the program.

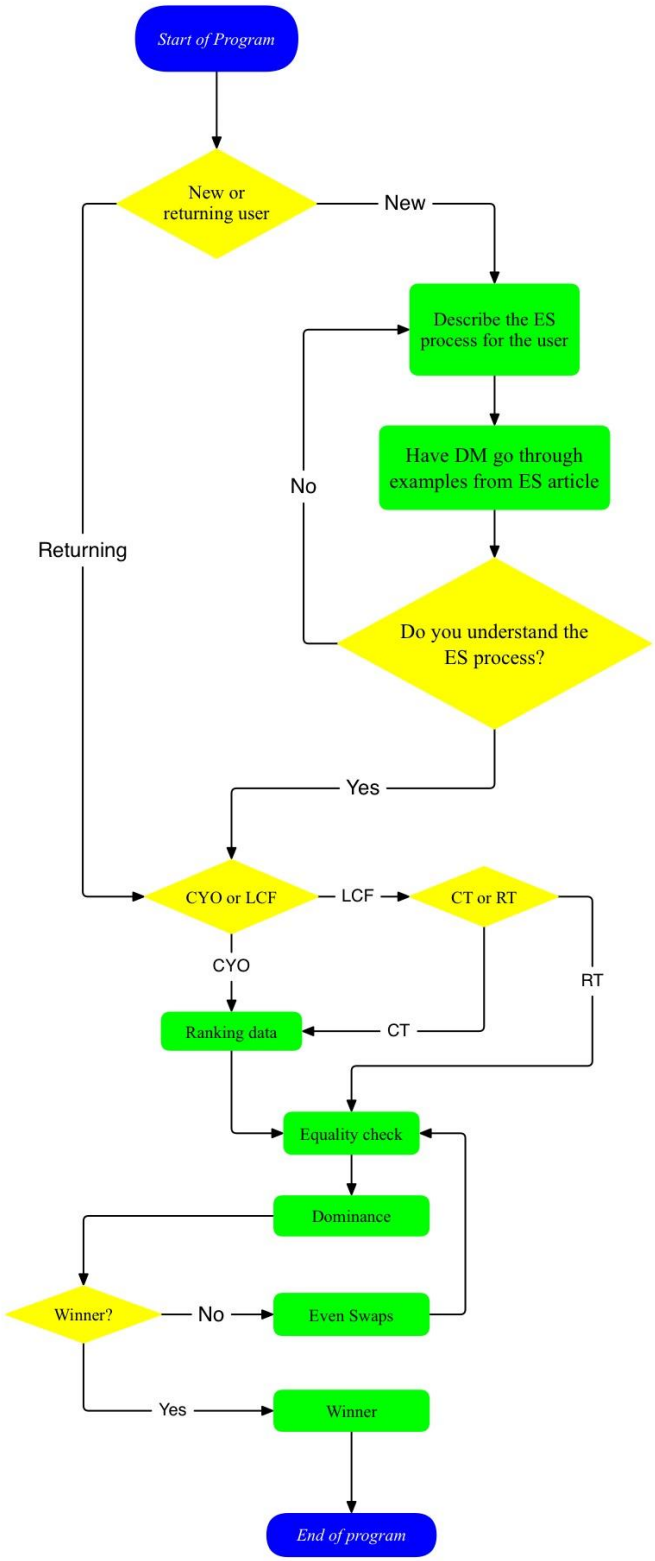


Figure 4.3: CFD version 2 of the ESP

The control flow diagram (CFD) is shown in Figure 4.3. The blue rectangles with the rounded corners represent the start and end of the program. The yellow diamonds indicate that the program continues in one direction or the other. Finally, the green rectangles specify what the program is doing. In an earlier version of the program (see Figure 4.4), the flow was such that once the user has decided on whether he would like to use the sample data to test the program or use user-defined data, he would enter the ES part of the program. Once in the ES part, the user would conduct ES until a final alternative (or winner) emerged, while the program checked for dominance and equality to remove alternatives and objectives.

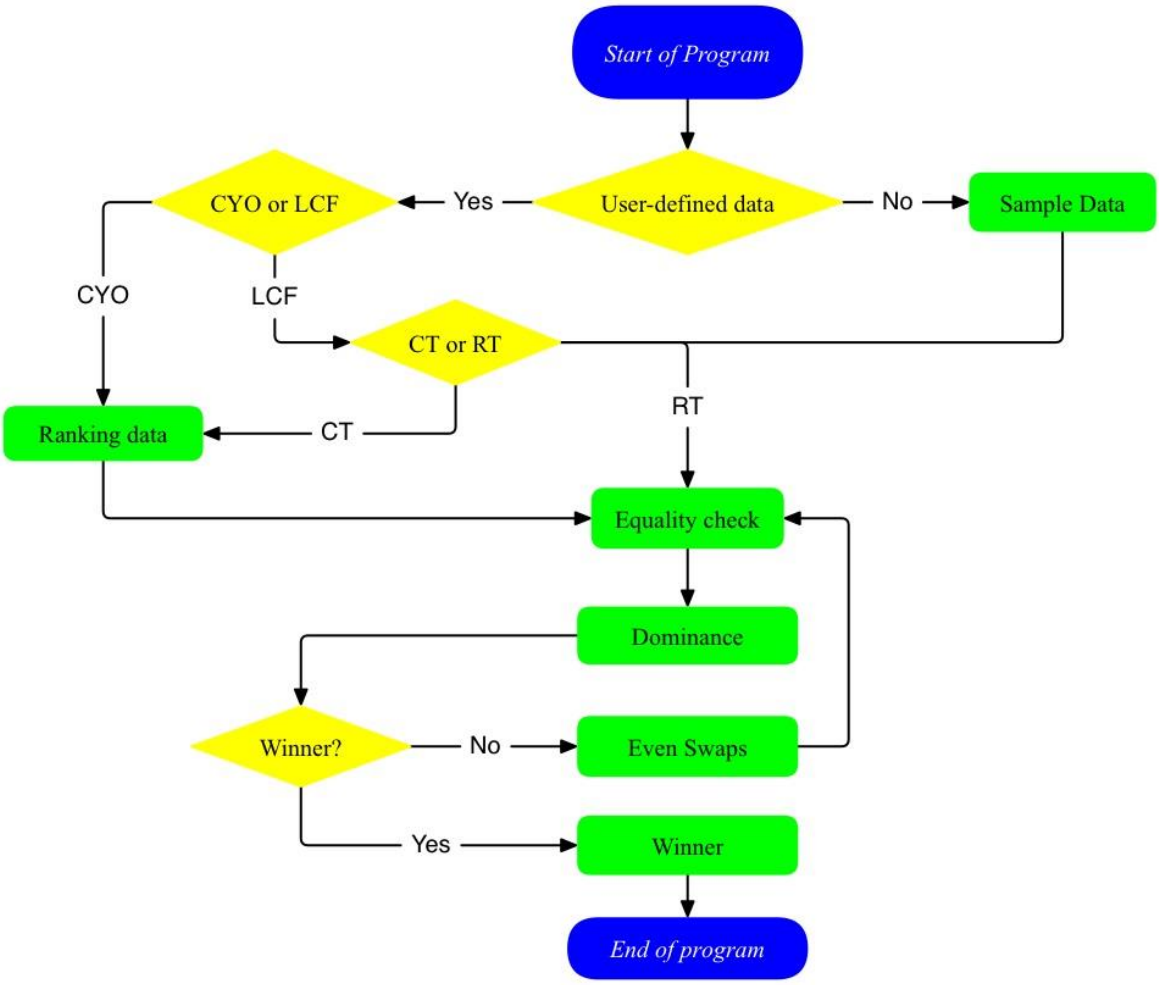


Figure 4.4: CFD version 1 of the ESP

The updated version of the program has a similar flow. However, there are some changes related to the users. In the current version, users are divided into new and returning users. A returning user is most likely familiar with the ES method and does not need any help in how it works, while a new user might not know how it works. Thus, it is important for this user to be able to go through the ES method such that he can become familiar with the method before using it on any decision problem. Another change is that the user is not able to select between user-defined data

and example data anymore, as the example data has moved to the new user option instead, while user-defined data is just in the CYO and LCF options of the program.

```
The following are the rules for using the even swaps program:

1. You are asked to input your username and password
1a. If new user, you will input what you want your username and password to be
1b. Then, the rules for conducting even swaps are displayed for you and you will go through an even swaps example
1c. If returning user, you will be asked whether you would like to create a consequences table (CT) and ranking table (RT)
or if you would like to load a CT or RT from a file
2. You are asked to create an RT if you created a CT in the program or if you loaded a CT from a file
3. Once in RT format, the program checks for equal attributes and removes these and checks for dominance
removing any dominated alternative
4. The program asks whether you would like to make an even swap and displays whether an alternative is likely
to dominate another, and displays the objective ranks that are almost equal such that it is easier to see what
objectives and alternatives you should make swaps on


Enter your username or enter quit to quit the program: quit

Program quit
None
```

Figure 4.5: Rules for using the ESP

The rules for using the ESP are displayed in Figure 4.5. After going through the rules, the user can enter his username and password or quit. If the user is a new user, he will type in what he wants his username to be.

```
Enter your username or enter quit to quit the program: John_Doe

Enter your password or enter quit to quit the program: unknown

Are you happy with your username and password? Y or N: N

Enter your username or enter quit to quit the program: Thanos

Enter your password or enter quit to quit the program: Destroyer of worlds

Welcome to the even swaps program Thanos!
```

Figure 4.6: New user elects to change username and password

In Figure 4.6, a new user has decided that he does not like his username and password, and that he would like to create a new username and password. The user is asked whether he is happy with his username and password. This is done such that the user should be aware that the

username and password is final and cannot be changed if he types Y (Yes). In the other case, if the user chooses N (No), he can change his username and password. However, the user should be aware that he is only able to do this once, and after this, he will be stuck with this username and password combination.

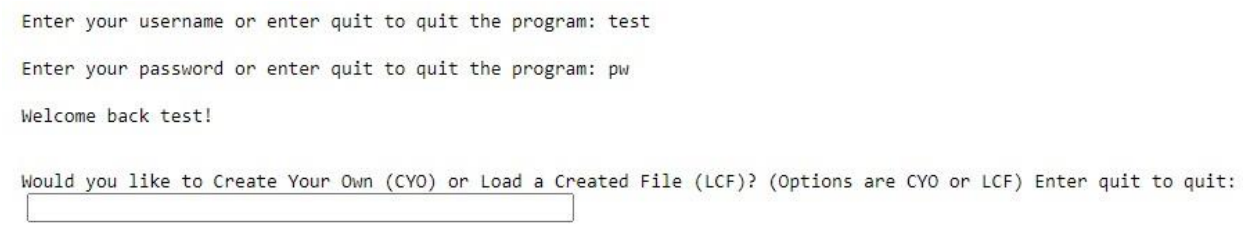


Figure 4.7: Returning user returns to program

Figure 4.7 displays the welcome back message for a returning user. From here, he can create a CT and an RT in either the program or load in a CT or an RT in Excel format that will be converted to a DF as explained previously.

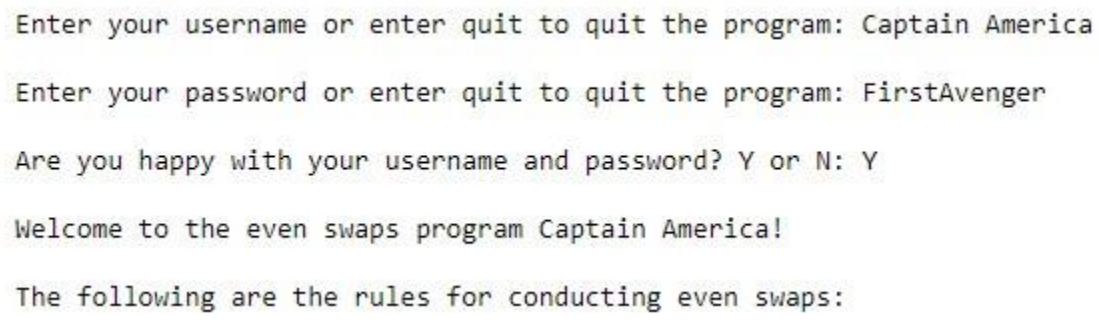


Figure 4.8: New user is happy with his username and password

The user enters his username and password, if he is happy with this combination, he is welcomed to the ESP, and the rules for conducting ES are displayed to him (Figure 4.10).

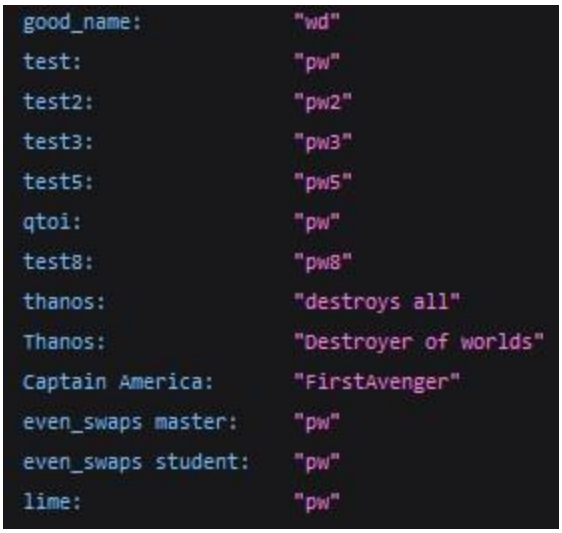


Figure 4.9: File of usernames and passwords

A file of the usernames and passwords is used to keep track of the different users in the program (Figure 4.9). A further version of the program could include better password management, etc. For more details on this, see section XX (4.7). This is important to be able to distinguish between the different users. The users can have the same password, but not the same username.

```
Welcome to the even swaps program test_user!

The following are the rules for conducting even swaps:

1. Create a consequences table (CT) of your alternatives and objectives as shown in fig. 1

2. Create a ranking table (RT) of the CT as shown in fig. 2
2a. This CT and RT are from figures in the Even Swaps article by Hammond et al. (1998)

3. Check for equal ranks on an objective in the RT, if equal ranks this objective
can be removed as shown in fig. 3 and fig. 4

4. Check for absolute and practical dominance to eliminate alternatives
4a. Absolute dominance is when an alternative has better or equal rank on all objectives as shown in fig. 5
4b. Practical dominance is when an alternative has better or equal rank on all but one objective as shown in fig. 5
4c. For this example, Job B absolutely dominates Job E, this is shown in lime
4d. Jobs C and D are being practically dominated by Jobs B and A respectively, this is shown in magenta
4e. Job A is in cyan, while job B is in green. This is just to distinguish between the alternatives and
the different types of dominance

5. The remaining  alternatives and objectives are used for even swaps (fig. 6)
5a. Pick an alternative and an objective, e.g. in fig. 1, it could be alternative 'Job B' and objective 'MS'
5b. For this objective and alternative, change the rank such that it is
equal to the rank of the other remaining alternatives (it is best to do this with alternatives where the rank is similar.
This makes it easier to make a swap.) This is shown in fig. 7
5c. For the same alternative, pick a different objective to make a compensation on, e.g. if you lowered the rank
in step 5b, you would increase the rank for the compensation objective shown in fig. 8
For this example the chosen objective to be swapped is opposite of what the compensation is.
It is easer to make swaps when the values are opposite for the swap and the compensation

6. Repeat steps 3 - 5 until a final alternative remains
```

Figure 4.10: Rules for doing ES for new users

The rules for conducting ES are displayed to the user (Figure 4.10). This lets new users who are unfamiliar with the ES method go through how the method works, and what they can get out of the method (for more details on the ES method, see section 3). Rule 2a. is more of a note to new users that lets them know that the CT and RT mentioned in Rules 1 and 2 are from Hammond et al. (1998).

Fig. 1 Consequences Table

	Objectives	Job A	Job B	Job C	Job D	Job E
0	MS	2000	2400	1800	1900	2200
1	FY	Moderate	Low	High	Moderate	None
2	BSD	Computer	People management, computer	Operations, Computer	Organization	Time management, multitasking
3	AVD	14	12	10	15	12
4	BS	Health, dental, retirement	Health, dental	Health	Health, retirement	Health, dental
5	ET	Great	Good	Good	Great	Boring

Fig. 2 Ranking Table

	Objectives	Job A	Job B	Job C	Job D	Job E
0	MS	3	1	5	4	2
1	FY	2	4	1	2	5
2	BSD	4	1	3	5	2
3	AV	2	3	5	1	3
4	BS	1	2	5	4	2
5	ET	1	3	3	1	5

```
Below are the abbreviations for the objectives for the CT and RT:

MS = Monthly Salary($)
FY = Flexibility
BSD = Business Skills Development
AVD = Annual Vacation Days
BS = Benefits
ET = Enjoyment
```

Figure 4.11: Example CT and RT for new users

From the ES rules, an example CT and RT (Fig. 1 and Fig. 2 in Figure 4.11) is displayed for the user such that he gets an idea of how the different tables in the process work. The abbreviations for the objectives are also displayed. This CT and RT is from Hammond et al. (1998, 1999).

Fig. 3 Equal ranks before removal

	Objectives	Parkway	Lombard	Baranov	Montana	Pierpoint
0	CIM	5	2	1	2	4
1	CA	2	2	4	1	3
2	OS	1	1	1	1	1
3	OSSF	2	3	5	1	3
4	MC	4	2	1	5	3

Fig. 4 Equal ranks after removal

	Objectives	Parkway	Lombard	Baranov	Montana	Pierpoint
0	CIM	5	2	1	2	4
1	CA	2	2	4	1	3
3	OSSF	2	3	5	1	3
4	MC	4	2	1	5	3

Fig. 5 Absolute and Practical dominance

	Objectives	Job A	Job B	Job C	Job D	Job E
0	MS	3	1	5	4	2
1	FY	2	4	1	2	5
2	BSD	4	1	3	5	2
3	AV	2	3	5	1	3
4	BS	1	2	5	4	2
5	ET	1	3	3	1	5

Figure 4.12: Equal attributes and dominance

Figure 4.12 shows how an objective with equal ranks for the alternatives would be removed (figs. 3 and 4 in Figure 4.12), and the distinct colors for absolute and practical dominance. Job A in cyan is practically dominating Job D in magenta. Job B in green is practically dominating Job C in magenta and absolutely dominating Job E in lime.

Fig. 6 Example Ranking Table after removing dominated alternatives

	Objectives	Job A	Job B
0	MS	3	1
1	FY	2	4
2	BSD	4	1
3	AV	2	3
4	BSD	1	2
5	ET	1	3

Fig. 7 Example Ranking Table after doing an even swap

	Objectives	Job A	Job B
0	MS	3	3
1	FY	2	4
2	BSD	4	1
3	AV	2	3
4	BSD	1	2
5	ET	1	1

Fig. 8 Example Ranking table after dropping equal ranks

	Objectives	Job A	Job B
1	FY	2	4
2	BSD	4	1
3	AV	2	3
4	BSD	1	2

Do you understand the even swaps process? Y or N:

Figure 4.13: Example of doing ES

Figure 6 in Figure 4.13 displays the example RT after removing dominated alternatives. From fig. 1 in Figure 4.11, the alternatives were Job A through Job E, but after removing dominated alternatives, the remaining alternatives are Job A and Job B. Furthermore, Figure 4.13 shows how in yellow highlight in Fig. 6, the two objectives are selected for an even swap. In the figure below (Fig. 7), the values have changed to be equal to alternative Job A after doing an even

swap. In the last figure (Fig. 8), the objectives that were chosen for the even swap have been dropped, as they are equal.

Fig. 8 Example Ranking table after dropping equal ranks

	Objectives	Job A	Job B
1	FY	2	4
2	BSD	4	1
3	AV	2	3
4	BSD	1	2

```
Do you understand the even swaps process? Y or N: N

The following are the rules for conducting even swaps:
```

Figure 4.14: User does not understand the ES process

The user is asked whether he understands the ES process, shown in Figure 4.14. In this example, the user selected N (no), which takes him back to the rules for conducting ES such that he can go through the rules again to make sure he understands them.

```
Do you understand the even swaps process? Y or N: Y

Would you like to Create Your Own (CYO) or Load a Created File (LCF)? (Options are CYO or LCF) Enter quit to quit:
```

Figure 4.15: User understands the ES process

Once the user understands the ES process, he selects Y (yes) and will be taken into the normal operations of the ESP, which is to either create a CT, then an RT in the program or load a created CT or RT (Figure 4.15).

4.1 CYO

This section describes what happens in the program if the user selects the CYO option as shown in Figure 4.16.

