MASTERARBEIT / MASTER’S THESIS

Titel der Masterarbeit / Title of the Master‘s Thesis

„Inequity-averse two-stage location decision-making processes“

verfasst von / submitted by

Judith Feltl, BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2021 / Vienna 2021

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 915
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Betriebswirtschaft
Betreut von / Supervisor:	ao. Univ.-Prof. i.R. Mag. Dr. Walter Gutjahr

Contents

1. Introduction 1
 1.1. Motivation 1
 1.2. Problem Statement 3
 1.3. Overview 4
2. Two-Stage Location Decisions 5
 2.1. Model 5
 2.2. Scenarios 6
 2.3. P-Center Problem 8
3. Evaluation Methods 10
 3.1. Conception of Fairness 10
 3.2. Calculation 11
 3.3. Ex-Ante Evaluation 12
 3.4. Ex-Post Evaluation 13
 3.5. Utilitarian Objective Function 14
4. Complete Enumeration 16
5. Genetic Algorithm 18
 5.1. Introduction 18
 5.2. Structure 19
 5.3. Chromosome Representation 20
 5.4. Initial Population 22
 5.5. Fitness Values and Diversity 23
 5.6. Selection 24
 5.7. Crossover 28
 5.8. Mutation 31
 5.9. Termination 32
6. Results 33
 6.1. General Settings and Overview 33
 6.2. Input Data and Randomness 34
 6.3. Small Problem Size 35
  6.3.1. Complete Enumeration 35
   6.3.1.1. Increasing Population Nodes 35
    6.3.1.1.1. Fitness Values and Performance 35
    6.3.1.1.2. Comparison of Costs 40

6.3.1.2. Increasing Potential Locations 44
6.3.1.2.1. Fitness Values and Performance 44
6.3.1.2.2. Comparison of Costs 46
6.3.1.3. Increasing Permanent Shelters 47
6.3.1.3.1. Fitness Values and Performance 47
6.3.1.3.2. Comparison of Costs 49
6.3.1.4. Increasing Temporary Locations 50
6.3.1.4.1. Fitness Values and Performance 50
6.3.1.4.2. Comparison of Costs 52
6.3.2. Genetic Algorithm 53
6.3.2.1. Increasing Population Nodes 53
6.3.2.1.1. Fitness Values and Performance 53
6.3.2.1.2. Comparison of Costs 55
6.3.2.2. Increasing Potential Locations 56
6.3.2.2.1. Fitness Values and Performance 56
6.3.2.2.2. Comparison of Costs 57
6.3.2.3. Increasing Permanent Shelters 58
6.3.2.3.1. Fitness Values and Performance 58
6.3.2.3.2. Comparison of Costs 60
6.3.2.4. Increasing Temporary Shelters 61
6.3.2.4.1. Fitness Values and Performance 61
6.3.2.4.2. Comparison of Costs 63
6.4. Large Problem Size 64
6.4.1. Fitness Values and Performance 64
6.4.2. Comparison of Costs 70
6.4.3. Comparison to Gutjahr (2020) 73
6.5. Performance CE vs. GA 75
6.5.1. Visual Comparison 75
6.5.2. Time Measurement 77
6.6. Case Study: Nepal 79
6.6.1. Settings 79
6.6.2. Applying Genetic Algorithm to Scenarios 80
6.6.2.1. Generating Scenarios 80
6.6.2.2. Fitness Values and Performance 81
6.6.2.3. Comparison of Costs 83

6.6.3. Real Event 85
6.6.3.1. Affected Districts 85
6.6.3.2. Fitness Values and Performance 86
6.6.3.3. Comparison of Costs 87
6.6.4. Comparison of Scenarios and Real Event 88
6.6.4.1. Fitness Values 89
6.6.4.2. Comparison of Costs 91
6.6.5. Choose Solution 92
6.7. Budget 93
6.7.1. Settings 93
6.7.2. Fitness Values and Performance 93
6.7.3. Comparison of Costs 95
6.7.4. Impact of Settings 96
6.7.4.1. Large Number of Solutions 97
6.7.4.2. Small number of Potential Locations 97
6.7.4.3. Chosen Settings 99
7. Conclusion 100
References 101
Appendix A – Complete Enumeration 103
Appendix B – Genetic Algorithm 109
Appendix C – Large Problem Size 115
Abstract 118
Zusammenfassung 119