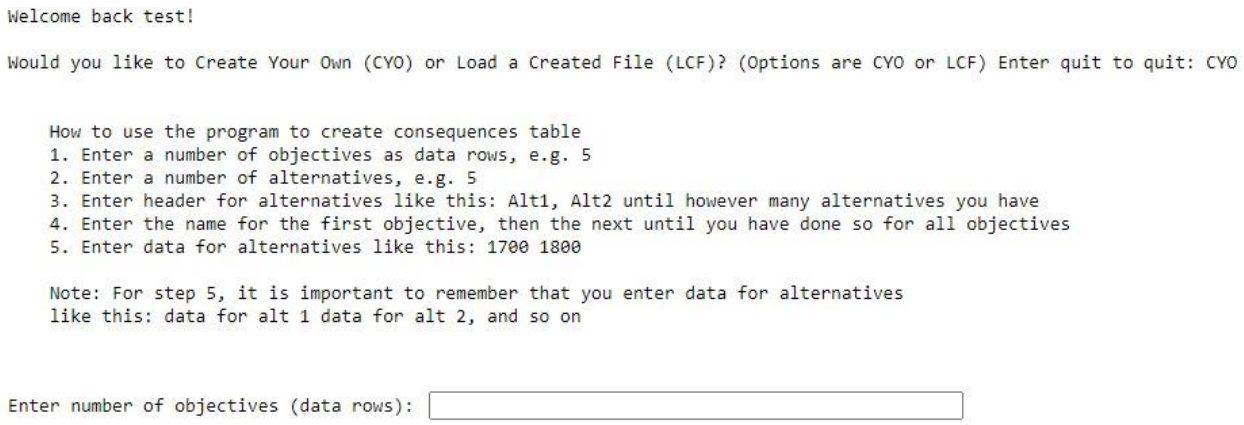


Figure 4.16: Rules for creating CT in ESP

Figure 4.16 displays what happens if the user selects CYO. The user gets a set of rules he must follow. They are listed below in Table 3:

Table 3: Rules for creating a CT

Rules for Creating a CT	
1	Enter a number of objectives as data rows, e.g., 5
2	Enter a number of alternatives, e.g., 5
3	Enter header for alternatives like this: Alt1, Alt2 until however many alternatives you have
4	Enter the name for the first objective, then the next until you have done so for all objectives
5	Enter data for alternatives like this: 1700 1800

Figure 4.16 also illustrates that the user can input how many objectives he would like. This is the first step from Table 3. For step 5, it is important for the user to remember that he is inputting data for each objective for all the alternatives, e.g., say you have one objective called *Price* and three alternative airlines to choose from. They are *United*, *American*, and *KLM*. You would enter the data as 1000 2000 3000 where 1000 would represent the price for the first alternative *United*, 2000 would represent the second alternative *American*, and 3000 would represent the third and final alternative *KLM*. Then you would repeat this process for the other objectives.

Figure 4.17: User enters number of objectives and alternatives

From Figure 4.17, it is evident that for this example, the user has gone through steps one and two of Table 3, where he has inputted the number of objectives (three in this case) and the number of alternatives (also three in this case).

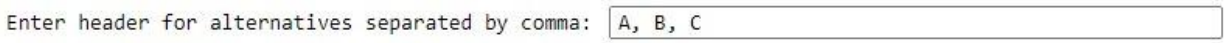


Figure 4.18: User enters names for alternatives

Figure 4.18 shows how the user would enter names for the alternatives. This is the third step in Table 3.

```
Enter name for objective 1: obj1
Enter name for objective 2: obj2
Enter name for objective 3: obj3
What type would you like to use for obj1? Float, Int, or String (F, I, or S): f
Enter data for alternatives for objective obj1 separated by space: 14.3 15.2 16.7
What type would you like to use for obj2? Float, Int, or String (F, I, or S): s
Enter data for alternatives for objective obj2 separated by space: good better best
What type would you like to use for obj3? Float, Int, or String (F, I, or S): i
Enter data for alternatives for objective obj3 separated by space: 4 7 6
Consequences table to be ranked
```

	Objectives	A	B	C
0	obj1	14.3	15.2	16.7
1	obj2	good	better	best
2	obj3	4	7	6

Figure 4.19: User creates CT

The user inputs names for each of the objectives he has as shown in Figure 4.19. From there, the user is asked whether he would like to use type *Int*, *String*, or *Float* for the objectives. *Int* means a whole number, e.g., 10, while *String* means an ordered sequence of characters, e.g., a word like "bread" or a phrase like "the red fox jumped over the white fence." Finally, *Float* means a decimal number, e.g., 14.6. Once the user has selected *Int*, *String*, or *Float*, he inputs the data for the alternatives for the objectives separated by space. When all these steps have been completed, the user is left with a CT in DF format that shows his objectives, his alternatives, and the data for the objectives and alternatives.

Consequences table to be ranked

	Objectives	A	B	C
0	obj1	14.3	15.2	16.7
1	obj2	good	better	best
2	obj3	4	7	6

Dataframe with only string objectives

	A	B	C
0	good	better	best

What would you like the rank for good to be?

Figure 4.20: User is asked to rank string objective

Figure 4.20 displays the created CT as a DF and the row of alternatives that contains *Strings*, which the user was asked about during the creation of the CT. This row of *Strings* is extracted from the DF since it is necessary for the user to be able rank the *Strings* objective according to his preferences, while for the numeric alternatives, the program can rank this. However, it is necessary for the user to input whether the rankings should be ascending or not as shown in Figure 4.22.

Consequences table to be ranked

	Objectives	A	B	C
0	obj1	14.3	15.2	16.7
1	obj2	good	better	best
2	obj3	4	7	6

Dataframe with only string objectives

	A	B	C
0	good	better	best

What would you like the rank for good to be? 1

What would you like the rank for better to be? 2

What would you like the rank for best to be? 3

Ranking data [1, 2, 3]

It is important to remember that the position of the data should be corresponding with the position of the data from the original CT

Which position in the dataframe is the data [1 2 3] from? 1

Figure 4.21: User is asked for index position for string objective

Figure 4.21 shows that the user has inputted a rank for the row containing *String* objectives. Next, the user is asked to input what the index is in the CT for where the data came from. For this example, the data is in index position 1 as can be seen in the DF above the *String* objectives.

Dataframe with only numbers

	A	B	C
0	14.3	15.2	16.7
1	1	2	3
2	4	7	6

Would you like the ranking for [14.3 15.2 16.7] with index 0 to be Ascending or Not? A or N:

Figure 4.22: User is asked whether rankings should be ascending or not

The DF with only numbers is illustrated in Figure 4.22. From here, the user is asked to input whether he would like the data for each row to be ascending or not starting with the first row, which is in index position 0.

Would you like the ranking for [14.3 15.2 16.7] with index 0 to be Ascending or Not? A or N: a

Would you like the ranking for [1 2 3] with index 1 to be Ascending or Not? A or N: n

Would you like the ranking for [4 7 6] with index 2 to be Ascending or Not? A or N: a

Ranked dataframe

	Objectives	A	B	C
0	obj1	1	2	3
1	obj2	3	2	1
2	obj3	1	3	2

The following are the rules for using the program from this stage onwards:

Figure 4.23: Created RT of the created CT

Figure 4.23 exhibits the ranked DF after the user has chosen if each row is ascending or not. From here, the user is getting into the ES part of the program. More details on this are in section 4.3.

4.2 LCF

This section details what happens if the user selects LCF. The user has two options for LCF, either loading a CT or loading an RT.

	A	B	C	D	E	F
1	Objectives	Job A	Job B	Job C	Job D	Job E
2	MS	2000	2400	1800	1900	2200
3	FY	Moderate	Low	High	Moderate	None
4	BSD	Computer	People management, computer	Operations, Computer	Organization	Time management, multitasking
5	AVD	14	12	10	15	12
6	BS	Health, dental, retirement	Health, dental	Health	Health, retirement	Health, dental
7	ET	Great	Good	Good	Great	Boring

Figure 4.24: Example CT in Excel

From Figure 4.24, a CT is shown in Excel. This is how the CT should look like when the user wants to load in a CT. It must be an Excel file otherwise it will not work. The program does not support loading files in other formats.

	A	B	C	D	E	F
1	Objectives	Parkway	Lombard	Baranov	Montana	Pierpoint
2	CIM	5	2	1	2	4
3	CA	5	2	4	1	3
4	OS	1	3	4	1	4
5	OSSF	2	3	5	1	3
6	MC	4	2	1	5	3

Figure 4.25: Example RT in Excel

The RT in Excel format is shown in Figure 4.25. The same applies for the RT as for the CT. In addition, it is important to note that the entered data should start in the first column and first row

as shown in these example tables. This has to do with how the pandas library in Python handles loading in data as a DF, and in this ESP, there is no method to clean the data, so this is something the user should be aware of.

```
Would you like to Create Your Own (CYO) or Load a Created File (LCF)? (Options are CYO or LCF) Enter quit to quit: LCF
Would you like to load a consequences table or a ranking table (CT or RT)? Enter quit to quit: quit
Program Quit
There is no dataframe to do even swaps on
There is no final dataframe
```

Figure 4.26: User is asked whether he would like to load a CT or an RT

The options for the user upon entering the LCF part of the program are to load a CT, an RT, or quit. In Figure 4.26, the user chose to quit the program.

```
Would you like to load a consequences table or a ranking table (CT or RT)? Enter quit to quit: RT
What is the name of the file you would like to load? Enter quit to quit: Test DF Miller 1
Displaying loaded dataframe
```

	Objectives	Parkway	Lombard	Baranov	Montana	Pierpoint
0	CIM	5	2	1	2	4
1	CA	2	2	4	1	3
2	OS	1	1	1	1	1
3	OSSF	2	3	5	1	3
4	MC	4	2	1	5	3

```
Even Swaps Program
```

Figure 4.27: User elected to load in an RT

Figure 4.27 highlights that the user loads in an RT he has already created and enters the ES part of the program. For more details on the ES part of the program, see section 4.3. For more details on the errors in the program, see section 4.4.

Would you like to load a consequences table or a ranking table (CT or RT)? Enter quit to quit: CT

What is the name of the file you would like to load? Enter quit to quit: Test CT Miller 1

Consequences table to be ranked

	Objectives	Parkway	Lombard	Baranov	Montana	Pierpoint
0	CIM	50	30	25	30	35
1	CA	60	90	80	95	85
2	OS	A	A	A	A	A
3	OSSF	850	750	550	1000	750
4	MC	1900	1850	1600	2000	1800

Dataframe with only string objectives

	Parkway	Lombard	Baranov	Montana	Pierpoint
0	A	A	A	A	A

Figure 4.28: User elected to load in a CT

The user selects to load a CT and then he enters the part of the program where he is asked to rank the CT. This is illustrated in Figure 4.28. The reason for this is that the ES in the program does not work for the CT; it only works for the RT. This is seen in the CFD for the program in Figure 4.3. The same rules apply for the CT as the RT about what type of file the user can load (see section 4.1 for details on the ranking of the CT). This ESP handles ES slightly differently from Hammond et al. (1998) and Mustajoki and Hämäläinen (2005, 2007) in the sense that they are conducting ES on the CT, whereas in the ESP, the ES are done on the RT instead. The reason for this is that it is easier to compare ranks that are closer to each other than values in a CT that can be difficult to judge what they represent. However, while using the ESP and conducting ES, it is important to remember what the rankings represent, and not start doing ES without being aware of this. By being aware of this, the users would be more likely to conduct swaps that are aligned

with the objectives that are more important to them, and not just making swaps just because the rankings are almost equal.

The following example has more objectives that are quantitative. This illustrates how the program works when the user opts to load a CT in the LCF option with more quantitative objectives.

```
Would you like to Create Your Own (CYO) or Load a Created File (LCF)? (Options are CYO or LCF) Enter quit to quit: LC
Would you like to load a consequences table or a ranking table (CT or RT)? Enter quit to quit: CT
What is the name of the file you would like to load? Enter quit to quit: Sahid CT
Consequences table to be ranked
```

	Objectives	Job A	Job B	Job C	Job D	Job E
0	MS	2000	2400	1800	1900	2200
1	FY	Moderate	Low	High	Moderate	None
2	BSD	Computer	People management, computer	Operations, Computer	Organization	Time management, multitasking
3	AVD	14	12	10	15	12
4	BS	Health, dental, retirement	Health, dental	Health	Health, retirement	Health, dental
5	ET	Great	Good	Good	Great	Boring

Figure 4.29: Sahid's CT from Hammond et al. (1998) in the ESP

From Figure 4.29, the user has selected the LCF option and opted to load a CT. This CT is one of the example CTs from Hammond et al. (1998).

Consequences table to be ranked

	Objectives	Job A	Job B	Job C	Job D	Job E
0	MS	2000	2400	1800	1900	2200
1	FY	Moderate	Low	High	Moderate	None
2	BSD	Computer	People management, computer	Operations, Computer	Organization	Time management, multitasking
3	AVD	14	12	10	15	12
4	BS	Health, dental, retirement	Health, dental	Health	Health, retirement	Health, dental
5	ET	Great	Good	Good	Great	Boring

Dataframe with only string objectives

	Job A	Job B	Job C	Job D	Job E
1	Moderate	Low	High	Moderate	None
2	Computer	People management, computer	Operations, Computer	Organization	Time management, multitasking
4	Health, dental, retirement	Health, dental	Health	Health, retirement	Health, dental
5	Great	Good	Good	Great	Boring

Figure 4.30: More string objectives to be ranked

The CT/DF with only string objectives is displayed in Figure 4.30. Four objectives are quantitative in this example. It is important to note that the index positions of the string objectives in the string DF correspond to the index positions in the original CT.

```
What would you like the rank for Moderate to be? 2
What would you like the rank for Low to be? 3
What would you like the rank for High to be? 1
What would you like the rank for Moderate to be? 2
What would you like the rank for None to be? 4

What would you like the rank for Computer to be?
```

Figure 4.31: Ranking the FY objective

Figure 4.31 shows that the user has input some ranks for the objective *FY*. The user must input ranks for alternatives that are the same. In this case, this would be *moderate*, which is for alternatives *Job A* and *Job D*.

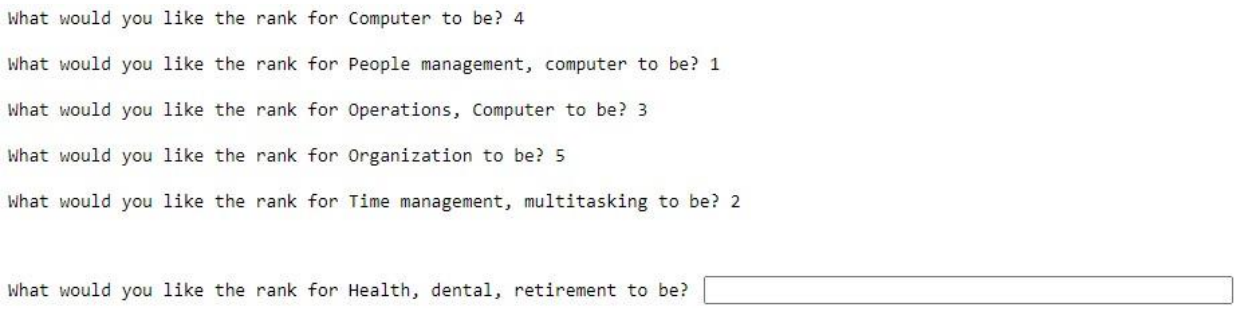


Figure 4.32: Ranking the BSD objective

The user continues to input ranks for the different alternatives for the objectives as illustrated by Figure 4.32. This is for the business skills development (BSD) objective.

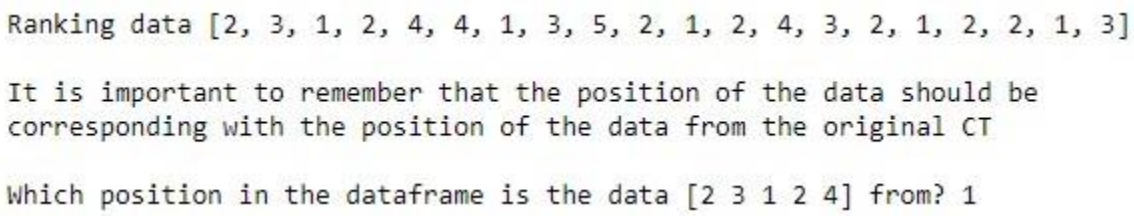


Figure 4.33: Index position for ranks

Once the user has completed ranking all the different alternatives, he is asked which index position the data is from. This is shown in Figure 4.33. From here, it is easy to see that the rankings 2, 3, 1, 2, 4 are for the *FY* objective (Figure 4.30), and so on for the other objectives. There is also a message displayed to the user such that he should be aware to input the correct index position for the data.

```
Ranking data [2, 3, 1, 2, 4, 4, 1, 3, 5, 2, 1, 2, 4, 3, 2, 1, 2, 2, 1, 3]

It is important to remember that the position of the data should be
corresponding with the position of the data from the original CT

Which position in the dataframe is the data [2 3 1 2 4] from? 1

Which position in the dataframe is the data [4 1 3 5 2] from? 2

Which position in the dataframe is the data [1 2 4 3 2] from? 4

Which position in the dataframe is the data [1 2 2 1 3] from? 5

Dataframe with only numbers
```

	Job A	Job B	Job C	Job D	Job E
0	2000	2400	1800	1900	2200
1	2	3	1	2	4
2	4	1	3	5	2
3	14	12	10	15	12
4	1	2	4	3	2
5	1	2	2	1	3

```
******************

Would you like the ranking for [2000 2400 1800 1900 2200] with index 0 to be Ascending or Not? A or N: n
```

Figure 4.34: DF with only numbers

From here, the user continues to input index positions for the ranking data, displayed in Figure 4.34. The DF with only numbers is displayed for the user such that it makes it easier for him to see the changes he made. Furthermore, the user is asked whether he would like the data for the different objectives to be ascending or not. This is shown in Figure 4.35. Once this is complete, the ranked DF is displayed for the user and the program enters the ES part of the program.

```
Would you like the ranking for [2000 2400 1800 1900 2200] with index 0 to be Ascending or Not? A or N: n
Would you like the ranking for [2 3 1 2 4] with index 1 to be Ascending or Not? A or N: a
Would you like the ranking for [4 1 3 5 2] with index 2 to be Ascending or Not? A or N: a
Would you like the ranking for [14 12 10 15 12] with index 3 to be Ascending or Not? A or N: n
Would you like the ranking for [1 2 4 3 2] with index 4 to be Ascending or Not? A or N: a
Would you like the ranking for [1 2 2 1 3] with index 5 to be Ascending or Not? A or N: a
Ranked dataframe
```

	Objectives	Job A	Job B	Job C	Job D	Job E
0	MS	3	1	5	4	2
1	FY	2	4	1	2	5
2	BSD	4	1	3	5	2
3	AVD	2	3	5	1	3
4	BS	1	2	5	4	2
5	ET	1	3	3	1	5

```
The following are the rules for using the program from this stage onwards:
```

Figure 4.35: Finished ranking CT

4.3 Even Swaps in ESP

```
The following are the rules for using the program from this stage onwards:

1. Check for equal rank for the objectives, if True drops these objectives
2. Checks dominance, if alternative is being dominated (practical or absolute) it is dropped
3. User is asked to conduct even swaps on remaining alternatives and objectives
4. User selects one alternative he/she would like to make a swap on to make ranks equal
5. User is asked to compensate for this swap on another objective on same alternative
6. Final alternative should be the decision for the user
7. The user may quit on entering the even swaps part of the program by typing 'Quit' where prompted to
```

Dataframe before removing equal rows

	Job A	Job B	Job C	Job D	Job E
0	3	1	5	4	2
1	2	4	1	2	5
2	4	1	3	5	2
3	2	3	5	1	3
4	1	2	5	4	2
5	1	3	3	1	5

Dataframe after removing equal rows

	Objectives	Job A	Job B	Job C	Job D	Job E
0	MS	3	1	5	4	2
1	FY	2	4	1	2	5
2	BSD	4	1	3	5	2
3	AVD	2	3	5	1	3
4	BS	1	2	5	4	2
5	ET	1	3	3	1	5

Figure 4.36: Rules for doing ES in the ESP

Figure 4.36 exhibits the rules for conducting ES in the ESP after the user has selected one of two options in the beginning, namely CYO or LCF, and done the required operations to get to this stage. Whether the user chooses CYO or LCF, he will always end up in the ES part of the program, unless he makes a mistake or types something wrong such that an error is thrown. For more details on each of these options, see the respective sections: 4.1, 4.2, and 4.4. It is possible

for the user to quit the program before entering the ES part of the program, as shown in several examples previously.

```
Parkway has been dropped from dataframe by the use of practical dominance. It is being dominated by Montana
Displaying new dataframe
```

	Lombard	Baranov	Montana	Pierpoint
0	2	1	2	4
1	2	4	1	3
2	3	4	1	4
3	3	5	1	3
4	2	1	5	3

Figure 4.37: Parkway dropped by practical dominance

```
Lombard has been dropped from dataframe by the use of practical dominance. It is being dominated by Montana
Displaying new dataframe
```

	Baranov	Montana	Pierpoint
0	1	2	4
1	4	1	3
2	4	1	4
3	5	1	3
4	1	5	3

Figure 4.38: Lombard dropped by practical dominance

```
Pierpoint has been dropped from dataframe by the use of practical dominance. It is being dominated by Montana
Displaying new dataframe
```

	Baranov	Montana
0	1	2
1	4	1
2	4	1
3	5	1
4	1	5

Figure 4.39: Pierpoint dropped by practical dominance

Figure 4.37, Figure 4.38, and Figure 4.39 displays how some of the alternatives have been eliminated from the DF by use of dominance, what type of dominance it is, and which alternative is dominating. In this case, *Parkway*, *Lombard*, and *Pierpoint* have been dropped using practical dominance by being dominated by *Montana*, but it is possible for the alternatives to be dropped using absolute dominance as is shown in Figure 4.40. Since *Parkway*, *Lombard*, and *Pierpoint* were all dominated by the same alternative, namely *Montana*, it should make it clear for the user that *Montana* is the best alternative of the remaining alternatives. He should make swaps to make *Montana* dominate the last alternative, *Baranov*. However, it might not always be this simple to see that an alternative is more likely to dominate another, but in similar cases to this, it would at least give the user an indication that an alternative is better than the rest.

	Job A	Job B	Job E
0	3	1	2
1	2	4	5
2	4	1	2
3	2	3	3
4	1	2	2
5	1	3	5

```
Job E has been dropped from dataframe by the use of absolute dominance. It is being dominated by Job B
Displaying new dataframe
```

	Job A	Job B
0	3	1
1	2	4
2	4	1
3	2	3
4	1	2
5	1	3

Figure 4.40: Example of alternative being dropped by absolute dominance

For the difference on the dominance types and more information about how the ES process works, see section 3. It is important to note that practical dominance is defined slightly differently for the use of this program compared with the definition by Hammond et al. (1998). Instead of being applied to a case where a few objectives are dominating and the rest are being dominated or are equal, it is being applied in the case of one objective dominating, while the rest are being dominated or are equal. This makes more alternatives be dominated, but since these alternatives only have one objective being better, they are likely to be considered for ES anyway. By removing these alternatives, it makes the burden for the user easier to deal with the remaining alternatives, as there are less alternatives to consider.

Dataframe before equal rows have been removed

	Baranov	Montana
0	1	2
1	4	1
2	4	1
3	5	1
4	1	5

Dataframe after equal rows have been removed

	Objectives	Baranov	Montana
0	CIM	1	2
1	CA	4	1
2	OS	4	1
3	OSSF	5	1
4	MC	1	5

Figure 4.41: DF before and after equality check

The DF before and after removing rows where the rankings are equal is shown in Figure 4.41. In this case, there are no equal rankings, so no rows have been removed. An example of rows being removed is shown in section 4.3.1.

```
Options are ['Baranov' 'Montana']

Below are the alternative(s) that are likely to dominate another alternative
by having more objectives dominating the other alternative if any

Montana is more likely to dominate Baranov

It is better to make swaps on an alternative that is more likely to be dominated

Below are the ranks that are almost equal if any

Objective CIM for Baranov and Montana is almost equal with these rankings: 1, 2

It is better to make swaps on objectives where the ranks are almost equal

Knowing this information, which alternative would you like to make an even swap on? Type quit to exit program: Baranov
```

Figure 4.42: Likely dominance and almost equal ranks

In Figure 4.42, it is shown that an alternative is more likely to display dominance over another. In this example, *Montana* has three objectives being ranked higher than *Baranov*, while *Baranov* has the remaining two being ranked higher (see Figure 4.41). This lets the user know that he should try to make ES that would make *Montana* dominate *Baranov* either absolutely or practically. Additionally, the program displays the ranks that are almost equal, if any. In this case, there is only one objective that is almost equal, namely *CIM*. The program displays this such that it is evident to the user that he should try to make this objective equal by changing the ranks of one of the alternatives. In addition, together with which alternatives are more likely to dominate another, it should be evident for the user that he should make an even swap on both conditions. I.e., in this example, *Montana* is more likely to dominate *Baranov* and there is one objective that is almost equal, namely *CIM*. The user should make a swap on *CIM* for *Baranov* such that *Montana* will dominate *Baranov*. It is important, however, to remember that these swaps should be based on the preferences of the user and that this program is just a tool to help

him make better decisions. Furthermore, it is important to remember that the program does not make suggestions on which objective the user should compensate for, but the general rule is to make compensation swaps on other objectives that are similar in rank, but opposite. Since the compensation should be in the other direction of the even swap. If the even swap lowers a rank, then the compensation should increase another objective's rank, and vice versa. Finally, the user should be aware of what objectives are most important to him and make swaps on the less important objectives first.

	Objectives	Lombard	Montana
0	CIM	5	3
1	CA	3	1
2	OS	3	7
3	OSSF	3	7
4	MC	3	5

Figure 4.43: DF with no almost equal ranks

```
Below are the ranks that are almost equal if any

There are no ranks that are almost equal

It is better to make swaps on objectives where the ranks are almost equal
```

Figure 4.44: No equal ranks for DF

As displayed in Figure 4.44, no ranks are almost equal for the RT. This shows that the program can handle cases where no ranks are close to being equal. This makes it easier for the user to see that it might not be so easy to make swaps on these alternatives. However, the program does not let the user know anything about ranks that are two ranks apart i.e., the user should be able to make swaps on these objectives without the program telling him that he is able to do so.

The first step is to choose an alternative

	Objectives	Baranov	Montana
0	CIM	1	2
1	CA	4	1
2	OS	4	1
3	OSSF	5	1
4	MC	1	5

Options are ['Baranov' 'Montana']

Which alternative would you like to change? Type quit to exit program:

Figure 4.45: User chooses alternative

Figure 4.45 exhibits the first step in the ES process, which is for the user to choose what alternative he would like to make an even swap on. If the user wants to exit the program at this stage, he may type 'quit' to exit the program as shown in Figure 4.46.

```
Which alternative would you like to change? Type quit to exit program: quit

Program Quit

There is no final dataframe
```

Figure 4.46: User quits program instead of selecting an alternative

However, if the user wishes to continue with the ES process, he chooses what alternative to make a swap on, e.g., in this case he could choose *Baranov*. Once he selects an alternative, the next step is to choose an objective as displayed in Figure 4.47.

```
Options are ['Baranov' 'Montana']

Which alternative would you like to change? Type quit to exit program: Baranov

You have chosen Baranov

The second step is to choose which objective you would like to make a swap on

Options are ['CIM' 'CA' 'OS' 'OSSF' 'MC']

Choose an objective to do an even swap on or type quit to exit program:
```

Figure 4.47: User chooses an objective

```
The second step is to choose which objective you would like to make a swap on

Options are ['CIM' 'CA' 'OS' 'OSSF' 'MC']

Choose an objective to do an even swap on or type quit to exit program: quit

Program Quit

There is no final dataframe
```

Figure 4.48: User quits program instead of selecting an objective

```
Choose an objective to do an even swap on or type quit to exit program: CIM
```

	Objectives	Baranov	Montana
0	CIM	1	2

```
The value for CIM and Baranov is 1

The third step is to choose a new value for the objective and alternative

What would you like the value to be changed to?
```

Figure 4.49: User chooses new value for selected objective and alternative

From Figure 4.47, the options for the objectives are *CIM*, *CA*, *OS*, *OSSF*, and *MC*. The user is also able to quit at this stage if he feels like as illustrated in Figure 4.48. Figure 4.49 shows the value for the selected alternative and objective; in this case, the selected alternative is *Baranov*, while the selected objective is *CIM* such that the value is one. Furthermore, the user is asked

what he would like to change the value to. In addition, the user can see what the value for the other alternative(s) is/are. This makes it easy for the user to see what he could adjust the value to such that he can make the values equal so that an objective can be removed, which is the point of the ES process.

For this example, the value of the other alternative *Montana* is two. If the user feels like it would be too big a step to make the ranks equal, he may instead just increase or decrease the rank by a couple of values. For example, if the rank of *Baranov* for *CIM* was one and the rank for *Montana* was five, it could be better to make a swap that would change the *Baranov* rank to two or the *Montana* rank to four. However, it could be better to look for objectives where the ranks are closer to being equal as this makes it easier to make swaps.

```
What would you like the value to be changed to? 2

Your chosen value is 2

The next step is to choose an objective to compensate for
```

	Objectives	Baranov	Montana
1	CA	4	1
2	OS	4	1
3	OSSF	5	1
4	MC	1	5

```
Options are ['CA', 'OS', 'OSSF', 'MC']

Which objective would you like to compensate for? Type quit to exit program:
```

Figure 4.50: User selects objective to compensate on

From Figure 4.50, the user input two for what the value for *Baranov* and *CIM* should be changed to. The user then gets a choice of the remaining objectives, but for the same alternative to make a compensation on.

The next step is to choose an objective to compensate for

	Objectives	Baranov	Montana
1	CA	4	1
2	OS	4	1
3	OSSF	5	1
4	MC	1	5

Options are ['CA', 'OS', 'OSSF', 'MC']

Which objective would you like to compensate for? Type quit to exit program: CA

	Objectives	Baranov	Montana
1	CA	4	1

The value for the compensated objective: CA and Baranov is 4

Figure 4.51: DF with remaining objectives is displayed

The DF with the objectives for the user to compensate on is shown in Figure 4.51. This makes it easier for the user to see what the ranks are for the remaining objectives he can make a compensation swap on.

The next step is to choose an objective to compensate for

	Objectives	Baranov	Montana
1	CA	4	1
2	OS	4	1
3	OSSF	5	1
4	MC	1	5

Options are ['CA', 'OS', 'OSSF', 'MC']

Which objective would you like to compensate for? Type quit to exit program: quit

Program Quit

There is no final dataframe

Figure 4.52: User quits instead of selecting compensation value

Which objective would you like to compensate for? Type quit to exit program: CA

	Objectives	Baranov	Montana
1	CA	4	1

The value for the compensated objective: CA and Baranov is 4

The next step is to choose a new compensated value for the objective and alternative

What would you like the compensated value to be?

Figure 4.53: User is asked for compensation value

Figure 4.52 illustrates that it is possible for the user to exit the program at this stage as well. The user chose to compensate for the objective *CA* for this example as displayed by Figure 4.53. The user is then asked to choose a new value for this objective.

What would you like the compensated value to be? 3

Your chosen compensated value is 3

Dataframe after doing even swaps operations on it

	Objectives	Baranov	Montana
0	CIM	2	2
1	CA	3	1
2	OS	4	1
3	OSSF	5	1
4	MC	1	5

****

Figure 4.54: DF after making an even swap

Figure 4.54 exhibits that the user input three for the compensated objective *CA* for alternative *Baranov*. The DF after doing ES operations on it is shown after this such that it is clear to the user which alternatives and objectives, he made changes on.

Dataframe before removing equal rows

	Baranov	Montana
0	2	2
1	3	1
2	4	1
3	5	1
4	1	5

Dataframe after removing equal rows

	Objectives	Baranov	Montana
0	CA	3	1
1	OS	4	1
2	OSSF	5	1
3	MC	1	5

Baranov has been dropped from dataframe by the use of practical dominance. It is being dominated by Montana

This is the final alternative

	Objectives	Montana
0	CA	1
1	OS	1
2	OSSF	1
3	MC	5

Figure 4.55: Alternative dropped after doing ES

The DF before and after rows with equal ranks have been removed is illustrated in Figure 4.55. The row with the index position 0 has been removed and the DF has been updated with the remaining objectives. This is because the rankings are equal for this objective. Then, the program checks for dominance again. *Baranov* is dropped, as *Montana* is better on all, but one objective namely *MC*. The new DF with only *Montana* is displayed.

This is the final alternative

	Objectives	Montana
0	CA	1
1	OS	1
2	OSSF	1
3	MC	5

Would you like to create a file of the final dataframe? Y or N: N

You chose N. Have a good day
None

Figure 4.56: User elects to not create a new file of the final alternative

From here, the user is asked whether he would like to create a file of the final DF or not as shown in Figure 4.56. The file type for this is comma-separated values (CSV). In this case, the user chose not to create a file of the DF.

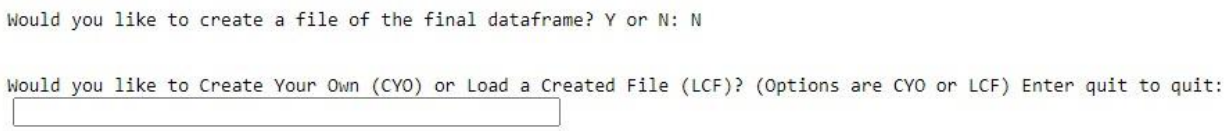


Figure 4.57: User returns to CYO and LCF options

If the user selects N (No), he has the option to begin a new problem or quit the program as is the case in Figure 4.57. However, if the user selects Y (Yes) he is asked to enter the name of the file that will be a CSV file. From there, the user can start a new problem or quit the program (Figure 4.58).

Would you like to create a file of the final dataframe? Y or N: Y

Enter the name of your csv file: Final Alt after ES

Final Alt after ES.csv

Displaying dataframe of final alternative file

	Objectives	Montana
0	CA	1
1	OS	1
2	OSSF	1
3	MC	5

Would you like to Create Your Own (CYO) or Load a Created File (LCF)? (Options are CYO or LCF) Enter quit to quit:

Figure 4.58: User elected to create a file of the final alternative

Figure 4.58 shows that the user has selected a name for his CSV file and the contents of the file are displayed back to him.

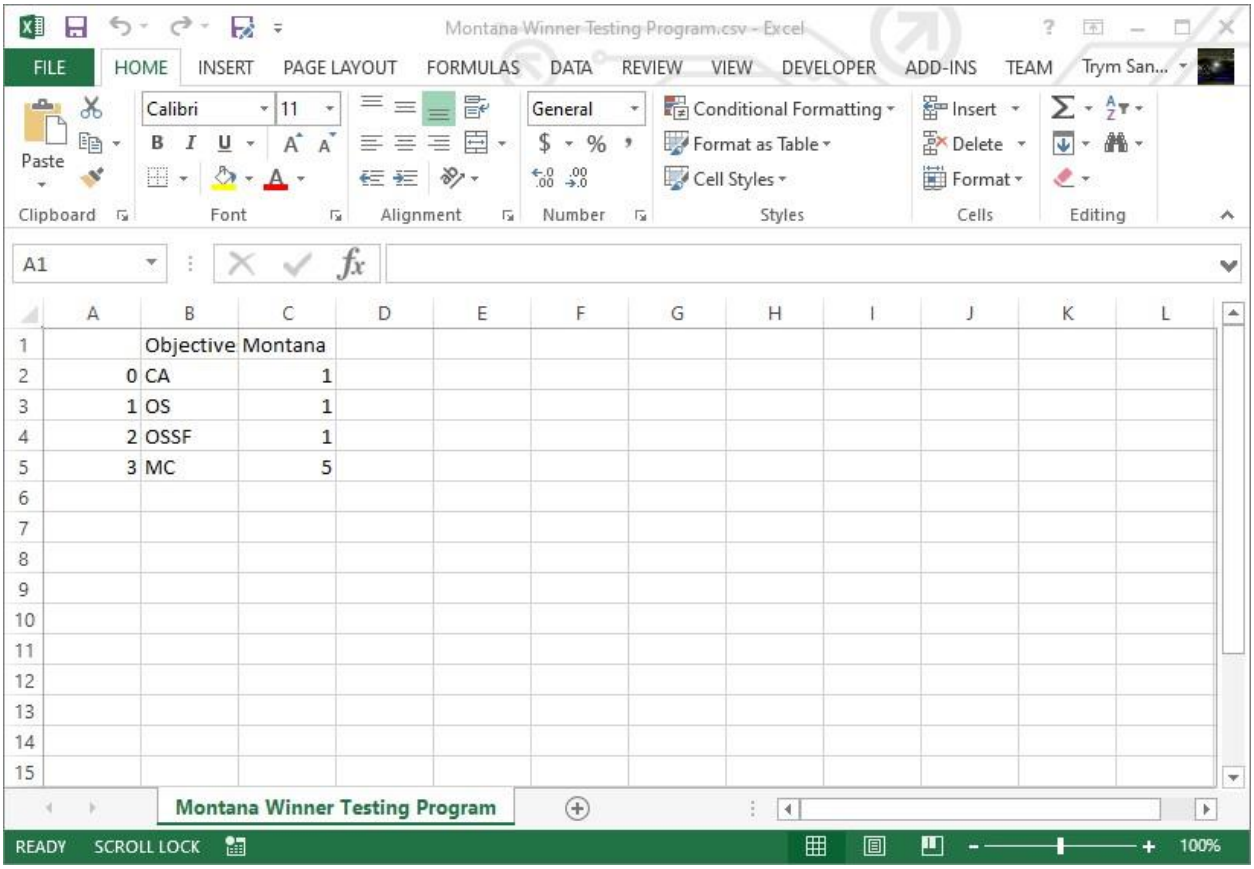


Figure 4.59: Final alternative in Excel

Figure 4.59 illustrates the contents of the CSV file in Excel. It is important to remember that whether the user selects CYO or LCF that the files he would like to work with should be in the right format and in the same folder that the program file is in.

4.3.1 Example of rows being removed during ES operations

Displaying loaded dataframe

	Objectives	Parkway	Lombard	Baranov	Montana	Pierpoint
0	CIM	5	2	1	2	4
1	CA	2	2	4	1	3
2	OS	1	1	4	1	4
3	OSSF	2	1	5	1	3
4	MC	4	2	1	5	3

Figure 4.60: Example DF for removing rows during ES operations

Figure 4.60 displays an example DF being used to demonstrate how equal rows are removed from the DF after applying ES on it. This DF is a slight variation on Miller's RT from Hammond et al. (1998).

The first step is to choose an alternative

	Objectives	Lombard	Baranov	Montana
0	CIM	2	1	2
1	CA	2	4	1
2	OS	1	4	1
3	OSSF	1	5	1
4	MC	2	1	5

Figure 4.61: Dominated alternatives have been removed

Following the DF in Figure 4.60, a new DF is made after going through a dominance check to remove alternatives (Figure 4.61). It is clear to see that the *Parkway* and *Pierpoint* alternatives have been removed.

Dataframe after doing even swaps operations on it

	Objectives	Lombard	Baranov	Montana
0	CIM	2	2	2
1	CA	2	3	1
2	OS	1	4	1
3	OSSF	1	5	1
4	MC	2	1	5

Figure 4.62: DF after doing ES

From the DF in Figure 4.62, the objective *CIM* have been changed for the alternative *Baranov* such that it is equal for all alternatives. This means that this objective can be removed from the DF. The objective *CIM* have been compensated for by a change in the *CA* objective.

Dataframe after removing equal rows

	Objectives	Lombard	Baranov	Montana
0	CA	2	3	1
1	OS	1	4	1
2	OSSF	1	5	1
3	MC	2	1	5

Baranov has been dropped from dataframe by the use of practical dominance. It is being dominated by Lombard

Displaying new dataframe

	Lombard	Montana
0	2	1
1	1	1
2	1	1
3	2	5

Figure 4.63: Alternative dropped after ES

Figure 4.63 illustrates that the *Baranov* alternative has been removed using practical dominance. From here, the remaining alternatives are *Lombard* and *Montana*. It is evident that the *OS* and

OSSF objectives have become equal after removing *Baranov* such that these objectives should also be removed from consideration.

Dataframe before equal rows have been removed

	Lombard	Montana
0	2	1
1	1	1
2	1	1
3	2	5

Dataframe after equal rows have been removed

	Objectives	Lombard	Montana
0	CA	2	1
1	MC	2	5

The first step is to choose an alternative

	Objectives	Lombard	Montana
0	CA	2	1
1	MC	2	5

Options are ['Lombard' 'Montana']

Figure 4.64: Equal rows dropped

The *OS* and *OSSF* objectives have been removed because of their equal ranks and the user is able to continue with the ESP with the remaining objectives (Figure 4.64).

4.4 Errors in the use of ESP

It is important to remember that there may be several user errors during the handling of the program. The user errors that may occur during the use of the program are described in this section. However, the program might not handle some user errors, as it is difficult to account for all the several types of user errors that could occur during the use of a program. All the error messages from the figures in this section are in bold. This makes it easier for the user to see when something goes wrong and what the message is. In all the places the user is asked for input, it is worth noting that punctuation matters a great deal, e.g., if the name of an alternative is Baranov, the user must type it exactly like this and not use variations where some letters are capitalized, etc.

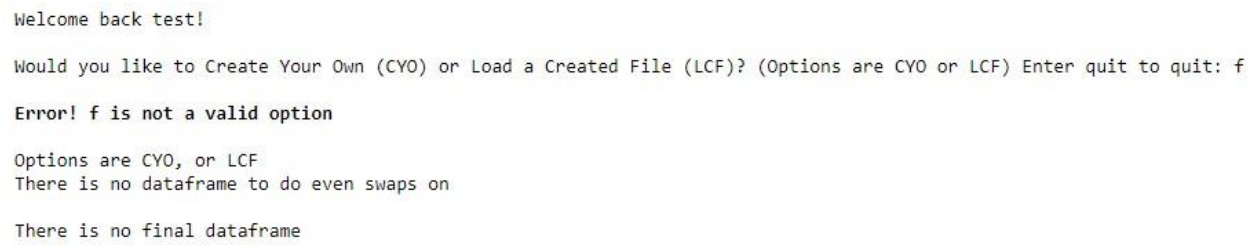


Figure 4.65: User does not enter correct option for CYO and LCF

Figure 4.65 exhibits the error message for the user when he tries to type anything other than CYO, LCF, or quit. The user is not able to continue with the program and will have to run it again.

```
Which alternative would you like to change? Type quit to exit program: fg
Error! fg is not a valid name
Options are ['Baranov' 'Montana']
Dataframe before removing equal rows
```

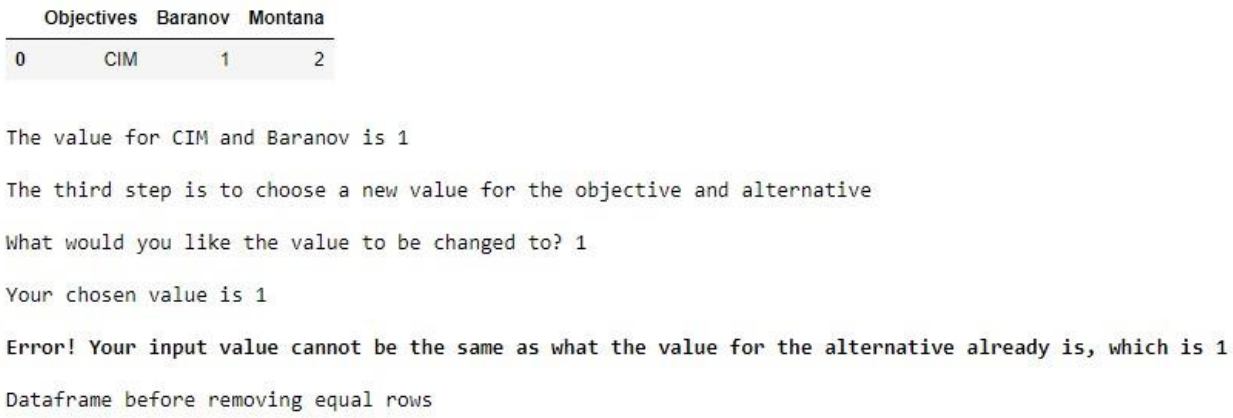
Figure 4.66: User enters incorrect name of alternative

The error message when the user is in the ES part of the program where he is asked what alternative he would like to make an even swap on is displayed by Figure 4.66. The alternatives are displayed to him such that he is aware of the alternatives he can work with.

```
The second step is to choose which objective you would like to make a swap on
Options are ['CIM' 'CA' 'OS' 'OSSF' 'MC']
Choose an objective to do an even swap on or type quit to exit program: fg
Error! fg is not part of the list of objectives
Options are ['CIM' 'CA' 'OS' 'OSSF' 'MC']
Dataframe before removing equal rows
```

Figure 4.67: User enters name of objective not in RT

From Figure 4.67, it is illustrated that there is an error message when the user enters a name for an objective that is not in the list of the objectives for the problem he is working on. The objectives are displayed for the user in a similar manner to how the alternatives are displayed when he makes a typo for an alternative.