List of Figures

Figure 1-1 Reported Disasters 2
Figure 1-2 People Killed 2
Figure 1-3 People Affected 2
Figure 1-4 Disaster Damage 2
Figure 2-1 Two-Stage Model 5
Figure 2-2 Scenarios in Excel 8
Figure 2-3 P-Center Problem 9
Figure 3-1 Ex-Ante Calculation 12
Figure 3-2 Ex-Post Calculation 13
Figure 3-3 Utilitarian Calculation 15
Figure 4-1 Complete Enumeration 16
Figure 4-2 Pseudocode Complete Enumeration 17
Figure 5-1 Genetic Algorithm Model 19
Figure 5-2 Pseudocode Genetic Algorithm 19
Figure 5-3 Binary and Integer Chromosomes 21
Figure 5-4 Pseudocode Initial Population 23
Figure 5-5 Pseudocode GA Fitness 24
Figure 5-6 Roulette Wheel Selection 25
Figure 5-7 Tournament Selection 26
Figure 5-8 Pseudocode Roulette Wheel Selection 26
Figure 5-9 Pseudocode Tournament Selection 27
Figure 5-10 One-Point and Two-Point Crossover 28
Figure 5-11 Pseudocode One-Point Crossover 29
Figure 5-12 Pseudocode Two-Point Crossover 30
Figure 5-13 Pseudocode Repair-Mechanism 31
Figure 5-14 Pseudocode Mutation 32
Figure 6-1 Generating Random Numbers 34
Figure 6-2 CE: Population Nodes: Single Seed: Fitness Values 36
Figure 6-3 CE: Population Nodes: Single Seed: Percentage Differences and Performance ... 37
Figure 6-4 CE: Population Nodes: Fitness Values 39
Figure 6-5 CE: Population Nodes: Percentage Differences and Performance 39
Figure 6-6 CE: Population Nodes: Single Seed: Ex-Ante and Ex-Post Costs 42
Figure 6-7 CE: Population Nodes: Ex-Ante and Ex-Post Costs 43
Figure 6-8 CE: Potential Locations: Fitness Values 44
Figure 6-9 CE: Potential Locations: Percentage Differences 45
Figure 6-10 CE: Potential Locations: Performance and Combinations 46
Figure 6-11 CE: Potential Locations: Ex-Ante and Ex-Post Costs 47
Figure 6-12 CE: Permanent Shelters: Fitness Values 48
Figure 6-13 CE: Permanent Shelters: Percentage Differences 48
Figure 6-14 CE: Permanent Shelters: Performance and Combinations 49
Figure 6-15 CE: Permanent Shelters: Ex-Ante and Ex-Post Costs 50
Figure 6-16 CE: Temporary Shelters: Fitness Values 51
Figure 6-17 CE: Temporary Shelters: Percentage Differences and Performance 51
Figure 6-18 CE: Temporary Locations: Ex-Ante and Ex-Post Costs 52
Figure 6-19 GA: Population Nodes: Fitness Values 54