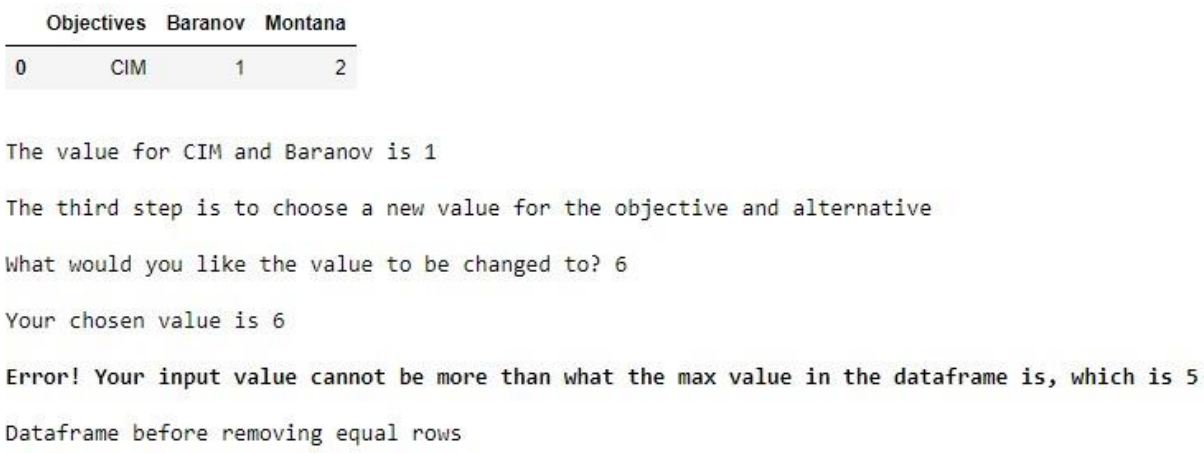


	Objectives	Baranov	Montana
0	CIM	1	2

```
The value for CIM and Baranov is 1
The third step is to choose a new value for the objective and alternative
What would you like the value to be changed to? 1
Your chosen value is 1
Error! Your input value cannot be the same as what the value for the alternative already is, which is 1
Dataframe before removing equal rows
```

Figure 4.68: User enters swap value that is the same as the value already is

Figure 4.68 shows the error message when the user inputs the same value for the alternative as what the value already is. In this case, the value for the selected alternative and objective is 1 and the user tried to input 1, which will not work. Nor does it have any meaning, as the point is to adjust the value such that objectives, and then eventually alternatives can be removed.



	Objectives	Baranov	Montana
0	CIM	1	2

```
The value for CIM and Baranov is 1
The third step is to choose a new value for the objective and alternative
What would you like the value to be changed to? 6
Your chosen value is 6
Error! Your input value cannot be more than what the max value in the dataframe is, which is 5
Dataframe before removing equal rows
```

Figure 4.69: User enters value that is greater than the greatest rank in the RT

The error message when the user tries to input a value that is greater than the highest value in the DF across all alternatives is displayed by Figure 4.69. In this case, the highest value is 5 and the user tried to input 6. The reason for including an error message for this scenario is that it does

not make sense for the user to be able to enter a value that is higher than any ranking in the DF as the rankings are based on the number of alternatives. E.g., if you have three alternatives, the ranking would be 1, 2, 3, and it would not make sense to be able to rank one of the alternatives as 4. Another example is if the rankings are 1, 1, and 3. Then, it would not make sense for the ranking to be 4, as the highest rank is 3.

	Objectives	Baranov	Montana
0	CIM	1	2

```
The value for CIM and Baranov is 1
The third step is to choose a new value for the objective and alternative
What would you like the value to be changed to? -1
Your chosen value is -1
Error! Your chosen value is less than or equal to zero, but it has to be positive for the even swap
Dataframe before removing equal rows
```

Figure 4.70: User enters value that is less than lowest value in RT

Figure 4.70 exhibits the error message when the user tried to input a negative number for the ranking that he would like to change. As is stated in the error message from this figure, the ranking cannot be negative or zero. This is because it does not make sense to input a negative or zero rank, as ranks are from 1 until however many alternatives the user is dealing with.

	Objectives	Baranov	Montana
0	CIM	1	2

```
The value for CIM and Baranov is 1
The third step is to choose a new value for the objective and alternative
What would you like the value to be changed to? fg
Error! That is not an Int number. Please enter an Int number
```

Figure 4.71: User enters non-int for swap value

From Figure 4.71, the user tried to enter a non-*Int* for what he would have liked to change the value into. This error checking case catches cases when the user enters anything that is not of type *Int*. This is because it does not make sense to enter anything else for this input. The user is not able to exit the program at this stage, but if he types a non-*Int*, he will come back to the scenario displayed in Figure 4.45. From here, the user can exit the program. The same applies for when the user is dealing with the compensation value.

```
The next step is to choose an objective to compensate for
```

	Objectives	Baranov	Montana
1	CA	4	1
2	OS	4	1
3	OSSF	5	1
4	MC	1	5

```
Options are ['CA', 'OS', 'OSSF', 'MC']
Which objective would you like to compensate for? Type quit to exit program: CIM
Error! Your chosen compensated objective cannot be the same as the objective you would like to compensate for
```

Figure 4.72: User enters same name for compensated objective as swap objective

Figure 4.72 shows the error message when the user tries to enter the name of an objective he would like to compensate for. It does not make sense for the user to be able to compensate on the

same objective that he would like to make an even swap on, as it defeats the whole purpose of the ES process. The options that are available to the user to make a compensation on are displayed to him such that there should be no confusion as to which objectives he can make a compensation on.

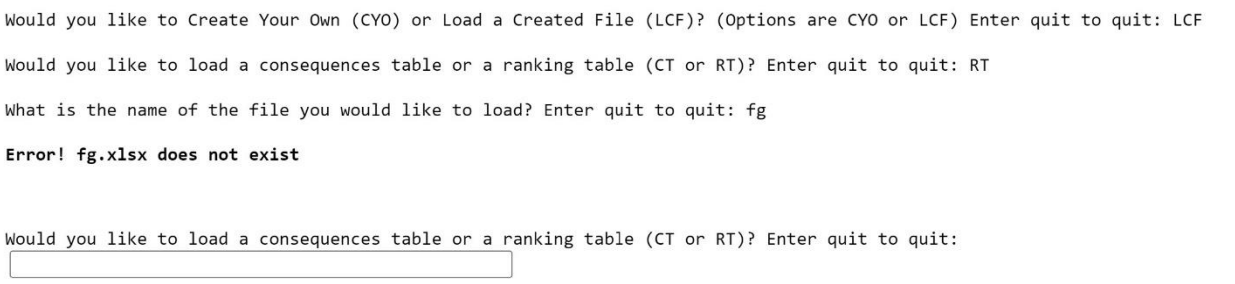


Figure 4.73: User enters the name of Excel file that does not exist or is not in same folder as program

Figure 4.73 illustrates the error message when the user tries to enter the name of a file that does not exist or is not located in the same folder as the program file. This happens in the LCF option of the program. The user is then asked whether he would like to load a CT or an RT again.

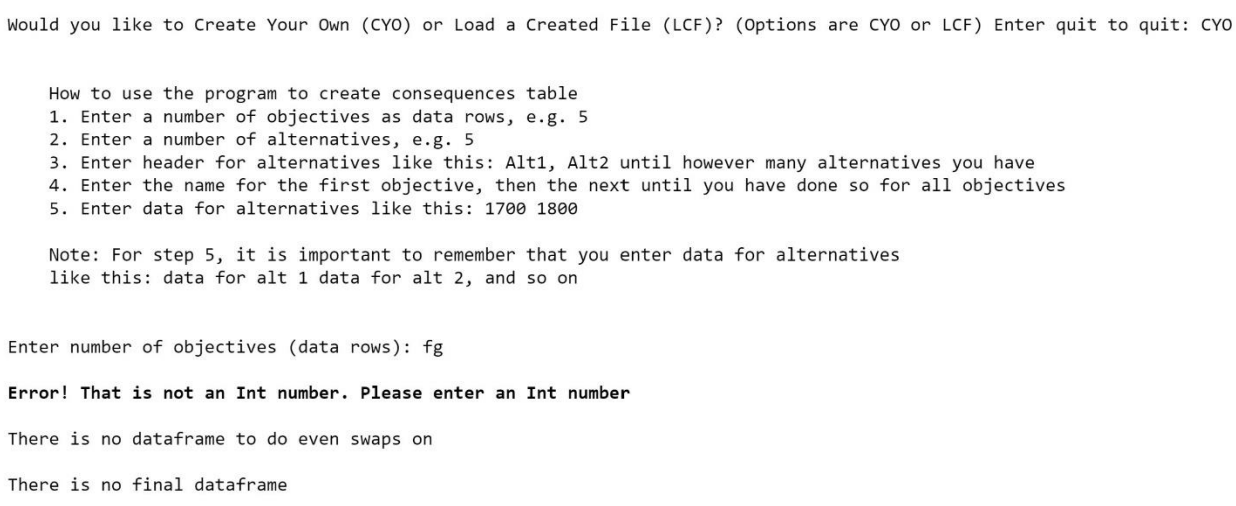


Figure 4.74: User enters non-int for number of objectives

Figure 4.74 displays the error message for when the user attempts to enter something that is not an *Int* for the number of objectives in the CYO part of the program. The same applies for step 2 from Table 3 as can be seen in Figure 4.75.

```
Enter number of objectives (data rows): 2

Enter number of alternatives: fg

Error! That is not an Int number. Please enter an Int number

There is no dataframe to do even swaps on

There is no final dataframe
```

Figure 4.75: User enters non-int for number of alternatives

```
Would you like to Create Your Own (CYO) or Load a Created File (LCF)? (Options are CYO or LCF) Enter quit to quit: LCF

Would you like to load a consequences table or a ranking table (CT or RT)? Enter quit to quit: fg

Error! fg is not a valid option

Options are CT or RT

Would you like to load a consequences table or a ranking table (CT or RT)? Enter quit to quit:
```

Figure 4.76: User does not type CT or RT

The error message for when the user attempts to enter something that is not CT or RT for the option of whether he would like to load a CT, or an RT is shown by Figure 4.76. The program returns the user to the option of whether he would like to load a CT or an RT.

```
Ranking data [1, 1, 1, 1, 1]

Which position in the dataframe is the data [1 1 1 1 1] from? fg

Error! That is not an Int number. Please enter an Int number

There is no dataframe to do even swaps on

There is no final dataframe
```

Figure 4.77: User enters non-int for index position

From Figure 4.77, the error message for when the user attempts to enter a non-*Int* for the index position when he wants to input where in the DF the ranking data is from is displayed.

```
Dataframe with only string objectives
```

	Parkway	Lombard	Baranov	Montana	Pierpoint
0	A	B	C	D	E

```
What would you like the rank for A to be? fg

Error! That is not an Int number. Please enter an Int number

There is no dataframe to do even swaps on

There is no final dataframe
```

Figure 4.78: User enters non-int for rankings

Figure 4.78 exhibits the error message when the user attempts to enter a non-*Int* for the ranking of the rows with *String* objectives.

Dataframe with only string objectives

	Parkway	Lombard	Baranov	Montana	Pierpoint
0	A	B	C	D	E

What would you like the rank for A to be? -1

Error! Your chosen rank is less than or equal to zero, but it has to be positive

There is no dataframe to do even swaps on

There is no final dataframe

Figure 4.79: User enters rank that is less than one

The error message shown in Figure 4.79 is for when the user attempts to enter a rank, which is negative or zero for the string objectives in the ranking part of the program. It does not make sense for the user to be able to enter a rank of this type, as it serves no purpose for a rank to be zero or negative. In the same sense, it does not make sense for the index position of where the data is coming from in the DF to be negative. However, it is possible to have a zero index. Therefore, there is an error message if the index position is negative as seen in Figure 4.80.

Ranking data [1, 2, 3, 4, 5]

Which position in the dataframe is the data [1 2 3 4 5] from? -1

Error! Your chosen index position is negative, but it has to be zero or greater

There is no dataframe to do even swaps on

There is no final dataframe

Figure 4.80: User enters incorrect type for index position

From Figure 4.81, the user entered the wrong password. He is then asked to enter his username and password again to login.

```
Enter your username or enter quit to quit the program: test
Enter your password or enter quit to quit the program: dw
Incorrect password for user: test
Enter your username or enter quit to quit the program: test
Enter your password or enter quit to quit the program: pw
Welcome back test!
```

Figure 4.81: User enters incorrect password

```
Displaying loaded dataframe
```

	Objectives	Parkway	Lombard	Baranov	Montana	Pierpoint
0	CIM	2	1.0	3	5.0	4
1	CA	1	1.0	1	1.0	1
2	OS	1	NaN	1	1.0	1
3	OSSF	1	1.0	1	1.0	1
4	MC	1	1.0	1	NaN	1

Figure 4.82: DF with NaNs

A DF with NaNs (Not a Number) is shown in Figure 4.82. This is used to show what happens if the user loads in a file with NaNs in it.

Dataframe before removing equal rows

	Parkway	Lombard	Baranov	Montana	Pierpoint
0	2	1.0	3	5.0	4
1	1	1.0	1	1.0	1
2	1	NaN	1	1.0	1
3	1	1.0	1	1.0	1
4	1	1.0	1	NaN	1

Error! Not possible to do even swaps as there are or could be NaNs in the dataframe. There is no final alternative

There is no final dataframe

Figure 4.83: User tries to do ES on DF with NaNs

From Figure 4.83, an error message is displayed if there are NaNs in the DF. It is not possible to make ES on a DF with missing data. Therefore, the user should be aware that the data he is loading in does not contain any missing data. This can be difficult to spot if the table is large containing much data.

4.5 Probable dominance example

The ESP is supposed to work with Bayesian updating and probable dominance as seen in the work by Bhattacharjya and Kephart (2014). The ESP does not work like this now, but a simple program shows how the two-attribute example from Bhattacharjya and Kephart (2014) works.

The same principles as seen in section 3.4.2 applies, i.e., M (the attributes) = 2, and the DM's marginal value functions are linear and normalized to between 0 and 1. Let the system believe that $w_1 \approx Uniform(0, 1)$. From Bhattacharjya and Kephart (2014), $\boldsymbol{x}$ = (0.2, 0.6), which in this sense can be represented by alternative 1, or a_1, where $x_1(a_1) = 0.2$ and $x_2(a_1) = 0.6$. Furthermore, the value of alternative 1 can be represented by $v(a_1) = w_1 x_1(a_1) + (1 - w_1)x_2(a_1) = 0.2w_1 + 0.6(1 - w_1) = 0.6 - 0.4w_1$.

Alternative 2, or a_2 leads to attributes that are unknown, i.e., $x_1(a_2) = ?$ and $x_2(a_2) = ?$. For simplicity, the notation (a_2) in $x_1(a_2)$ and $x_2(a_2)$ is dropped such that the value of alternative 2 is then given by $v(a_2) = w_1x_1 + (1 - w_1)x_2 = x_2 + (x_1 - x_2)w_1$. If a_1 dominates a_2, $v(a_1) > v(a_2)$, then for $x_1(a_1) = 0.2$ and $x_2(a_1) = 0.6$:

$$0.6 - 0.4w_1 > x_2 + (x_1 - x_2)w_1$$

$$0.6 - x_2 > (x_1 - x_2 + 0.4)w_1$$

$$\begin{cases} \dfrac{0.6 - x_2}{x_1 - x_2 + 0.4} > w_1, & if\ x_1 - x_2 + 0.4 > 0 \\ \dfrac{0.6 - x_2}{x_1 - x_2 + 0.4} < w_1, & otherwise \end{cases}$$

For a_1 absolutely dominates a_2, then

$$\begin{cases} \dfrac{0.6 - x_2}{x_1 - x_2 + 0.4} > 1, & if\ x_1 - x_2 + 0.4 > 0 \\ \dfrac{0.6 - x_2}{x_1 - x_2 + 0.4} < 0, & otherwise \end{cases}$$

because w_1 is an uncertain number between 0 and 1. Thus, for a_1 absolutely dominating a_2, then

$$\begin{cases} x_1 < 0.2, & if\ x_1 - x_2 + 0.4 > 0 \\ x_2 < 0.6, & otherwise \end{cases}$$

which means that if $x_1 < 0.2$ and $x_2 < 0.6$, a_1 absolutely dominates a_2. If alternative a_1 leads to attributes $x_1(a_1) = 0.2$ and $x_2(a_1) = 0.6$, the probability that alternative a_1 dominates a_2 is

$$P_{a_1a_2} = \begin{cases} 1, & if\ x_1 < 0.2\ and\ x_2 < 0.6 \\ \dfrac{0.6 - x_2}{x_1 - x_2 + 0.4}, & if\ x_1 - x_2 + 0.4 > 0 \\ 1 - \dfrac{0.6 - x_2}{x_1 - x_2 + 0.4}, & if\ x_1 - x_2 + 0.4 < 0 \\ 0, & if\ x_1 > 0.2\ and\ x_2 > 0.6 \end{cases}$$

where $P_{a_1 a_2} = 1$ means that a_1 absolutely dominates a_2, $P_{a_1 a_2} = 0$ means that a_1 is absolutely dominated by a_2, and $0 < P_{a_1 a_2} < 1$ are the regions of potential practical dominance (as determined by probable dominance) where $w_1 = U(0,1)$.

This example can be further generalized for other values of $x_1(a_1)$ and $x_2(a_1)$. Let the system believe that $w_1 \approx Uniform(u_{min}, u_{max})$. Alternative 1, or a_1 leads to attributes $x_1(a_1) = x_1^1$ and $x_2(a_1) = x_2^1$. Furthermore, the value of alternative 1 can be represented by $v(a_1) = w_1 x_1^1 + (1 - w_1)x_2^1 = x_2^1 + (x_1^1 - x_2^1)w_1$.

Alternative 2, or a_2 leads to attributes $x_1(a_2) = x_1^2$ and $x_2(a_2) = x_2^2$. The value of alternative 2 is then given by $v(a_2) = w_1 x_1^2 + (1 - w_1)x_2^2 = x_2^2 + (x_1^2 - x_2^2)w_1$. If a_1 dominates a_2, $v(a_1) > v(a_2)$, then

$$x_2^1 + (x_1^1 - x_2^1)w_1 > x_2^2 + (x_1^2 - x_2^2)w_1$$

$$x_1^2 - x_2^2 > (x_1^2 - x_2^2 - x_1^1 + x_2^1)w_1$$

If $x_1^2 - x_2^2 - x_1^1 + x_2^1 > 0$

$$w_1 < \frac{x_1^2 - x_2^2}{x_1^2 - x_2^2 - x_1^1 + x_2^1}$$

and if $x_1^2 - x_2^2 - x_1^1 + x_2^1 < 0$

$$w_1 < 1 - \frac{x_1^2 - x_2^2}{x_1^2 - x_2^2 - x_1^1 + x_2^1}$$

Then, if a_1 leads to attributes $x_1(a_1) = x_1^1$ and $x_2(a_1) = x_2^1$, the probability that a_1 dominates a_2 is

$$P_{a_1a_2} = \begin{cases} 1, & if\ x_1 < x_1^1\ and\ x_2 < x_2^1 \\ \dfrac{x_1^2 - x_2^2}{x_1^2 - x_2^2 - x_1^1 + x_2^1}, & if\ x_1^2 - x_2^2 - x_1^1 + x_2^1 > 0 \\ 1 - \dfrac{x_1^2 - x_2^2}{x_1^2 - x_2^2 - x_1^1 + x_2^1}, & if\ x_1^2 - x_2^2 - x_1^1 + x_2^1 < 0 \\ 0, & if\ x_1 > x_1^1\ and\ x_2 > x_2^1 \end{cases}$$

where $P_{a_1a_2} = 1$ means that a_1 absolutely dominates a_2, $P_{a_1a_2} = 0$ means that a_1 is absolutely dominated by a_2 and $0 < P_{a_1a_2} < 1$ are the regions of potential practical dominance (as determined by probable dominance) where $w_1 = U(0,1)$. If the distribution over w_1 is $\boldsymbol{U}(u_{min}, u_{max})$, w_1 can be any value in [0, 1], but the probability $P_{a_1a_2}$ is constrained by $\boldsymbol{U}(u_{min}, u_{max})$. Furthermore, if $\boldsymbol{U}(u_{min} = 0, u_{max} = 1)$, $P_{a_1a_2} = w_1$. However, the general case for any $[u_{min}, u_{max}]$ is

$$F(x) < \begin{cases} 0, & if\ x < u_{min} \\ \dfrac{x - u_{min}}{u_{max - u_{min}}}, & if\ u_{min} \le x \le u_{max} \\ 1, & if\ x > u_{max} \end{cases}$$

For any quantile x, the inverse function is given by

$$p = \frac{x - u_{min}}{u_{max - u_{min}}}$$

and so, for any w_1, the inverse function is given by

$$p = \frac{w_1 - u_{min}}{u_{max - u_{min}}}$$

From this generalization of the two-attribute example, it is possible to test different values for $x_1$and x_2, and different values for $\boldsymbol{U}(u_{min}, u_{max})$. Figure 4.84 shows the reproduced Figure 3.13 from section 3.4.2, while Figure 4.85 is the reproduced Figure 3.14 from section 3.4.2. The black

point represents the point $x_1 = 0.2$ and $x_2 = 0.6$ from the description above, which is the same coordinates that were used by Bhattacharjya and Kephart (2014). The same uniform distribution is used in Figure 4.85, namely $\boldsymbol{U}(0.4, 0.6)$.

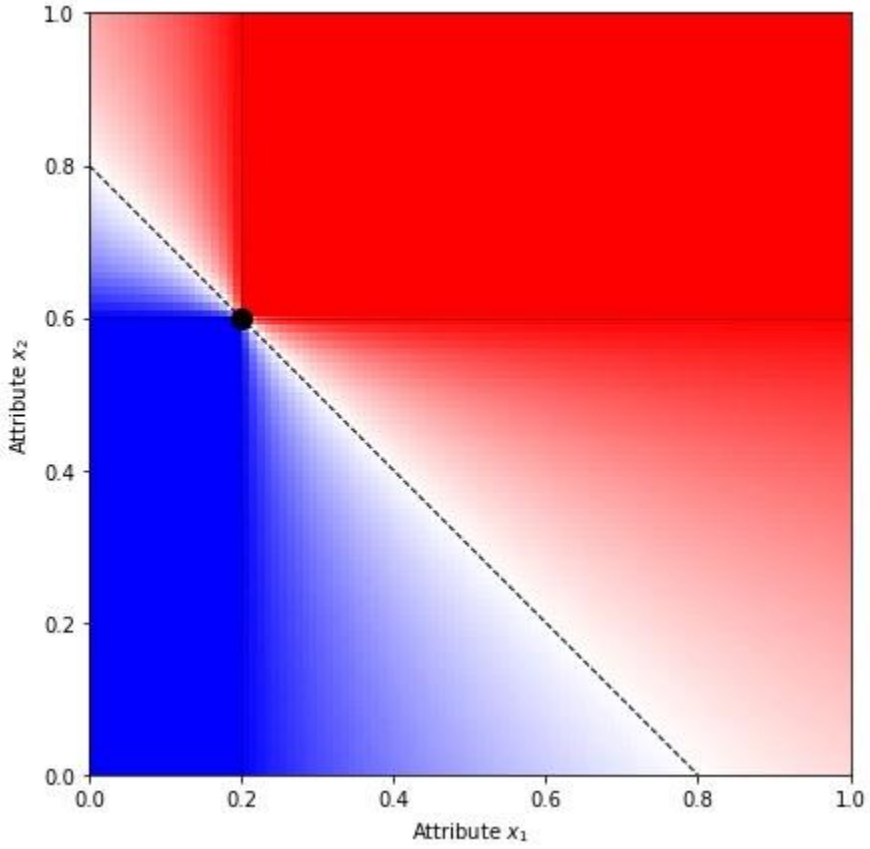


Figure 4.84: Reproduced figure 3.13 of two-attribute example

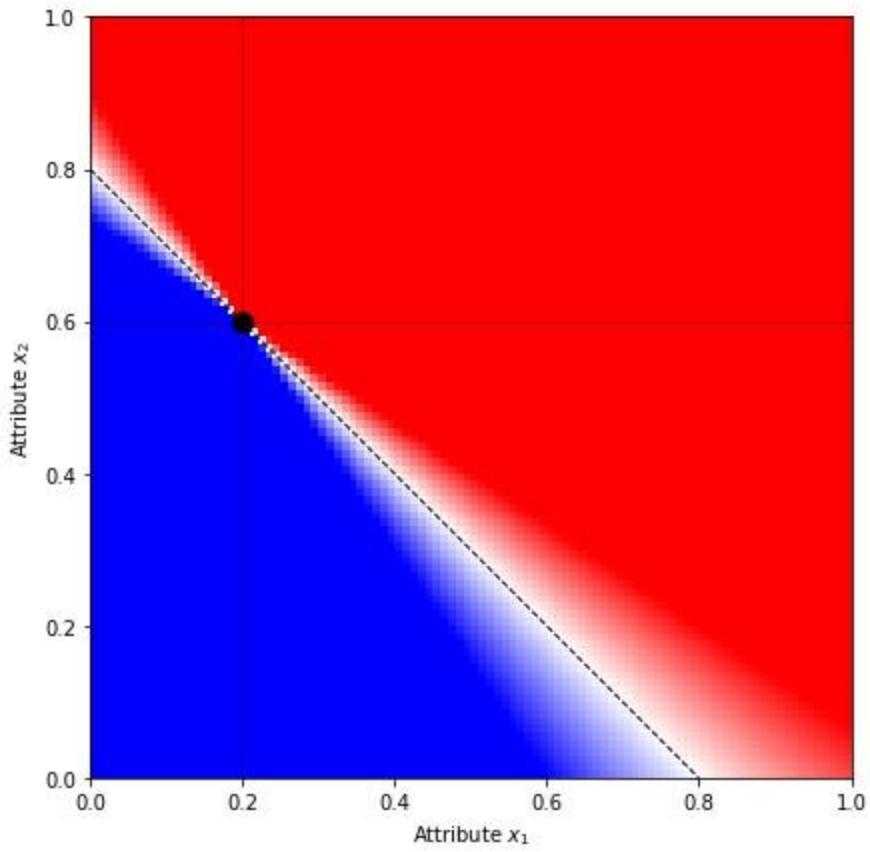


Figure 4.85: Reproduced figure 3.14 of two-attribute example

From here, it is possible to adjust the value for x_1 and x_2, e.g., $x_1 = 0.4$ and $x_2 = 0.8$ leads to Figure 4.86. The red section, which shows alternatives that absolutely dominate a_1 is much smaller for this example, while the blue section, which shows alternatives that are absolutely dominated by a_1 is larger for this example. Therefore, it is possible to adjust these coordinates to see how the regions of absolute dominance and potential practical dominance (shown in the lighter shaded regions) will adjust. This makes it easy to find how one alternative might absolutely dominate another by testing different coordinate pairs.

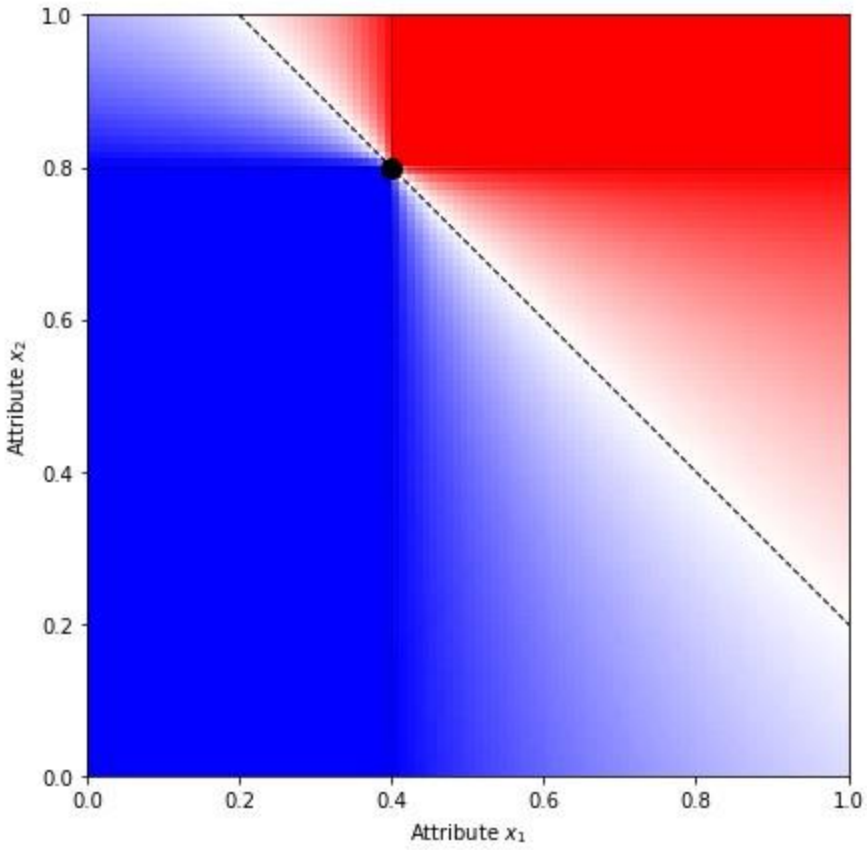


Figure 4.86: Two-attribute example when $x_1 = 0.4$ and $x_2 = 0.8$ for $U(0, 1)$

From Figure 4.87, it is possible to adjust the lower and upper limit of the uniform distribution such that different results can be generated. This example is for $x_1 = 0.3$ and $x_2 = 0.7$, while the lower limit is adjusted to 0.2 and the upper limit to 0.8 for the uniform distribution, i.e., $\boldsymbol{U}(0.2, 0.8)$. The region of uncertainty is larger compared to the region of uncertainty for the case where $x_1 = 0.2$, $x_2 = 0.6$, and $\boldsymbol{U}(0.4, 0.6)$ (Figure 4.84). This makes this simple program versatile as it is easy to change the coordinate values and the limits for the uniform distribution. An addition to this program would be to make it more flexible such that other distributions could be used as well.

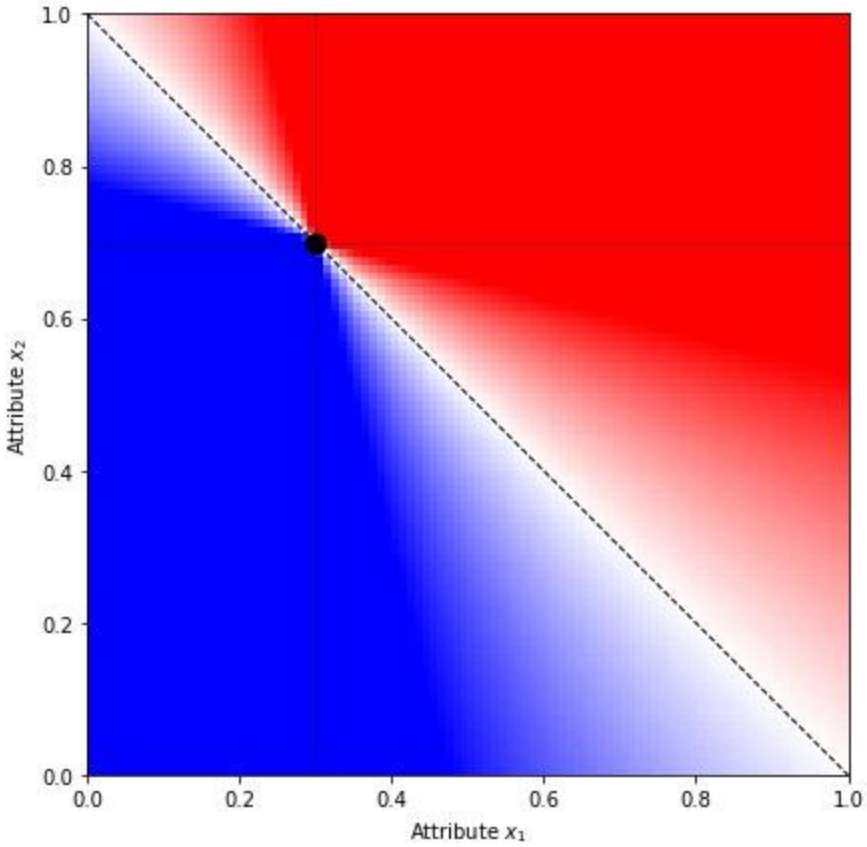


Figure 4.87: Two-attribute example when $x_1 = 0.3$ and $x_2 = 0.7$ for $U(0.2, 0.8)$

4.6 Discussion of results

As shown in previous sections in chapter 4, the ESP is a functioning DSS, and as such the first research goal described in section 1.3 has been achieved. DMs can conduct ES with it, while the ESP is able to remove dominated alternatives, simplifying the decision problem. The new users of the ESP receive a better understanding of how the ES method works, since an example is shown for them. This helps the new users to become familiar with the ES method, and rather focus on the decision problem. Furthermore, the ESP shows which alternatives are more likely to dominate another by having better ranks compared to the other alternatives. The ESP also shows which objectives have ranks that are close to being equal. This guarantees that the users receive a

better idea of swaps that are more likely to make an objective equivalent and an alternative dominated. In addition, the users are able to enter the data for the decision problem directly in the ESP, or loading in the data in the form of a CT or RT. This makes the ESP flexible, as users are not restricted by having to do it one way or the other.

In section 4.5, a simple example shows how the two-attribute example works and how it can be adjusted with different coordinate points and uniform limits within $\boldsymbol{U}(0,1)$. This two-attribute example shows how the probable dominance works. It is possible to further add to this by applying it to larger problems with more alternatives and attributes. Due to the complexity of the probable dominance functions shown in section 3.4.2, a significant effort has been put into developing the example in section 4.5. It is particularly challenging dealing with the preferences of DMs, as the preferences can be so varied, uncertain, and changed many times. In addition, it is difficult to simulate how the one-dimensional marginal value functions and the weights for the DM might be able to change and update during the process. This complexity caused time constraints that did not allow for incorporating the functions shown in section 4.5 into the ESP. As such the second research goal described in section 1.3 is thereby partly achieved.

The following paragraph describes how the probable dominance functions from the two-attribute example in section 4.5 could be incorporated into the ESP. The ESP could propose swaps that are aligned with the preferences of the DM by the DM answering the question of whether he prefers one alternative over another. The flow of the program could be changed such that instead of the user being asked what swaps he would like to make; the user would be asked whether he prefers one alternative over the other (Figure 4.88 and Figure 4.89). If yes, the ESP would make a swap in this direction (meaning this alternative is preferred over the other). Otherwise, the program would make a swap in the other direction. The ESP would then learn or capture this

preference of the user such that the ESP could propose similar swaps in the future. Eventually the ESP would make judgements on its own without any guidance from the user. This would mean the user would input his alternatives and objectives for the decision problem he is currently working on, and the ESP would remove dominated alternatives and make swaps based on previous problems the user has gone through. Another thing that could be incorporated into the ESP is the proposal of swaps that the DM can do based on the remaining alternatives after removing dominated alternatives similar to Smart-Swaps (Mustajoki and Hämäläinen, 2007).

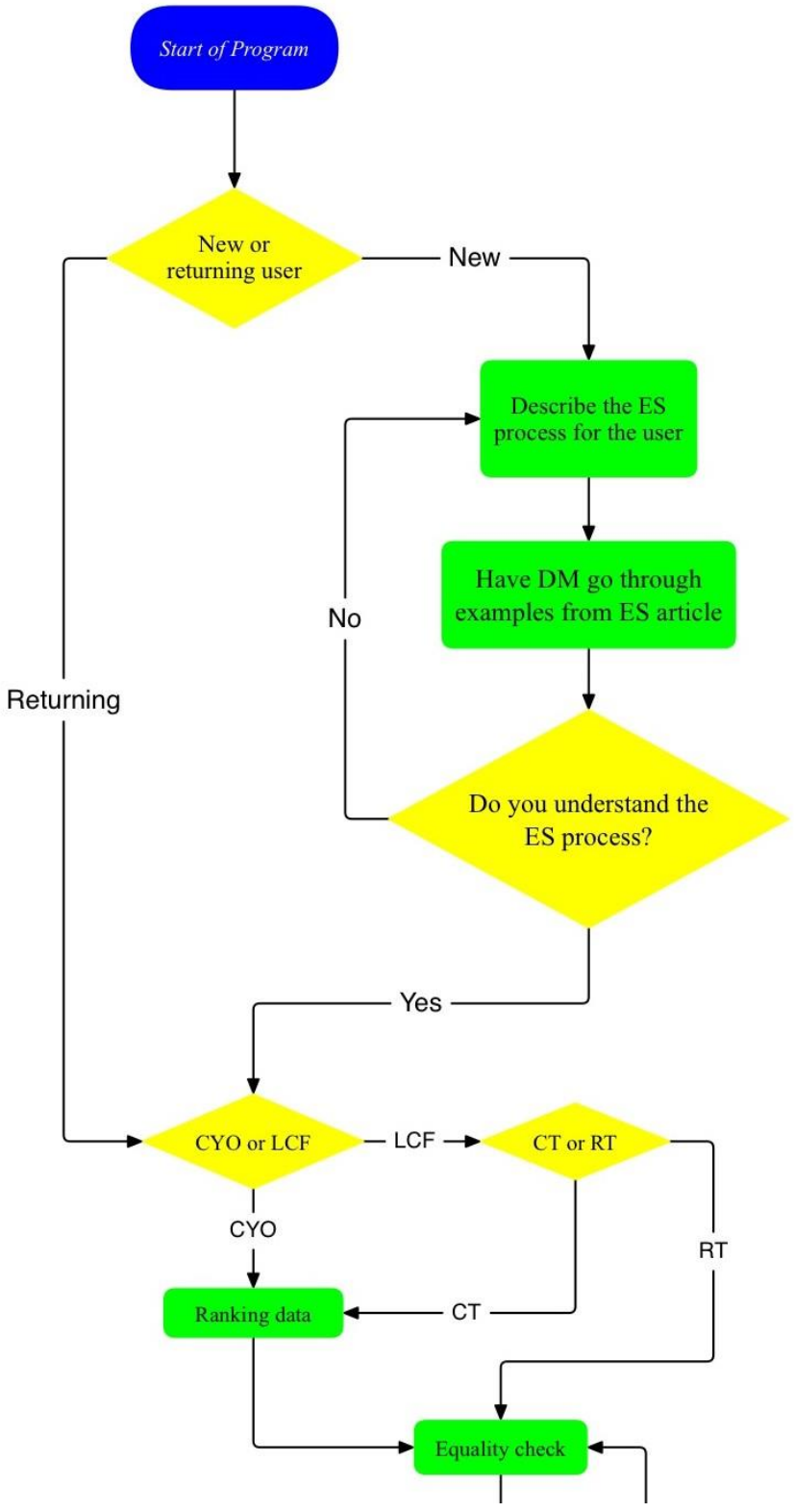


Figure 4.88: CFD part 1 of how the ESP could work in a future version

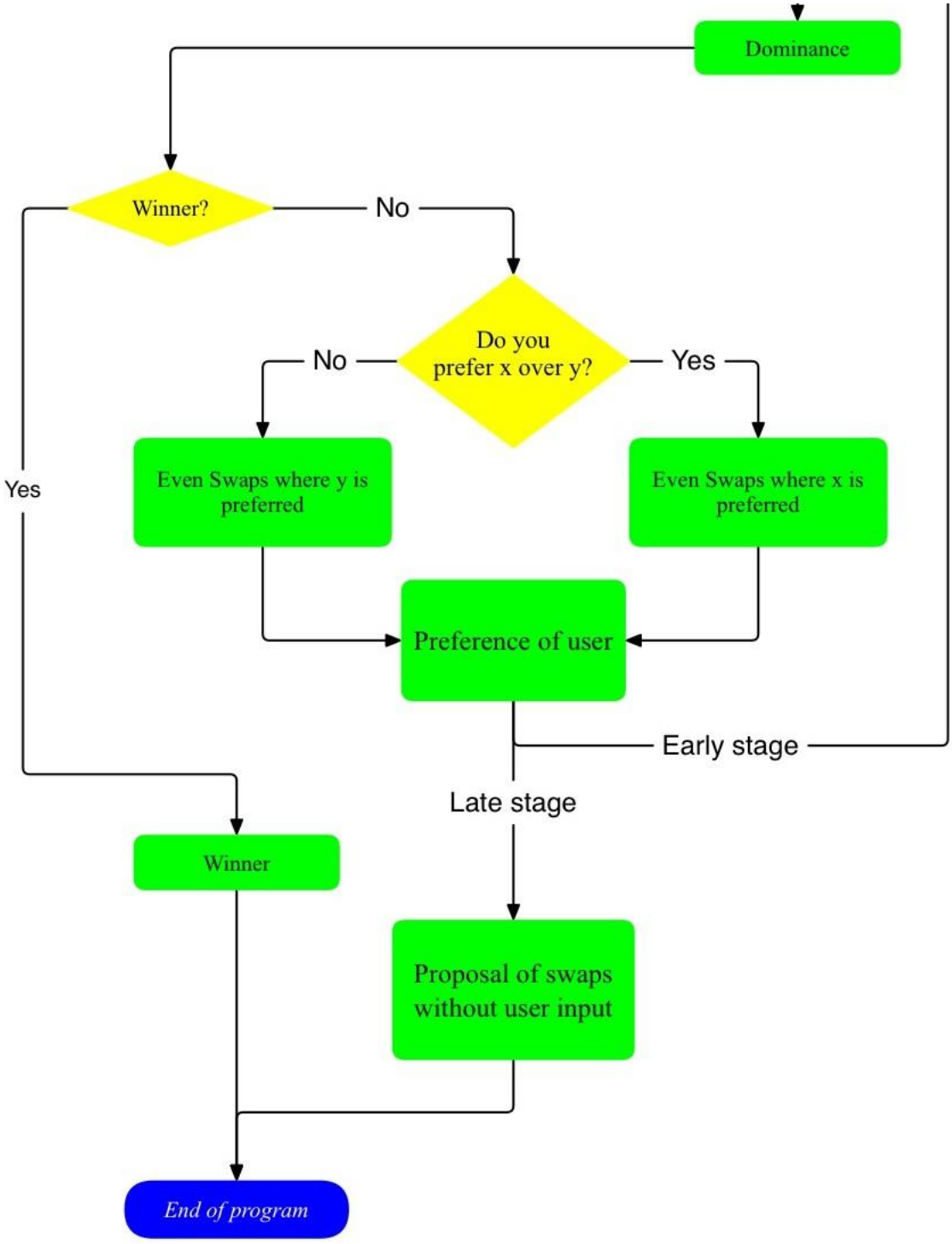


Figure 4.89: CFD part 2 of how the ESP could work in a future version

Considering what the current version of the ESP can achieve even without the probable dominance functions incorporated, the ESP is a valuable tool for DMs to make better decisions when using the ES method. It is similar to the Smart-Swaps program by Mustajoki and Hämäläinen (2005, 2007). There are some variations to it though such as that the ES are done on an RT instead of a CT, practical dominance is defined slightly differently, and there are no proposals for swaps that can be made. However, the ESP is still a viable substitution for the Smart-Swaps program, as the Smart-Swaps program was made in 2004 to be run in a browser by using Java (Mustajoki and Hämäläinen, 2004b). This makes Smart-Swaps old and outdated as it is more difficult to use Java with browsers today. Additionally, the user interface for Smart-Swaps is outdated. This makes the ESP a practical replacement for Smart-Swaps as the ESP is user-friendly and easy to run. For more details on Smart-Swaps, see section 3.3.