Figure 6-20 GA: Population Nodes: Percentage Differences and Performance 54
Figure 6-21 GA: Population Nodes: Ex-Ante and Ex-Post Costs 55
Figure 6-22 GA: Potential Locations: Fitness Values 56
Figure 6-23 GA: Potential Locations: Percentage Differences and Performance 57
Figure 6-24 GA: Potential Locations: Ex-Ante and Ex-Post Costs 58
Figure 6-25 GA: Permanent Shelters: Fitness Values 59
Figure 6-26 GA: Permanent Shelters: Percentage Differences and Performance 59
Figure 6-27 GA: Permanent Shelters: Combinations 60
Figure 6-28 GA: Permanent Shelters: Ex-Ante and Ex-Post Costs 61
Figure 6-29 GA: Temporary Shelters: Fitness Values 62
Figure 6-30 GA: Temporary Shelters: Percentage Differences and Performance 62
Figure 6-31 GA: Temporary Shelters: Ex-Ante and Ex-Post Costs 63
Figure 6-32 Large GA: Fitness Values Ex-Ante 66
Figure 6-33 Large GA: Fitness Values Ex-Post 67
Figure 6-34 Large GA: Percentage Differences 68
Figure 6-35 Large GA: Performance 68
Figure 6-36 Large GA: Variance in Numbers of Generations 69
Figure 6-37 Large GA: Ex-Ante Costs 72
Figure 6-38 Large GA: Ex-Post Costs 72
Figure 6-39 Large GA: Variance of Percentage Cost Differences 74
Figure 6-40 Comparison CE vs GA, Potential Locations, 100 Seeds 75
Figure 6-41 Comparison CE vs GA, Potential Locations, Single Seed 76
Figure 6-42 Comparison CE vs GA, Temporary Shelters, 100 Seeds 76
Figure 6-43 Comparison CE vs GA, Temporary Shelters, Single Seed 77
Figure 6-44 Nepal: Map of Central Nepal 79
Figure 6-45 Nepal: Scenarios 81
Figure 6-46 Nepal: Fitness Values 82
Figure 6-47 Nepal: Percentage Differences 82
Figure 6-48 Nepal: Performance 83
Figure 6-49 Nepal: Ex-Ante and Ex-Post Costs 84
Figure 6-50 Nepal: Earthquake April 2015 85
Figure 6-51 Nepal: Real Event: Fitness Values 86
Figure 6-52 Nepal: Real Event: Percentage Differences and Performance 87
Figure 6-53 Nepal: Real Event: Ex-Ante and Ex-Post Costs 88
Figure 6-54 Nepal: Comparison: Fitness Values 89
Figure 6-55 Nepal: Comparison: Percentage Differences 90
Figure 6-56 Nepal: Comparison: Ex-Ante and Ex-Post Costs 92
Figure 6-57 Budget: Fitness Values 94
Figure 6-58 Budget: Percentage Differences and Performance 94
Figure 6-59 Budget: Variance in Needed Generations, Frequency of Occurrences 95
Figure 6-60 Budget: Ex-Ante and Ex-Post Costs 96
Figure 6-61 Impact: Fitness Values 98
Figure 6-62 Impact: Performance 98
Figure 6-63 Chosen Settings: Performance 99

List of Tables

Table 2-1 Excel: Imported Coordinates 7
Table 2-2 Excel: Imported Scenario Data 7
Table 5-1 Problem Sizes 21
Table 5-2 GA: Performance Comparison 22
Table 5-3 GA: Percentages of Individuals 25
Table 6-1 Default Settings 35
Table 6-2 Additional Settings for GA 35
Table 6-3 CE: Population Nodes: Single Seed: Fitness Values 36
Table 6-4 CE: Population Nodes: Fitness Values 38
Table 6-5 CE: Population Nodes: Single Seed: Comparison of Costs 41
Table 6-6 CE: Population Nodes: Comparison of Costs 42
Table 6-7 CE: Potential Locations: Fitness Values 44
Table 6-8 CE: Potential Locations: Permanent and Temporary Combinations 45
Table 6-9 CE: Potential Locations: Comparison of Costs 46
Table 6-10 CE: Permanent Shelters: Fitness Values 47
Table 6-11 CE: Permanent Shelters: Permanent and Temporary Combinations 48
Table 6-12 CE: Permanent Shelters: Comparison of Costs 49
Table 6-13 CE: Temporary Shelters: Fitness Values 50
Table 6-14 CE: Temporary Shelters: Comparison of Costs 52
Table 6-15 GA: Population Nodes: Fitness Values 53
Table 6-16 GA: Population Nodes: Comparison of Costs 55
Table 6-17 GA: Potential Locations: Fitness Values 56
Table 6-18 GA: Potential Locations: Comparison of Costs 57
Table 6-19 GA: Permanent Shelters: Fitness Values 58
Table 6-20 GA: Permanent Shelters: Permanent and Temporary Combinations 60
Table 6-21 GA: Permanent Shelters: Comparison of Costs 60
Table 6-22 GA: Temporary Shelters: Fitness Values 61
Table 6-23 GA: Temporary Shelters: Comparison of Costs 63
Table 6-24 Large GA: Fitness Values 65
Table 6-25 Large GA: Comparison of Costs 71
Table 6-26 Large GA: Comparison of Percentage Differences 73
Table 6-27 Large GA: Number of Occurrences of Smaller Percentage Differences 74
Table 6-28 Performance: Ex-Ante 78
Table 6-29 Performance: Ex-Post 78
Table 6-30 Nepal: Fitness Values 81
Table 6-31 Nepal: Number of Generations 83
Table 6-32 Nepal: Comparison of Costs 83
Table 6-33 Nepal: Real Event: Fitness Values 86
Table 6-34 Nepal: Real Event: Number of Generations 87
Table 6-35 Nepal: Real Event: Comparison of Costs 88
Table 6-36 Nepal: Comparison: Fitness Values 89
Table 6-37 Nepal: Comparison: Comparison of Costs 91
Table 6-38 Nepal: Solution: Decision-Making based on Fitness Values 92
Table 6-39 Nepal: Solution: Decision-Making based on Utilitarian Costs 92
Table 6-40 Budget: Settings for 4/1 and 2/2 93