Looking at the modified ES by Dereli and Altun (2012; 2014), one can see that the ESP behaves differently to this, as it is designed to do. The modified ES is a modification of the ES method designed to be better suited for multi-issue negotiations. The ESP is designed to be a DSS tool to guide the DM through the ES method.

Furthermore, there are some limitations to the ESP as there are to any other program. Some of these limitations include:

- The user is only able to apply ES on rankings and not consequences.
- The ESP does not account for any other errors that may occur during the usage of this problem beyond the ones that are described in section 4.4.
- The LCF option of the ESP can only load in .xlsx files (Excel files), and no other type of files.

- The ESP does not work with data that is not in DF format or missing data.
- The ESP does not use probable dominance from Bhattacharjya and Kephart (2014) to deal with the preferences of the user.

4.7 Further improvements/additions to the ESP

It is always possible to improve any data program, whether that is through efficiency, user interactions, or other things such as a graphical user interface (GUI). As described in section 4.6, it is possible to add probable dominance to the ESP to improve the ESP's beliefs about the preferences of the DM ensuring the ESP ultimately would make judgements on its own without any guidance from the user.

It is also possible to make the ESP more user-friendly by making it into a GUI, similar to Figure 3.9 in section 3.3. This could make it easier for DMs to go through the process of making ES, as this process can be somewhat challenging to understand for new users in the current version of the ESP. However, experienced users of the ESP and similar programs should not have any problems with the current version.

Another improvement to the ESP could be to add hashing to the passwords created by users to strengthen the security of these. This would increase user-friendliness since the users then cannot access any data from other users.

In addition, the data could be stored in a database enabling the users access to previous decision problems they have gone through. This would make it easier for the user to see what swaps he has made for similar problems in the past. With this information, the user can make a better judgement on the current problem he is working on.

5 Conclusion

In this thesis, the even swaps method has been introduced and discussed in terms of how there has been made modifications to it after its introduction, how it has been supported by software and decision support system tools, and how it works. Decision support systems have been discussed in terms of how they can help decision makers make better decisions with a focus on particularly how decision support systems can guide the decision maker using the even swaps method. The main contribution of this thesis is the even swaps program, which makes it easier for decision makers to make improved multi-criteria decisions when using the even swaps method. This even swaps program checks for dominance (practical and absolute) by removing any dominated alternatives. It also helps the decision maker make swaps by showing which alternatives are more likely to dominate others, and the objectives with close to equal ranks. This makes it easier for decision makers to use the even swaps method, as the computational burden of having to check for dominance is reduced. Furthermore, alternatives that are likely to dominate another and objectives that have almost equal ranks are displayed to the user. With this even swaps program, decision makers can focus on the decision context they face and the swaps they need to make rather than how to deal with dominance. In addition, this even swaps program helps inexperienced users learn the even swaps method by showing useful examples and is still versatile enough that experienced users can benefit from using the program. The even swaps program allows the user to either create a consequences table and then a ranking table, or to load them from a file (either a consequences table or a ranking table).

An example of how probable dominance for decision makers in terms of the even swaps method can be implemented has been discussed. In this example, it is possible to see that different coordinate pairs and different limits for the uniform distribution makes it easier for a decision

maker to see if an alternative is likely to display absolute dominance over another. Furthermore, it displays uncertain regions (the probable dominance regions) that change when the limits for the uniform distribution over the DM's weights changes. It is possible to add to this probable dominance example by including different distributions over the weights such as a normal distribution or a binomial distribution.

Future potential improvements to the program include:

- Proposing swaps that are possible to make with the remaining alternatives and objectives such that an alternative would become dominated.
- Incorporating Bayesian Smart-Swaps such that probable dominance could help to capture and improve the even swaps program's beliefs about the preferences of the decision maker.
- Making a more user-friendly tool by developing a graphical user interface.

References

Altun, K., and T. Dereli, 2014, Even easier multi-issue negotiation through Modified Even-Swaps considering practically dominated alternatives: Computers & Industrial Engineering, v. 76, p. 307–317, doi:10.1016/j.cie.2014.08.015.

Altun, K., T. Dereli, and A. Baykasoğlu, 2013, Development of a framework for customer co-creation in NPD through multi-issue negotiation with issue trade-offs: FUZZYSS11: 2nd International Fuzzy Systems Symposium 17-18 November 2011, Ankara, Turkey, v. 40, no. 3, p. 873–880, doi:10.1016/j.eswa.2012.05.043.

Augustin, R. P., 1999, Making sense of site selection.: IIE Solutions, v. 31, no. 8, p. 34–36.

Belaid, A., and J. Razmak, 2013, Multi-Criteria Decision Support Systems: A glorious history and a promising future, *in* 2013 5th International Conference on Modeling, Simulation and Applied Optimization (ICMSAO): IEEE, p. 1–9, doi:10.1109/ICMSAO.2013.6552678.

Bhattacharjya, D., and J. Kephart, 2014, Bayesian interactive decision support for multi-attribute problems with even swaps: Uncertainty in Artificial Intelligence - Proceedings of the 30th Conference, UAI 2014, p. 72–81.

Dereli, T., and K. Altun, 2012, Modified Even-Swaps: A novel, clear, rational and an easy-to-use mechanism for multi-issue negotiation: Computers & Industrial Engineering, v. 63, no. 4, p. 1013–1029, doi:10.1016/j.cie.2012.06.013.

Dolan, J. G., 2010, Multi-Criteria Clinical Decision Support: A Primer on the Use of Multiple-Criteria Decision-Making Methods to Promote Evidence-Based, Patient-Centered Healthcare: The Patient, v. 3, no. 4, p. 229–248, doi:10.2165/11539470-000000000-00000.

Elahi, G., and E. Yu, 2012, Comparing alternatives for analyzing requirements trade-offs – In the absence of numerical data: Special Section: Engineering Complex Software Systems through Multi-Agent Systems and Simulation, v. 54, no. 6, p. 517–530, doi:10.1016/j.infsof.2011.10.007.

Elahi, G., and E. Yu, 2009, Trust Trade-off Analysis for Security Requirements Engineering, *in* 2009 17th IEEE International Requirements Engineering Conference, 2009 17th IEEE International Requirements Engineering Conference: IEEE, p. 243–248, doi:10.1109/RE.2009.12.

Gregory, R., and K. Wellman, 2001, Bringing stakeholder values into environmental policy choices: a community-based estuary case study: Ecological Economics, v. 39, no. 1, p. 37–52, doi:10.1016/S0921-8009(01)00214-2.

Hammond, J., R. Keeney, and H. Raiffa, 1998, Even Swaps: A Rational Method for Making Trade-offs: Harvard business review, v. 76, p. 137–143.

Hammond, J. S., R. L. Keeney, and H. Raiffa, 1999, Smart Choices A Practical Guide to Making Better Decisions, 1st ed. (revised): Harvard Business School Press, 244 p.

Hurley, W. J., and W. S. Andrews, 2003, Option Analysis: Using the method of even swaps: Canadian Military Journal, v. 4, no. 3, p. 43–46.

Jabbari, M., S. Sheikh, M. Rabiee, and A. Oztekin, 2022, A collaborative decision support system for multi-criteria automatic clustering: Decision Support Systems, v. 153, p. 113671, doi:10.1016/j.dss.2021.113671.

Kajanus, M., J. Ahola, M. Kurttila, and M. Pesonen, 2001, Application of even swaps for strategy selection in a rural enterprise: Management Decision, v. 39, no. 5, p. 394–402, doi:10.1108/00251740110395688.

Kask, S., C. Kline, and K. Lamoureux, 2011, Modeling tourist and community decision making: The SAVE Market: Annals of Tourism Research, v. 38, no. 4, p. 1387–1409, doi:10.1016/j.annals.2011.03.011.

Keser, B., 2005, An interactive approach for multi-criteria sorting problems, Master of Science: Middle East Technical University, Turkey, 109 p.

Lahtinen, T. J., and R. P. Hämäläinen, 2016, Path dependence and biases in the even swaps decision analysis method: European Journal of Operational Research, v. 249, no. 3, p. 890–898, doi:10.1016/j.ejor.2015.09.056.

Lahtinen, T. J., R. P. Hämäläinen, and C. Jenytin, 2020, On preference elicitation processes which mitigate the accumulation of biases in multi-criteria decision analysis: European Journal of Operational Research, v. 282, no. 1, p. 201–210, doi:10.1016/j.ejor.2019.09.004.

Li, H.-L., and L.-C. Ma, 2008, Visualizing decision process on spheres based on the even swap concept: I.T. and Value Creation, v. 45, no. 2, p. 354–367, doi:10.1016/j.dss.2008.01.004.

Luo, C., 2020, A comprehensive decision support approach for credit scoring: Industrial Management & Data Systems, v. 120, no. 2, p. 280–290, doi:10.1108/IMDS-03-2019-0182.

Luo, C.-M., 2008, Comparing Two Rational Decision-making Methods in the Process of Resignation Decision: The Journal of Human Resource and Adult Learning, v. 4, no. 1, p. 21–29.

Luo, C.-M., and C. Bor-Wen, 2006, Applying Even-Swap Method to Structurally Enhance the Process of Intuition Decision-Making: Systemic Practice and Action Research, v. 19, no. 1, p. 45–59, doi:10.1007/s11213-005-9003-3.

Ma, L.-C., and H.-L. Li, 2011, Using Gower Plots and Decision Balls to rank alternatives involving inconsistent preferences: Decision Support Systems, v. 51, no. 3, p. 712–719, doi:10.1016/j.dss.2011.04.004.

Milutinovic, G., U. Ahonen-Jonnarth, and S. Seipel, 2018, GISwaps: A new method for decision making in continuous choice models based on even swaps: International Journal of Decision Support System Technology, v. 10, no. 3, p. 57–78, doi:10.4018/IJDSST.2018070104.

Mustajoki, J., and R. P. Hämäläinen, 2005, A Preference Programming Approach to Make the Even Swaps Method Even Easier: Decision Analysis, v. 2, no. 2, p. 110–123.

Mustajoki, J., and R. P. Hämäläinen, 2004a, Making Even Swaps Even Easier: Systems Analysis Laboratory, Helsinki University of Technology, 25 p.

Mustajoki, J., and R. P. Hämäläinen, 2004b, Smart-Swaps: accessed June 1, 2022, http://smart-swaps.aalto.fi/.

Mustajoki, J., and R. P. Hämäläinen, 2007, Smart-Swaps — A decision support system for multicriteria decision analysis with the even swaps method: Decision Support Systems, v. 44, no. 1, p. 313–325, doi:10.1016/j.dss.2007.04.004.

Podinovski, V. V., 2016, Decision chains in analyzing multicriteria choice problems: Automatic Documentation and Mathematical Linguistics, v. 50, no. 6, p. 237–242, doi:10.3103/S0005105516060030.

Power, D. J., and R. Sharda, 2007, Model-driven decision support systems: Concepts and research directions: Integrated Decision Support, v. 43, no. 3, p. 1044–1061, doi:10.1016/j.dss.2005.05.030.

Qin, R., H. Liao, and L. Jiang, 2021, An enhanced even swaps method based on prospect theory with hesitant fuzzy linguistic information and its application to the selection of emergency logistics plans under the COVID-19 pandemic outbreak: Journal of the Operational Research Society, p. 1–13, doi:10.1080/01605682.2021.1897485.

Razmak, J., and B. Aouni, 2014, Decision Support System and Multi-Criteria Decision Aid: A State of the Art and Perspectives: Journal of Multi-Criteria Decision Analysis, v. 22, doi:10.1002/mcda.1530.

Shim, J. P., M. Warkentin, J. F. Courtney, D. J. Power, R. Sharda, and C. Carlsson, 2002, Past, present, and future of decision support technology: Decision Support System: Directions for the Nest Decade, v. 33, no. 2, p. 111–126, doi:10.1016/S0167-9236(01)00139-7.

Turban, E., J. E. Aronson, and T.-P. Liang, 2005, Decision Support Systems and Intelligent Systems, 7th ed. (revised): Prentice-Hall, Inc.

Wachowicz, T., 2010, DECISION SUPPORT IN SOFTWARE SUPPORTED NEGOTIATIONS.: DERYBŲ SPRENDIMŲ PARAMOS PROGRAMA., v. 11, no. 4, p. 576–597.

Wachowicz, T., 2007, Even Swaps Method for Developing Assessment Capabilities of E-Negotiation System, *in* Proceedings of the International Multiconference on Computer Science and Information Technology: p. 597–606.

Appendices

Appendix A

A1 Installation Guide

The installation steps for the ESP are as follows:

1. Download and install Anaconda Navigator from here:

   https://docs.anaconda.com/anaconda/navigator/install/

   a. Follow the steps to install Jupyter Notebook
2. Download and install the necessary packages:

   a. NumPy: https://numpy.org/install/

   b. Pandas: https://pandas.pydata.org/getting_started.html

   c. Matplotlib: https://matplotlib.org/stable/users/getting_started/

   d. SciPy: https://docs.scipy.org/doc/scipy/getting_started.html

      i. Matplotlib and SciPy are not needed for the ESP, but they are necessary for the probable dominance example program
3. Download and unzip the ESP and the necessary files into a suitable folder from:

   https://github.com/tsandb1/Even-Swaps-Program

A2 User Manual

The following are the steps to use the ES program:

1. Open the ESP in Jupyter Notebook

   a. Make sure the program and the files are in the same folder

   b. The files include:

i. Example Consequence and Ranking Table.xlsx
ii. Test DF Miller 1.xlsx
iii. Test RT Sahid.xlsx
iv. Test RT Sahid 2.xlsx

2. Go to Kernel → Restart & Run All or Cell → Run All
3. Create username and password
4. Either create a CT or an RT in the program or load in a CT or an RT in xlsx format

The following are the steps to use the simple two-attribute example program:

1. Open the program in Jupyter Notebook
2. Go to Kernel → Restart & Run All or Cell → Run All
3. Follow the prompts to select a value for $x_1(a_1)$ and $x_2(a_1)$, number of steps for how smooth the probable dominance should be, and lower and upper limit for uniform distribution

Appendix B

B1 ESP complete code

In this appendix is the complete code for the ESP.

```
[ ]:
1 import numpy as np
2 import pandas as pd
3 import warnings
4 import colorama
5 import json
```

```
[ ]:
1 warnings.filterwarnings('ignore')
```

Appendix B Figure 1: Importing libraries and ignoring warnings

```
1  def error_messages (error_msg):
2      """
3      Function to print error messages in bold
4
5      Parameters
6      __________
7      error_msg: error message to be printed
8
9      Returns
10     _______
11     None
12     """
13     #For making text bold
14     bold_start = "\033[1m"
15     bold_end = "\033[0m"
16     print(bold_start + error_msg + bold_end)
```

Appendix B Figure 2: Error messages function

```python
def ranking_table (df):
    """
    Function to rank a consquences table
    User is asked to input rank for qualitative (non-numeric) objectives
    User is asked to input index for the ranks (which row the data is from)
    User is asked whether the ranking for each row is ascending or not

    Parameters
    __________
    df: dataframe to do the rankings on

    Returns
    _______
    df_ranked: ranked dataframe
    """

    print('\nConsequences table to be ranked')
    display(df)

    flag = False #For exiting program if user does not input number for the ranks

    dfwo = df.loc[:, df.columns[1:]] #df without objectives
    dfwo_copy = dfwo.copy()
    df_obj = df.iloc[:,[0]] #Objectives column

    rank_data = []

    for i in dfwo:
        df_str = dfwo[dfwo[i].apply(lambda x: isinstance(x, str))] #Checks whether df contains strings

    len_df_str = len(df_str.index) #Length of the index for df_str

    if df_str.empty:
        print('\nThere is no string objectives in the dataframe')

    else:
        print('\nDataframe with only string objectives')
        display(df_str)
```

Appendix B Figure 3: Ranking table function part 1

```
        j = 0
        while j < len(df_str):
            for rowIndex, row in df_str.iterrows():
                    for columnIndex, value in row.items():
                        try:
                            input_rank = int(input('\nWhat would you like the rank for {} to be? '.format(value)))
                            flag = True
                        except ValueError:
                            value_error_msg = '\nError! That is not an Int number. Please enter an Int number\n'
                            error_messages(value_error_msg)

                        if not flag:
                            return

                        if (input_rank <= 0):
                            error_msg1 = '\nError! Your chosen rank is less than or'
                            error_msg2 = ' equal to zero, but it has to be positive\n'
                            error_msg_whole = error_msg1 + error_msg2
                            error_messages(error_msg_whole)
                            return

                        rank_data.append(input_rank)
                    print()
                    j += 1

        splits = np.array_split(rank_data, len_df_str) #Split rank_data, based on len of index
        print('\nRanking data', rank_data)
        print('\nIt is important to remember that the position of the data should be')
        print('corresponding with the position of the data from the original CT')

        flag2 = False #For exiting program if user does not input number for the index position of where the ranks belong
        g = 0
        while g < len(splits):
            split_ranks = splits[g]
            try:
                input_rank_pos = int(input('\nWhich position in the dataframe is the data {} from? '.format(split_ranks)))
                dfwo_copy.at[input_rank_pos] = split_ranks
                flag2 = True
            except ValueError:
                value_error_msg = '\nError! That is not an Int number. Please enter an Int number\n'
                error_messages(value_error_msg)
            if not flag2:
                return
```

Appendix B Figure 4: Ranking table function part 2

```
83
84              if (input_rank_pos < 0):
85                  error_msg = '\nError! Your chosen index position is negative, but it has to be zero or greater\n'
86                  error_messages(error_msg)
87                  return
88
89              g += 1
90
91      print('\nDataframe with only numbers')
92      display(dfwo_copy)
93      print('******************')
94
95      row_data = []
96
97      for r in range(len(dfwo_copy)):
98          row = dfwo_copy.loc[r]
99          row_vals = row.values
100         input_ascending = input('\nWould you like the ranking for {0} with index {1}'.format(row_vals, r) +
101                                 ' to be Ascending or Not? A or N: ')
102
103         if input_ascending.lower() in {'a'}:
104             df_ranked_a = row.rank(method = 'min', ascending = True).astype(int)
105
106         elif input_ascending.lower() in {'n'}:
107             df_ranked_a = row.rank(method = 'min', ascending = False).astype(int)
108         else:
109             error_msg = 'Error! Options are A or N'
110             error_messages(error_msg)
111             break
112
113         row_data.append(df_ranked_a)
114
115     df_ranked = pd.DataFrame(row_data)
116     df_ranked = pd.concat([df_obj, df_ranked], axis = 1)
117
118     print('\nRanked dataframe')
119     display(df_ranked)
120
121     return df_ranked
```