Table 6-41 Budget: Fitness Values 94
Table 6-42 Budget: Comparison of Costs 96
Table 6-43 Impact of Settings: Number of Solutions, 20 Potential Locations 97
Table 6-44 Impact of Settings: Number of Solutions, 12 Potential Locations 97
Table 6-45 Chosen Settings: Number of Solutions 99

1. Introduction

1.1. Motivation

According to the International Federation of Red Cross and Red Crescent Societies (IFRC) (2016), millions of people are at risk to suffer from preventable disasters. Although this is a well-known problem, the global investment is low when it comes to strengthening community resilience. Out of 100 US dollars that are used for international aid, only 40 cents are spent on means to reduce the risk of disasters, though it can save money and even lives. Therefore, the IFRC suggests that more money needs to be spent finding better solutions that will reduce the impact of disasters and help communities deal with them.

Alejandra Borunda (2019) from National Geographic states that the climate change also affects the severity of disasters: The on average warmest years ever measured by the National Oceanic and Atmospheric Administration (NOAA) are 2014 to 2018, with, according to the IFRC (2016), 32 major droughts in 2015.

In order to get a better overview of the recent situation regarding disasters, the following four figures are created with some of the data of Tables 1 to 4 (original source: EM-DAT, CRED, University of Louvain, Belgium) of the Worlds Disasters Report 2016 (Sanderson and Sharma, 2016). For simplicity reasons only the number of disasters per region (Africa, America, Asia, Europe, and Oceania) and not the number of disasters per current state of human development are considered in this overview. The figures show data from 2006 to 2015 for each region and year separately and the sum ("Total") of all regions for each year.

Figure 1-1 shows the number of reported disasters per year, which are relatively constant for most of the regions. On average, 609 disasters occurred per year during these ten years. "Total" shows an overall decrease from 2006 to 2013 with a peak in 2010.

According to Guha-Sapir, Vos, Below, and Ponserre (2011) at CRED (Centre for Research on the Epidemiology of Disasters), 2010 was the worst year in terms of deaths in over two decades. The two major disasters were the earthquake in Haiti and the heat wave (June – August) in Russia. The reported deaths were 222.570 in Haiti, and 55.800 (most of them caused by the heat wave) in Russia, which explains the highest peak in Figure 1-2. The other peak is caused by two enormous disasters in Asia, a cyclone and an earthquake, which accounted worldwide for 95.9% (or over 225.000 in absolute figures) of the reported deaths (Rodriguez, Vos, Below, and Guha-Sapir, 2009). The average number of deaths per year from 2006 - 2015 is 77.191,11, whereas the average number of people affected is 191.755.600.

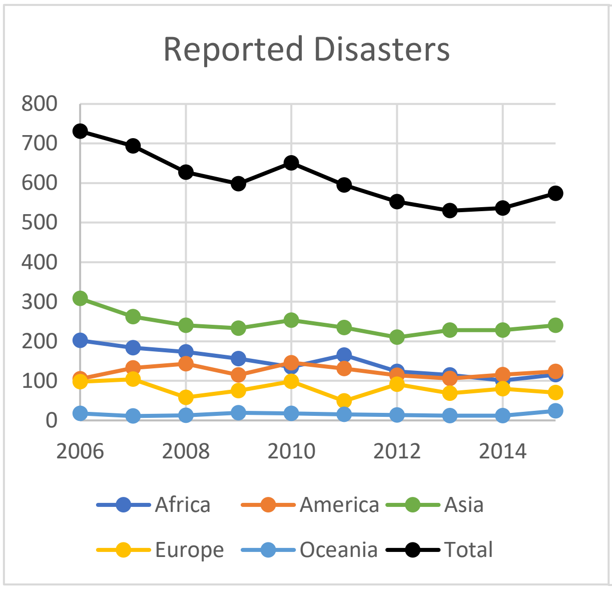


Figure 1-1 Reported Disasters from 2006-2015

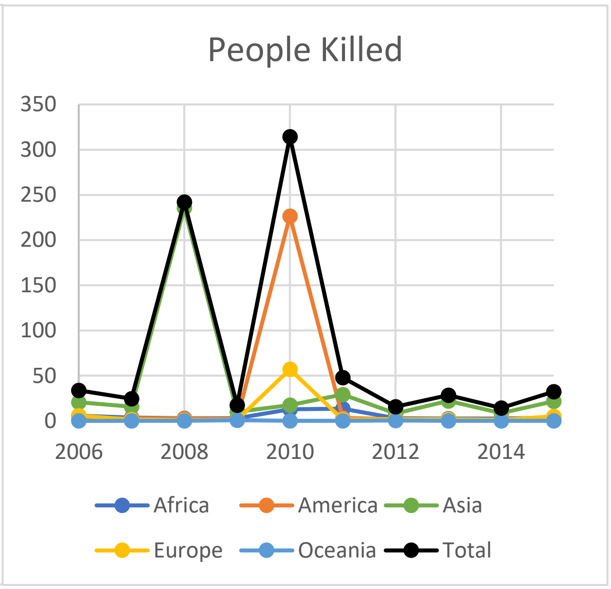


Figure 1-2 People Killed from 2006-2015, in thousands

Figure 1-3 shows that Asia is a lot more affected by disasters than the other regions. Guha-Sapir, Vos, Below, and Ponserre (2012) state that 33 disasters including floods, storms, and tropical cyclones hit the Philippines in 2011 and affected many people. This country has never seen that many disasters, while Europe has had the smallest number of disasters since 1989. The number of disasters reported in Oceania was similarly low as in Europe and thus not many people were affected (Guha-Sapir et al., 2012).