Appendix B Figure 5: Ranking table function part 3

```
def cyo ():
    """
    Function for the CYO option in the user_choice(input_decision) function where the user
    is asked to create his/her own consequences table (CT)

    Parameters
    __________
    None

    Returns
    _______
    df: dataframe in CT format to be ranked
    """

    user_statement = """

    How to use the program to create consequences table
    1. Enter a number of objectives as data rows, e.g. 5
    2. Enter a number of alternatives, e.g. 5
    3. Enter header for alternatives like this: Alt1, Alt2 until however many alternatives you have
    4. Enter the name for the first objective, then the next until you have done so for all objectives
    5. Enter data for alternatives like this: 1700 1800

    Note: For step 5, it is important to remember that you enter data for alternatives
    like this: data for alt 1 data for alt 2, and so on
    """

    print(user_statement)

    try:
        num_obj = int(input("\nEnter number of objectives (data rows): "))
        num_alts = int(input('\nEnter number of alternatives: '))

    except ValueError:
        value_error_msg = '\nError! That is not an Int number. Please enter an Int number\n'
        error_messages(value_error_msg)
        return
```

Appendix B Figure 6: CYO function part 1

```
    obj_header = 'Objectives'

    alternatives_header = [y for y in input('\nEnter header for alternatives separated by comma: ').split(', ')]

    data_objectives = []
    data_alternatives = []

    p = 1
    while p <= num_obj:
        row_objectives = input('\nEnter name for objective {}: '.format(p))

        if row_objectives in data_objectives:
            error_msg = '\nThat objective name already exists'
            error_messages(error_msg)
            row_objectives = input('\nEnter a new name for objective {}: '.format(p))

        data_objectives.append(row_objectives)
        p += 1

    q = 1

    input_text = '\nWhat type would you like to use for {}? Float, Int, or String (F, I, or S): '
    while q <= num_obj:
        input_type = input(input_text.format(data_objectives[q - 1]))

        if input_type.lower() in {'f'}:
            row_alternatives = [float(x) for x in input('\nEnter data for alternatives' +
                                ' for objective {} separated by space: '.format(data_objectives[q - 1])).split()]
        elif input_type.lower() in {'i'}:
            row_alternatives = [int(x) for x in input('\nEnter data for alternatives' +
                                ' for objective {} separated by space: '.format(data_objectives[q - 1])).split()]
        elif input_type.lower() in {'s'}:
            row_alternatives = [x for x in input('\nEnter data for alternatives' +
                                ' for objective {} separated by space: '.format(data_objectives[q - 1])).split()]
        else:
            error_msg = '\nError! Options are F, I, or S\n'
            error_messages(error_msg)
            return
```

Appendix B Figure 7: CYO function part 2

```
        if len(row_alternatives) > num_alts:
            error_msg1 = ('\nError! No more than') + str(num_alts)
            error_msg2 = (' alternatives are allowed')
            error_msg_whole = error_msg1 + error_msg2
            error_messages(error_msg_whole)
            break

        data_alternatives.append(row_alternatives)
        q += 1

    df1 = pd.DataFrame(data_objectives, columns = [obj_header])
    df2 = pd.DataFrame(data_alternatives, columns = alternatives_header)

    con_df = pd.concat([df1, df2], axis = 1) #Consequences DF
    con_df.index = range(len(con_df.index))

    df = ranking_table(con_df) #Ranking consequences table

    return df #Returning ranking table of created consequences table
```

Appendix B Figure 8: CYO function part 3

```
def lcf ():
    """
    Function for the LCF option in the user_choice(input_decision) function where the user
    is asked to load in a consequences table (CT) or a ranking table (RT)

    Parameters
    __________
    None

    Returns
    _______
    df: dataframe that is either in CT or RT format
    """

    flag = True #To exit program if file is not found
    flag2 = True #To exit program if user enters quit

    while flag2:
        try:
            file_decision = input('\nWould you like to load a consequences table' +
                                  ' or a ranking table (CT or RT)? Enter quit to quit: ')
            if file_decision.lower() == 'Quit':
                flag2 = False

        except EOFError:
            print('\nProgram Quit')
            break

        if (file_decision.lower() == 'rt'):
            try:
                filename = input('\nWhat is the name of the file you would like to load? Enter quit to quit: ')

                if (filename.lower() == 'quit'):
                    print('\nProgram Quit')
                    flag = False

                else:
                    filename = filename + '.xlsx'
                    df = load_files(filename)
                    print('\nDisplaying loaded dataframe')
                    display(df)
                    return df #Returning loaded RT dataframe
```

Appendix B Figure 9: LCF function part 1

```
            except FileNotFoundError:
                error_msg1 = ('\nError! ') + str(filename)
                error_msg2 = (' does not exist\n')
                error_msg_whole = error_msg1 + error_msg2
                error_messages(error_msg_whole)

        elif (file_decision.lower() == 'ct'):
            try:
                filename = input('\nWhat is the name of the file you would like to load? Enter quit to quit: ')

                if (filename.lower() == 'quit'):
                    print('\nProgram Quit')
                    flag = False

                else:
                    filename = filename + '.xlsx'
                    loaded_df = load_files(filename)
                    df = ranking_table(loaded_df)
                    return df #Returning loaded ranked CT dataframe

            except FileNotFoundError:
                error_msg1 = ('\nError! ') + str(filename)
                error_msg2 = (' does not exist\n')
                error_msg_whole = error_msg1 + error_msg2
                error_messages(error_msg_whole)

        elif (file_decision.lower() == 'quit'):
            print('\nProgram Quit')
            flag = False

        else:
            error_msg1 = ('\nError! ') + str(file_decision)
            error_msg2 = (' is not a valid option\n')
            error_msg_whole = error_msg1 + error_msg2
            error_messages(error_msg_whole)
            print('Options are CT or RT')

        if not flag:
            return
```

Appendix B Figure 10: LCF function part 2

```
def user_choice (input_decision):
    """
    Function that lets the user create his/her own consequences table (CT) or load a previously created xlsx file
    by using the functions cyo and lcf, respectively

    Parameters
    _________
    input_decision: Whether the user would like to create his/her own CT or load a previously created xlsx file

    Returns
    _______
    df: Dataframe to be used for even swaps
    """

    if (input_decision == 'CYO'): #Create Your Own
        df = cyo()
        return df

    elif (input_decision == 'LCF'): #Load Created File
        df = lcf()
        return df

    elif (input_decision.lower() == 'quit'):
            raise EOFError

    else:
        error_msg1 = ('\nError! ') + str(input_decision)
        error_msg2 = (' is not a valid option\n')
        error_msg_whole = error_msg1 + error_msg2
        error_messages(error_msg_whole)
        print('Options are CYO, or LCF')
        return
```

Appendix B Figure 11: User choice function

```
def equality_check (df):
    """
    Function to check if a dataframe row has equal values

    Parameters
    __________
    df: dataframe to do operations on

    Returns
    _______
    df with unique rows
    """

    df = df[df.nunique(axis = 1).ne(1)]
    return df
```

Appendix B Figure 12: Equality check function

```
def compare_alternatives (alt1, alt2):
    """
    Function that compares alternatives (columns) in a dataframe
    based on rank

    Parameters

    _________
    alt1: dataframe column for first alternative
    alt2: dataframe column for second alternative

    Returns

    _______
    dominated: loser between the alternatives, which alternative is being dominated by another
    dominating: winner amongst the alternatives, which alternative is dominating the other
    dominance_type: what type of dominance it is (practical or absolute)
    """

    alt1_wins = 0
    alt2_wins = 0

    dominated = 'none'
    dominating = 'none'
    dominance_type = 'no dominance'

    #Checks if alternative name is different from itself,
    #since there is no reason to compare an alternative with itself
    if (alt1.name != alt2.name):
        for row in range(0, len(alt1)):

            if (int(alt1[row]) < int(alt2[row])): #Updates alt 1 wins
                alt1_wins += 1

            elif (int(alt1[row]) > int(alt2[row])): #Updates alt 2 wins
                alt2_wins += 1

        if (alt1_wins > alt2_wins & alt2_wins < 2): #Alt 1 dominates alt 2
            dominated = alt2.name
            dominating = alt1.name

            if (alt2_wins == 1):
                dominance_type = 'practical'
            elif (alt2_wins == 0):
                dominance_type = 'absolute'
```

Appendix B Figure 13: Compare alternatives function part 1

```
        elif (alt2_wins > alt1_wins & alt1_wins < 2): #Alt 2 dominates alt 1
            dominated = alt1.name
            dominating = alt2.name

            if (alt1_wins == 1):
                dominance_type = 'practical'
            elif (alt1_wins == 0):
                dominance_type = 'absolute'

    return dominance_type, dominated, dominating
```

Appendix B Figure 14: Compare alternatives function part 2

```
def almost_equal (df):
    """
    Function that checks whether the ranks of an alternative is one rank apart from another alternative
    e.g. rank of alt 1 is 1 and rank of alt 2 is 2 for same objective,
    would return these alternatives with the ranks for that objective

    Parameters
    __________
    df: dataframe to do operations on

    Returns
    _______
    None
    """

    dfwo = df.loc[:, df.columns[1:]] #df without objectives
    df_obj = df.iloc[:,0] # Dataframe objectives
    obj_index = df_obj.index.values
    difference = []

    for i in dfwo.columns:
        for j in dfwo.columns:
            alt1 = dfwo[i]
            alt2 = dfwo[j]
            name1 = alt1.name
            name2 = alt2.name

            if (name1 != name2):
                for count, (k, l) in enumerate(zip(alt1, alt2)):
                    diff = abs(k - l)
                    difference.append(diff)
                    if diff == 1:
                        if name1 < name2:
                            if count in obj_index:
                                obj_name = df_obj.at[count]
                                msg = ('Objective ' + obj_name + ' for '
                                       + name1 + ' and ' + name2 + ' is almost equal with these rankings: '
                                       + str(k) + ', ' + str(l))
                                print(msg)

    if 1 not in difference:
        print('There are no ranks that are almost equal')
```

Appendix B Figure 15: Almost equal function

```
def count_num_objectives (alt1, alt2):
    """
    Function that counts the number of objectives that dominate another alternative

    Parameters
    _________
    alt1: dataframe column for first alternative
    alt2: dataframe column for second alternative

    Returns
    _______
    None
    """

    alt1_wins = 0
    alt2_wins = 0

    dominated = 'none'
    dominating = 'none'

    #Checks if alternative name is different from itself,
    #since there is no reason to compare an alternative with itself
    if(alt1.name != alt2.name):
        for row in range(0, len(alt1)):

            if (int(alt1[row]) < int(alt2[row])): #Updates alt 1 wins
                alt1_wins += 1

            elif (int(alt1[row]) > int(alt2[row])): #Updates alt 2 wins
                alt2_wins += 1

        if (alt1_wins > alt2_wins): #Alt 1 dominates alt 2
            dominated = alt2.name
            dominating = alt1.name
            print(dominating + ' is more likely to dominate ' + dominated)

        elif (alt1_wins == alt2_wins):
            if alt1.name > alt2.name:
                print('Neither ' + alt1.name + ' nor ' + alt2.name + ' is more likely to dominate the other')
```

Appendix B Figure 16: Count Num objectives function

```
def df_obj_count (df):
    """
    Function that uses the function count_num_objectives(alt1, alt2)
    to count how many objectives have a higher rank for alt1 compared with alt2

    Parameters
    __________
    df: dataframe to do operations on

    Returns
    _______
    None
    """
    for i in df.columns:
        for j in df.columns:
            count_num_objectives(df[i], df[j])
```

Appendix B Figure 17: DF obj count function

```
def max_rank (df):
    """
    Function that checks what the max rank is in a ranking table (RT)

    Parameters
    __________
    df: dataframe to do operations on

    Returns
    _______
    df_max_value: max rank in RT
    """

    dfwo = df.loc[:, df.columns[1:]] #Dataframe without objectives
    df_max_value = dfwo.max().max()
    return df_max_value
```

Appendix B Figure 18: Max rank function

```
def even_swaps (df, max_df):
    """
    Function that does even swaps on a dataframe
    The user inputs which objective and alternative he/she would like to make a swap on,
    then he/she inputs an objective to make a compensation on

    Parameters
    _________
    df: dataframe
    max_df: max rank in dataframe before dropping dominated alternatives

    Returns
    _______
    None
    """

    print('\nThe first step is to choose an alternative')

    display(df)

    alternatives = df.columns[1:].values

    print('\nOptions are', alternatives)

    print('\nBelow are the alternative(s) that are likely to dominate another alternative')
    print('by having more objectives dominating the other alternative if any\n')

    dfwo = df.loc[:, df.columns[1:]] #Dataframe without objectives for counting ranks
    df_obj_count(dfwo)

    print('\nIt is better to make swaps on an alternative that is more likely to be dominated')
    print('\nBelow are the ranks that are almost equal if any\n')

    almost_equal(df)

    print('\nIt is better to make swaps on objectives where the ranks are almost equal')

    input_alt_text1 = '\nKnowing this information, which alternative would you'
    input_alt_text2 = ' like to make an even swap on? Type quit to exit program: '
    input_alt_text = input_alt_text1 + input_alt_text2
    input_alt = input(input_alt_text)
```

Appendix B Figure 19: Even swaps function part 1

```
    if (input_alt in alternatives):

        print('\nYou have chosen', input_alt)
        print('\nThe second step is to choose which objective you would like to make a swap on')

        objectives = df.iloc[:,0].values #Names of dataframe objectives
        print('\nOptions are', objectives)

        df_obj = df.iloc[:,0] # Dataframe objectives

        input_obj = input('\nChoose an objective to do an even swap on or type quit to exit program: ')
        obj_in_df = df_obj.isin([input_obj]).any() #Objective is in dataframe

        if (obj_in_df == True):
            chosen_obj = df.loc[df_obj == input_obj]
            display(chosen_obj)

            alt_value = chosen_obj[input_alt] #Alternative ranking value
            alt_value = int(alt_value.to_string(index = False))

            print('\nThe value for', input_obj, 'and', input_alt, 'is', alt_value)
            print('\nThe third step is to choose a new value for the objective and alternative')

            while True:
                try:
                    input_swap = int(input('\nWhat would you like the value to be changed to? '))

                except ValueError:
                    value_error_msg = '\nError! That is not an Int number. Please enter an Int number\n'
                    error_messages(value_error_msg)
                    continue
                else:
                    break

            print('\nYour chosen value is', input_swap)

            if (input_swap <= 0):
                error_msg1 = ('\nError! Your chosen value')
                error_msg2 = (' is less than or equal to zero, but it has to be positive for the even swap\n')
                error_msg_whole = error_msg1 + error_msg2
                error_messages(error_msg_whole)

            elif (input_swap == alt_value):
                error_msg1 = ('\nError! Your input value cannot be the same as what the value for')
                error_msg2 = (' the alternative already is, which is ') + str(alt_value)
                error_msg_whole = error_msg1 + error_msg2
                error_messages(error_msg_whole)
```

Appendix B Figure 20: Even swaps function part 2

```
            elif (input_swap > max_df):
                error_msg1 = ('\nError! Your input value cannot be more than what the max value')
                error_msg2 = (' in the dataframe is, which is ') + str(max_df)
                error_msg_whole = error_msg1 + error_msg2
                error_messages(error_msg_whole)

            else:
                df.loc[df_obj == input_obj, input_alt] = input_swap

                rem_obj = [x for x in objectives if x != input_obj] #Remaining objectives
                df_rem_obj = df.copy()
                df_rem_obj = df_rem_obj[df_rem_obj['Objectives'] != input_obj]

                if (len(objectives) == 2):
                    oth_obj = [x for x in rem_obj if x != input_obj] #Other objective
                    compensated_obj = oth_obj[0]
                else:
                    print('\nThe next step is to choose an objective to compensate for')
                    display(df_rem_obj)
                    print('\nOptions are', rem_obj)

                    compensated_obj = input('\nWhich objective would you like to compensate for?' +
                                            ' Type quit to exit program: ')

                comp_obj_in_df = df_obj.isin([compensated_obj]).any() #Compensated objective is in dataframe

                if (input_obj == compensated_obj):
                    error_msg1 = ('\nError! Your chosen compensated objective cannot be the same')
                    error_msg2 = (' as the objective you would like to compensate for\n')
                    error_msg_whole = error_msg1 + error_msg2
                    error_messages(error_msg_whole)
                    df.loc[df_obj == input_obj, input_alt] = alt_value

                elif (comp_obj_in_df == True):
                    chosen_comp_obj = df.loc[df_obj == compensated_obj]
                    display(chosen_comp_obj)

                    alt_value_comp = chosen_comp_obj[input_alt]
                    alt_value_comp = int(alt_value_comp.to_string(index = False))

                    print('\nThe value for the compensated objective:', compensated_obj, end = ' ')
                    print('and', input_alt, 'is', alt_value_comp)
                    print('\nThe next step is to choose a new compensated value for the objective and alternative')
```

Appendix B Figure 21: Even swaps function part 3

```
                    while True:
                        try:
                            compensated_value = int(input('\nWhat would you like the compensated value to be? '))

                        except ValueError:
                            value_error_msg = '\nError! That is not an Int number. Please enter an Int number\n'
                            error_messages(value_error_msg)
                            df.loc[df_obj == input_obj, input_alt] = alt_value
                            continue
                        else:
                            break

                    print('\nYour chosen compensated value is', compensated_value)

                    if (compensated_value <= 0):
                        error_msg1 = ('\nError! Your chosen compensated value is less than or equal to zero,')
                        error_msg2 = (' but the compensated value has to be positive for the even swap\n')
                        error_msg_whole = error_msg1 + error_msg2
                        error_messages(error_msg_whole)
                        df.loc[df_obj == input_obj, input_alt] = alt_value

                    elif (compensated_value == alt_value_comp):
                        error_msg1 = ('\nError! Your input value cannot be the same as what the value for')
                        error_msg2 = (' the alternative already is, which is ') + str(alt_value_comp)
                        error_msg_whole = error_msg1 + error_msg2
                        error_messages(error_msg_whole)
                        df.loc[df_obj == input_obj, input_alt] = alt_value

                    elif (compensated_value > max_df):
                        error_msg1 = ('\nError! Your compensated value cannot be more than what the max value')
                        error_msg2 = (' in the dataframe is, which is ') + str(max_df)
                        error_msg_whole = error_msg1 + error_msg2
                        error_messages(error_msg_whole)
                        df.loc[df_obj == input_obj, input_alt] = alt_value

                    else:
                        df.loc[df_obj == compensated_obj, input_alt] = compensated_value

                    print('\nDataframe after doing even swaps operations on it\n')
                    display(df)
                    print('\n****************\n')

                elif (compensated_obj.lower() == 'quit'):
                    raise EOFError
```

Appendix B Figure 22: Even swaps function part 4

```
179
180                 else:
181                     error_msg1 = ('\nError! ') + str(compensated_obj)
182                     error_msg2 = (' is not part of the list of objectives\n')
183                     error_msg_whole = error_msg1 + error_msg2
184                     error_messages(error_msg_whole)
185                     print('Options are', rem_obj, '\n')
186                     df.loc[df_obj == input_obj, input_alt] = alt_value
187
188         elif (input_obj.lower() == 'quit'):
189             raise EOFError
190
191         else:
192             error_msg1 = ('\nError! ') + str(input_obj)
193             error_msg2 = (' is not part of the list of objectives\n')
194             error_msg_whole = error_msg1 + error_msg2
195             error_messages(error_msg_whole)
196             print('Options are', objectives, '\n')
197
198     elif (input_alt.lower() == 'quit'):
199         raise EOFError
200
201     else:
202         error_msg1 = ('\nError! ') + str(input_alt)
203         error_msg2 = (' is not a valid name\n')
204         error_msg_whole = error_msg1 + error_msg2
205         error_messages(error_msg_whole)
206         print('Options are', alternatives, '\n')
```

Appendix B Figure 23: Even swaps function part 5

```
def dataframe_reduction (df):
    """
    Function that reduces a dataframe consisting of alternatives
    Uses the function compare_alternatives(alt1, alt2) to check for dominance
    If an alternative dominates another, the alternative that is
    being dominated is removed from the dataframe

    Parameters
    __________
    df: dataframe to do the operations on

    Returns
    _______
    loser_count: counter to count how many losers there are
    """

    popped_obj = df.pop('Objectives')

    max_i = len(df.columns) - 1
    max_j = len(df.columns) - 1

    i = 0
    loser_count = 0

    while i <= max_i:

        j = 0
        alt1 = df.loc[:,df.columns[i]]

        while j <= max_j:

            alt2 = df.loc[:,df.columns[j]]
            dom_type, loser, winner = compare_alternatives(alt1, alt2)

            while loser in df:

                if (loser != 'none'):

                    df.drop(loser, axis = 1, inplace = True)
                    loser_count += 1

                    print()
                    print(loser, 'has been dropped from dataframe by the use of', dom_type, 'dominance.', end = ' ')
                    print('It is being dominated by', winner)
```