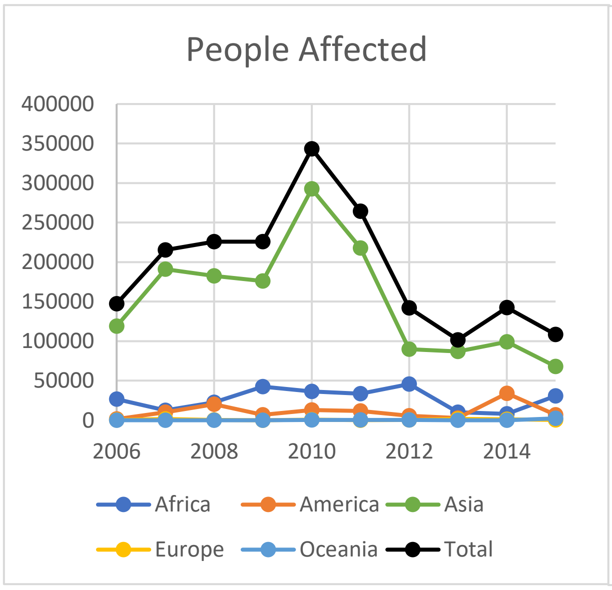


Figure 1-3 People Affected from 2006-2015, in thousands

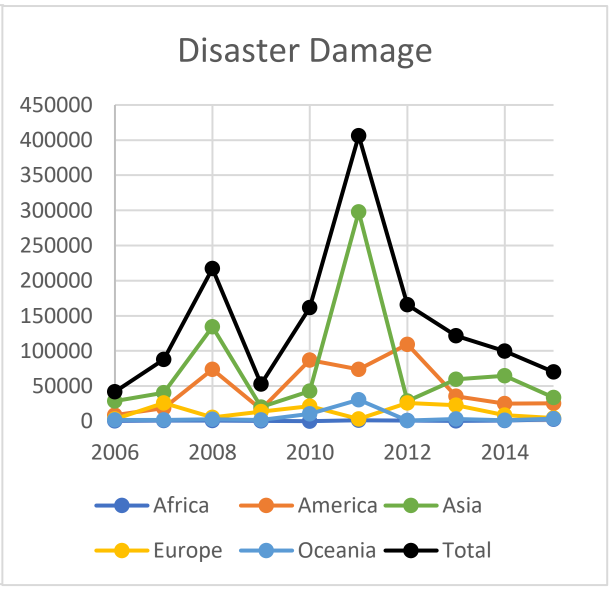


Figure 1-4 Disaster Damage (estimated) from 2006-2015, in millions US $

The economic damage of disasters during these ten years is on average 142.481.400.000 US Dollars. “Total” in Figure 1-4 shows a peak in 2008 and another one in 2011. Both peaks were mainly caused by disasters in America and Asia. In 2008, hydrological disasters in America caused more damage than those of the previous years and two massive disasters in Asia, a cyclone and an earthquake, accounted for 61.5% of the monetary damage worldwide (Rodriguez et al., 2009). In America in 2011, mainly hydrological and climatological disasters amounted to a damage of 67.3 billion US Dollars and in Asia an increase in geophysical and hydrological disasters caused a damage of 276.0 billion US Dollars, which was the highest damage reported so far (Guha-Sapir et al., 2012).

The discussed data stresses the importance of making major decisions in advance to be prepared and react quickly when a disaster occurs. In the long run, this will save lives and reduce costs. Since decisions must be fair (inequity-averse) in humanitarian operations, the goal is to distribute benefits equally among affected individuals (Mostajabdaveh, Gutjahr, and Salman, 2020).

1.2. Problem Statement

When planning future disaster relief, decision-makers do not know where the disaster will occur or how bad its effects will be. This work focuses on the preparedness aspect of disaster management by developing a fair plan that provides affected people with shelters.

This plan, later called solution, consists of two stages:
In the first stage, permanent shelters are built before the disaster occurs. The advantages of doing this are that these shelters do not have to be finished fast and are therefore cheaper, and in case a disaster occurs, there are already some shelters available.
The disaster itself is simulated by creating scenarios with different epicenters and strengths of impact. A second-stage decision is made for each of these scenarios separately, and temporary shelters are built, which are more expensive than the permanent shelters as they are needed immediately. This is important when a budget must be considered.

Permanent and temporary shelters can only be built at potential locations. Which of these locations are finally chosen depends on fairness values. “Fairness” is maximized (inequity-averse) and evaluated by two different approaches, ex-ante and ex-post. The aim of this thesis is to compare the results of these two approaches.

The solution is once determined by a complete enumeration algorithm and another time the first-stage decision is computed by a genetic algorithm, while the second-stage decision is still determined by the complete enumeration. These outcomes are then compared.

Additionally, the best utilitarian solution is calculated and compared to the utilitarian costs of the ex-ante and ex-post solution.

1.3. Overview

In the first part of this thesis, the relevant theory and the implementation of the algorithms is discussed, while Section 6 provides the experimental results.

Section 2 introduces and explains the two-stage model. Afterwards it is shown how scenarios are generated and then displayed in Excel. The last subsection visualizes how locations for shelters are chosen so that the distance to the population nodes is minimized.

Section 3 contains the calculation of the fairness and utilitarian values. Two evaluation approaches, ex-ante and ex-post, are introduced and the utilitarian objective function is explained.

Pseudo-codes are used to discuss the complete enumeration algorithm and the genetic algorithm in Sections 4 and 5 respectively. The basic operators a genetic algorithm usually consists of are explained in Section 5 as well.

In Section 6, the results of the two algorithms are discussed and the ex-ante and ex-post solutions are compared. Furthermore, the best ex-ante/ex-post solution is plugged into the ex-post/ex-ante objective function to compare the percentage differences of the solutions. A case study was conducted on the Gorkha earthquake that occurred in Nepal in April 2015, and the solution obtained is compared with the best possible solution. The last subsection introduces a budget and compares different combinations of permanent and temporary shelters.

2. Two-Stage Location Decisions

2.1. Model

As already mentioned before, two decisions must be made by the decision-maker. The first decision is made before the disaster occurs and is called first-stage decision. Here, it is decided at which potential locations the permanent shelters are built. Then, after some time, a disaster happens.