Appendix B Figure 24: Dataframe reduction function part 1

```
45
46                      if len(df.columns) > 1:
47                          print('\nDisplaying new dataframe')
48                          display(df)
49
50                      if (loser == alt1.name):
51                          max_i = len(df.columns) - 1
52                          max_j = len(df.columns) - 1
53                          j = max_j
54
55                      else:
56                          max_j -= 1
57                          max_i -= 1
58              else:
59                  j += 1
60
61          if (loser != alt1.name):
62              i += 1
63
64      df.insert(0, 'Objectives', popped_obj)
65
66      return loser_count
```

Appendix B Figure 25: Dataframe reduction function part 2

```
def running_even_swaps (df):
    """
    Reduces the dataframe when alternatives
    are being dominated by using the function dataframe_reduction(df)
    Conducts even swaps to remove objectives

    Parameters
    __________
    df: dataframe to do the operations on

    Returns
    _______
    Returns if there are NaNs in the dataframe
    df: final dataframe after checking for dominance and conducting even swaps on it
    """

    rules = """
The following are the rules for using the program from this stage onwards:\n
1. Check for equal rank for the objectives, if True drops these objectives
2. Checks dominance, if alternative is being dominated (practical or absolute) it is dropped
3. User is asked to conduct even swaps on remaining alternatives and objectives
4. User selects one alternative he/she would like to make a swap on to make ranks equal
5. User is asked to compensate for this swap on another objective on same alternative
6. Final alternative should be the decision for the user
7. The user may quit on entering the even swaps part of the program by typing 'Quit' where prompted to
    """

    if df is None:
        print('\nThere is no dataframe to do even swaps on')

    else:
        print(rules)
        max_length = len(df.columns[1:])
        loser_count = 0

        while True:
            try:
                while max_length > 1:
                    popped_obj = df.pop('Objectives')
                    print('\nDataframe before removing equal rows')
                    display(df)

                    isNan = df.isnull().values.any()
                    df = equality_check(df)
                    df.insert(0, "Objectives", popped_obj)
                    df.reset_index(drop = True, inplace = True)
```

Appendix B Figure 26: Running even swaps function part 1

```
                if isNan == True:
                    error_msg1 = ('\nError! Not possible to do even swaps as there are or could be')
                    error_msg2 = (' NaNs in the dataframe. There is no final alternative')
                    error_msg_whole = error_msg1 + error_msg2
                    error_messages(error_msg_whole)
                    return
                else:
                    print('\nDataframe after removing equal rows')
                    display(df)

                    max_df = max_rank(df) #Max rank in DF before dominated alternatives are removed
                    loser_count = dataframe_reduction(df)
                    max_length -= loser_count

                    if (max_length > 1):
                        popped_obj = df.pop('Objectives')
                        print('\nDataframe before equal rows have been removed')
                        display(df)

                        df = equality_check(df)
                        df.insert(0, "Objectives", popped_obj)
                        df.reset_index(drop = True, inplace = True)
                        if isNan == True:
                            error_msg1 = ('\nError! Not possible to do even swaps as there are or could be')
                            error_msg2 = (' NaNs in the dataframe. There is no final alternative')
                            error_msg_whole = error_msg1 + error_msg2
                            error_messages(error_msg_whole)
                            return
                        else:
                            print('\nDataframe after equal rows have been removed')
                            display(df)
                            even_swaps(df, max_df)
            return df

        except EOFError:
            print('\nProgram Quit')
            break
```

Appendix B Figure 27: Running even swaps function part 2

```
def load_files (filename):
    """
    Function to load a xlsx file into a dataframe

    Parameters
    __________
    filename: Name of the xlsx file to load

    Returns
    _______
    df: Loaded dataframe
    """

    df = pd.read_excel(filename)
    return df
```

Appendix B Figure 28: Load files function

```
def background_color (dat, c = 'red'):
    """
    Function to make the background color of a dataframe or data object red, or any other color

    Parameters
    __________
    dat: data object or df to change the background color on
    c: color, in this case it is set to be red

    Returns
    _______
    background color for dat
    """

    return [f'background-color: {c}' for i in dat]
```

Appendix B Figure 29: Background color function

```
def ES_process_and_examples ():
    """
    Function to describe the rules of the even swaps process
    and show an example of how it works

    Parameters
    __________
    None

    Returns
    _______
    None
    """

    rules = """
The following are the rules for conducting even swaps:\n
1. Create a consequences table (CT) of your alternatives and objectives as shown in fig. 1\n
2. Create a ranking table (RT) of the CT as shown in fig. 2
2a. This CT and RT is taken from figures in the Even Swaps article by Hammond et al. (1998)\n
3. Check for equal ranks on an objective in the RT, if equal ranks this objective
can be removed as shown in fig. 3 and fig. 4\n
4. Check for absolute and practical dominance to eliminate alternatives
4a. Absolute dominance is when an alternative has better or equal rank on all objectives as shown in fig. 5
4b. Practical dominance is when an alternative has better or equal rank on all but one objective as shown in fig. 5
4c. For this example, Job B absolutely dominates Job E, this is shown in lime
4d. Jobs C and D are being practically dominated by Jobs B and A respectively, this is shown in magenta
4e. Job A is in cyan, while job B is in green. This is just to distinguish between the alternatives and
the different types of dominance\n
5. The remaining  alternatives and objectives are used for even swaps (fig. 6)
5a. Pick an alternative and an objective, e.g. in fig. 1, it could be alternative 'Job B' and objective 'MS'
5b. For this objective and alternative, change the rank such that it is
equal to the rank of the other remaining alternatives (it is best to do this with alternatives where the rank is similar.
This makes it easier to make a swap.) This is shown in fig. 7
5c. For the same alternative, pick a different objective to make a compensation on, e.g. if you lowered the rank
in step 5b, you would increase the rank for the compensation objective shown in fig. 8
For this example the chosen objective to be swapped is opposite of what the compensation is.
It is easer to make swaps when the values are opposite for the swap and the compensation\n
6. Repeat steps 3 - 5 until a final alternative remains\n
    """

    print(rules)
```

Appendix B Figure 30: ES process and examples function part 1

```
    caption_style = [dict(selector = 'caption',
                          props = [('text-align', 'center'),
                                   ('font-size', '100%'),
                                   ('color', 'black')])]

    #Abbreviations
    abrvs = """
MS = Monthly Salary($)
FY = Flexibility
BSD = Business Skills Development
AVD = Annual Vacation Days
BS = Benefits
ET = Enjoyment
    """

    ct = pd.read_excel('Example Consequence and Ranking Table.xlsx', sheet_name = 'Sahid CT')
    ct_style = ct.style.set_caption('Fig. 1 Consequences Table').set_table_styles(caption_style)
    display(ct_style)

    rt = pd.read_excel('Example Consequence and Ranking Table.xlsx', sheet_name = 'Sahid RT')
    rt_style = rt.style.set_caption('Fig. 2 Ranking Table').set_table_styles(caption_style)
    display(rt_style)

    print('\nBelow are the abbreviations for the objectives for the CT and RT:')
    print(abrvs)

    er = pd.read_excel('Test DF Miller 1.xlsx') #Equal ranks

    er_copy = er.copy()
    highlight = lambda x: ['background: yellow' if x.name in [2] else '' for i in x]
    er_caption = 'Fig. 3 Equal ranks before removal'
    er = er.style.apply(highlight, axis = 1).set_caption(er_caption).set_table_styles(caption_style)

    display(er)

    er_copy.drop([2], axis = 0, inplace = True)
    er_copy_style = er_copy.style.set_caption('Fig. 4 Equal ranks after removal').set_table_styles(caption_style)
    display(er_copy_style)

    rt_copy = rt.copy()

    columns_with_color_dictionary = {'Job E': 'lime', 'Job B': 'green', 'Job D': 'magenta',
                                     'Job C': 'magenta', 'Job A': 'cyan'}
```

Appendix B Figure 31: ES process and examples function part 2

```
    style = rt_copy.style.set_caption('Fig. 5 Absolute and Practical dominance').set_table_styles(caption_style)
    for column, color in columns_with_color_dictionary.items():
        style = style.apply(background_color, axis = 0, subset = column, c = color)
    display(style)

    sahid_caption = 'Fig. 6 Example Ranking Table after removing dominated alternatives'
    sahid_rem_alts = pd.read_excel('Test RT Sahid.xlsx')
    highlight_sahid = lambda x: ['background: yellow' if x.name in [0, 5] else '' for i in x]
    sahid_rem_alts = sahid_rem_alts.style.apply(
        highlight_sahid, axis = 1).set_caption(sahid_caption).set_table_styles(caption_style)

    display(sahid_rem_alts)

    sahid_caption2 = 'Fig. 7 Example Ranking Table after doing an even swap'
    sahid_rem_alts2 = pd.read_excel('Test RT Sahid 2.xlsx')

    sahid_rem_alts_copy = sahid_rem_alts2.copy()

    highlight_sahid2 = lambda x: ['background: yellow' if x.name in [0, 5] else '' for i in x]
    sahid_rem_alts2 = sahid_rem_alts2.style.apply(
        highlight_sahid2, axis = 1).set_caption(sahid_caption2).set_table_styles(caption_style)

    display(sahid_rem_alts2)

    sahid_rem_alts_copy.drop([0, 5], axis = 0, inplace = True)
    sahid_rem_alts_copy_style = sahid_rem_alts_copy.style.set_caption(
        'Fig. 8 Example Ranking table after dropping equal ranks').set_table_styles(caption_style)
    display(sahid_rem_alts_copy_style)
```

Appendix B Figure 32: ES process and examples function part 3

```
def readUsers():
    """
    Function to read users from a json file

    Parameters
    __________
    None

    Returns
    _______
    {}: empty dictionary if not able to find the file
    """
    try:
        with open('users_file.json', 'r') as f: #users.json
            return json.load(f)
    except FileNotFoundError:
        return {}
```

Appendix B Figure 33: Read users function

```
def writeUsers(usr):
    """
    Function to write users to the json file

    Parameters
    __________
    None

    Returns
    _______
    None
    """
    with open('users_file.json', 'w+') as f: #users.json
            json.dump(usr, f)
```

Appendix B Figure 34: Write users function

```
def login(usr):
    """
    Function to login a user to use the even swaps program by asking for username and password

    Parameters
    __________
    usr: users are read in from the function readUsers()

    Returns
    _______
    returns False if the password is incorrect or if the user enters quit
    returns True if able to write user to file by using function writeUsers()
    """
    global uN
    global pW
    flag = True
    while flag:
        uN = input('\nEnter your username or enter quit to quit the program: ') #username
        if uN.lower() == 'quit':
            print('\nProgram quit')
            break
            return False
        pW = input('\nEnter your password or enter quit to quit the program: ') #password
        if pW.lower() == 'quit':
            print('\nProgram quit')
            break
            return False
        if uN in usr.keys():
            if pW == usr[uN]:
                print('\nWelcome back ' + uN + '!')
                flag = False
            else:
                error_msg = ('\nIncorrect password for user: ' + uN)
                error_messages(error_msg)
                return False
        else:
            if uN.lower() == 'quit':
                print('Username is', uN)
                break
                return False
            elif pW.lower() == 'quit':
                print('Password is', pW)
                break
                return False
```

Appendix B Figure 35: Login function part 1

```
45          else:
46              decision = input('\nAre you happy with your username and password? Y or N: ')
47              if decision.lower() in {'y'}:
48                  print('\nWelcome to the even swaps program ' + uN + '!')
49                  usr[uN] = pW
50                  ES_process_and_examples()
51
52                  uflag = True #understanding flag
53                  while uflag:
54                      understanding = input('\nDo you understand the even swaps process? Y or N: ')
55
56                      if understanding.lower() in {'n'}:
57                          ES_process_and_examples()
58                      elif understanding.lower() in {'y'}:
59                          uflag = False
60                          flag = False
61                      else:
62                          error_msg = '\nError! Options are Y or N'
63                          error_messages(error_msg)
64
65              elif decision.lower() in {'n'}:
66                  uN = input('\nEnter your username or enter quit to quit the program: ') #username
67                  pW = input('\nEnter your password or enter quit to quit the program: ') #password
68                  if uN in usr.keys():
69                      if pW == usr[uN]:
70                          print('\nWelcome back ' + uN + '!')
71                          flag = False
72                      else:
73                          print('\nIncorrect password for user:', uN)
74                          return False
75                  else:
76                      print('\nWelcome to the even swaps program ' + uN + '!')
77                      usr[uN] = pW
78                      ES_process_and_examples()
79
80                      uflag = True #understanding flag
81                      while uflag:
82                          understanding = input('\nDo you understand the even swaps process? Y or N: ')
83
```

Appendix B Figure 36: Login function part 2

```
83
84                              if understanding.lower() in {'n'}:
85                                  ES_process_and_examples()
86                              elif understanding.lower() in {'y'}:
87                                  uflag = False
88                                  flag = False
89                              else:
90                                  error_msg = '\nError! Options are Y or N'
91                                  error_messages(error_msg)
92                      else:
93                          error_msg = '\nError! Options are Y or N'
94                          error_messages(error_msg)
95
96      writeUsers(usr)
97      return True
```

Appendix B Figure 37: Login function part 3

```
def ES_and_file_creation ():
    """
    Function that executes all other functions, and asks the user whether he/she would like
    to create a csv file of the final dataframe

    Parameters
    __________
    None

    Returns
    _______
    return_msg: Returns message if there is no final dataframe
    """

    global uN
    global pW

    #esp: Even swap program
    esp_rules = """
The following are the rules for using the even swaps program:\n
1. You are asked to input your username and password
1a. If new user, you will input what you want your username and password to be
1b. Then, the rules for conducting even swaps are displayed for you and you will go through an even swaps example
1c. If returning user, you will be asked whether you would like to create a consequences table (CT) and ranking table (RT)
or if you would like to load a CT or RT from a file
2. You are asked to create a RT if you create a CT in the program or if you load a CT from a file
3. Once in RT format, the program checks for equal attributes and removes these and checks for dominance
removing any dominated alternative
4. The program asks whether you would like to make an even swap and displays whether an alternative is likely
to dominate another, and displays the objective ranks that are almost equal such that it is easier to see what
objectives and alternatives you should make swaps on
    """

    print(esp_rules)
```

Appendix B Figure 38: ES and file creation function part 1

```
35
36      user_info_flag = True
37      flag = True
38      while user_info_flag:
39          try:
40              users = readUsers()
41              success = login(users)
42              if uN.lower() == 'quit' or pW.lower() == 'quit':
43                  break
44              else:
45                  while not success:
46                      success = login(users)
47                  while flag:
48                      try:
49                          input_decision = input('\nWould you like to Create Your Own (CYO) or Load a Created File (LCF)?' +
50                                                 ' (Options are CYO or LCF) Enter quit to quit: ')
51                          if input_decision.lower() == 'quit':
52                              user_info_flag = False
53
54                          df = user_choice(input_decision)
55                      except EOFError:
56                          print('\nProgram Quit')
57                          break
58
59                      final_df = running_even_swaps(df)
60
61                      return_msg = ''
62
63                      if final_df is None:
64                          return_msg = "There is no final dataframe"
65                          print()
66                          return return_msg
67
68                      else:
69                          print('\nThis is the final alternative')
70
71                          display(final_df)
72
73                          ifd_text = '\nWould you like to create a file of the final dataframe? Y or N: '
74                          input_filename_decision = input(ifd_text)
75                          if input_filename_decision.lower() in {'y'}:
```

Appendix B Figure 39: ES and file creation function part 2

```
                            input_filename = input('\nEnter the name of your csv file: ') + '.csv'
                            print()
                            print(input_filename)

                            final_df.to_csv(input_filename)
                            read_df = pd.read_csv(input_filename)
                            read_df.drop(read_df.iloc[:,0:1], inplace = True, axis = 1)
                            print('\nDisplaying dataframe of final alternative file')
                            display(read_df)
                            flag = True

                        elif input_filename_decision.lower() in {'n'}:
                            flag = True

                        else:
                            error_msg = '\nError! Options are Y or N'
                            error_messages(error_msg)
        except EOFError:
            print('\nProgram quit')
            user_info_flag = False
            flag = False

es_func = ES_and_file_creation()
print(es_func)
```

Appendix B Figure 40: ES and file creation function part 3

B2 Two-attribute example code

In this appendix is the complete code for the two-attribute example.

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as ss
```

Appendix B Figure 41: Importing libraries for two-attribute example

```python
def prob_dom (x11, x12, n_steps, probLB, probUB):
    """
    Function to calculate probable dominance for the two-attribute example from Bhattacharjya and Kephart (2014)

    Parameters
    __________
    x11: x-coordinate for alternative 1 (attribute x1)
    x12: y-coordinate for alternative 1 (attribute x2)
    n_steps: number of steps for x1 and x2 for alternative 2
    probLB: lower bound for uniform distribution
    probUB: upper bound for uniform distribution

    Returns
    _______
    None
    """

    n_min = 1 / n_steps / 2
    n_max = 1 - n_min

    x21 = np.linspace(n_min, n_max, n_steps)
    x22 = np.linspace(n_min, n_max, n_steps)

    p_x1x2 = []

    for j in reversed(x22):
        p_x1 = []
        for i in x21:
            x_num = x12 - j
            x_denom = i - j - x11 + x12
            if x_denom > 0:
                prob = ss.uniform.cdf(x = ((x_num) / (x_denom)), loc = probLB, scale = probUB - probLB)
                p_x1.append(prob)
            elif x_denom < 0:
                prob = 1 - ss.uniform.cdf(x = ((x_num) / (x_denom)), loc = probLB, scale = probUB - probLB)
                p_x1.append(prob)
            elif x_denom == 0:
                if x_num > 0:
                    p_x1.append(1)
                elif x_num < 0:
                    p_x1.append(0)
        p_x1x2.append(p_x1)
```

Appendix B Figure 42: Probable dominance function part 1

```
43
44      # getting the original colormap using cm.get_cmap() function
45      orig_map = plt.cm.get_cmap('bwr')
46
47      # reversing the original colormap using reversed() function
48      reversed_map = orig_map.reversed()
49
50      plt.figure(figsize = (12, 7))
51
52      plt.xlabel('Attribute $x_1$')
53      plt.ylabel('Attribute $x_2$')
54
55      # Add lines
56      x1 = [0, 1]
57      y1 = [x12, x12]
58      plt.plot(x1, y1, "-", color = "black", linewidth = 1, alpha = 0.3)
59
60      x2 = [x11, x11]
61      y2 = [0., 1.]
62      plt.plot(x2, y2, "-", color = "black", linewidth = 1, alpha = 0.3)
63
64      diag = x11 + x12
65      x = [0, diag]
66      y = [diag, 0]
67      plt.plot(x, y, "--", color = "black", linewidth = 1)
68
69      plt.scatter(x11, x12, s = 100, facecolors = ['black'], edgecolors = ['black'], norm = False)
70
71      plt.imshow(p_x1x2, cmap = reversed_map, extent = [0, 1, 0, 1])
72      plt.show()
```

Appendix B Figure 43: Probable dominance function part 2

```
def prob_dom_example ():
    """
    Function to input different values for x1_a1, x2_a1, n_steps, u_min, and u_max
    for the two-attribute example with a uniform distribution

    Parameters
    __________
    None

    Returns
    _______
    None
    """

    # Distribution over w_1 is U(u_min, u_max). w1 can take any value on [0, 1] but the
    # probablity p(w1) is constrained by U(u_min, u_max)

    x1_a1 = float(input('\nWhat would you like the value for x\u2081(a\u2081) to be? '))
    x2_a1 = float(input('\nWhat would you like the value for x\u2082(a\u2081) to be? '))
    n_steps = int(input('\nHow many steps would you like to use? '))
    u_min = float(input('\nWhat would you like the min value for u to be? '))
    u_max = float(input('\nWhat would you like the max value for u to be? '))
    print()

    prob_dom(x1_a1, x2_a1, n_steps, u_min, u_max)

prob_dom_example()
```

Appendix B Figure 44: Function to test different values for the two-attribute example

Appendix C

In this appendix is the letter from Benjamin Franklin to Joseph Priestly, reprinted from

(Hammond et al., 1998).

London

Sept. 19, 1772

Dear Sir,

In the affair of so much importance to you, wherein you ask my advice, I cannot, for want of sufficient premises, advise you *what* to determine, but if you please I will tell you *how*.

When those difficult cases occur, they are difficult, chiefly because while we have them under consideration, all the reasons *pro* and *con* are not present to the mind at the same time; but sometimes one set present themselves, and at other times another, the first being out of sight. Hence the various purposes or inclinations alternatively prevail, and the uncertainty that perplexes us.

To get over this, my way is to divide half a sheet of paper by a line into two columns; writing over the one *Pro,* and over the other *Con*. Then, during three or four days consideration, I put down under the different heads short hints of the different motives, that at different times occur to me, for or against the measure.

When I have thus got them all together in one view, I endeavor to estimate their respective weights; and where I find two, one on each side, that seem equal, I strike them both out. If I find a reason *pro* equal to two reasons *con*, I strike out the three. If I judge some two reasons *con*, equal to some three reasons *pro*, I strike out the five; and thus proceeding I find at length where the balance lies; and if, after a day or two of further consideration, nothing new that is of importance occurs on either side, I come to a determination accordingly.

And, though the weight of reasons cannot be taken with the precision of algebraic quantities, yet, when each is thus considered, separately and comparatively, and the whole lies before me, I think I can judge better, and am less liable to make a rash step; and in fact I have found great advantage from this kind of equation, in what may be called *moral or prudential algebra*.

Wishing sincerely that you may determine for the best, I am ever, my dear friend, yours most affectionately,

B. Franklin