Multiple scenarios are generated to simulate real disasters that affect different areas. Only a few scenarios are considered because otherwise the computing time would be too high.

After the disaster occurred, the second-stage decision must be made. Temporary shelters are built at some of the remaining potential locations. At least one permanent shelter and one temporary shelter must be built.

The two-stage model visualized in Figure 2-1 is based on the decision tree of Gutjahr (2020), who also addresses inequity-averse optimization under uncertainty and compares the results of the ex-ante and ex-post approach. In his paper, individuals are assigned to the best shelter in the second-stage decision, while additional shelters are built in this thesis, which is the main difference between the two models.

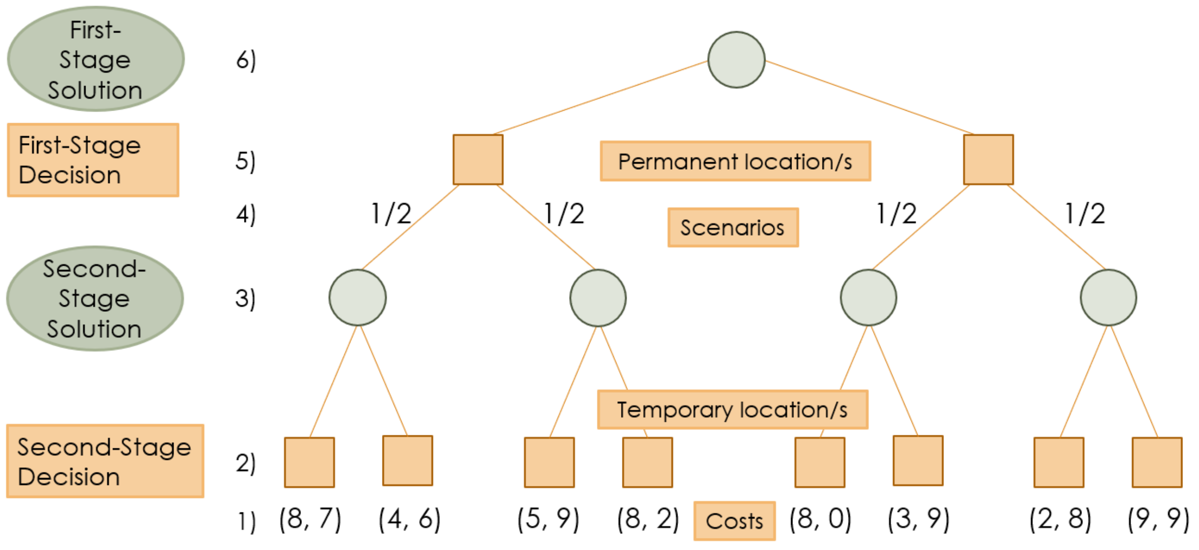


Figure 2-1 Two-Stage Model, adapted from Gutjahr (2020)

Decisions are made from bottom to top, which is called backward induction (Gutjahr, 2020).

1) In this model there are two population nodes. A population node can represent e.g. one person, village, or city. The first index of the cost vector shows the distance of the first population node to its nearest built shelter. These distances are also referred to as costs or

fairness values and their calculation is later discussed in subsection 2.3 “P-Center Problem”. The same applies to the second index.

2) A square represents a combination of temporary shelters selected from the remaining potential locations. The remaining potential locations consist of all potential locations minus those locations that were selected for the permanent shelters. There are as many squares as there are possible combinations (complete enumeration). Each combination consists of as many potential locations as temporary shelters should be built. This model has two different temporary combinations for each permanent combination.
3) The second-stage solution contains the decision of 2), where the fairest temporary combination of the scenario (or random event) is chosen. This decision is based on the chosen evaluation approach, ex-ante or ex-post.
4) Next come the scenarios or random events visualized by the edges. There are two scenarios per permanent combination and each of them has a probability of 1/2 (50%) to occur. For any permanent combination, the sum of the percentages of all scenarios is always 100%.
5) These squares represent all possible permanent combinations. In this case, there are two of them.
6) The very top of the model contains the fairest combination of the first-stage decision of the previous step. Which combination is chosen depends again on the applied evaluation approach.

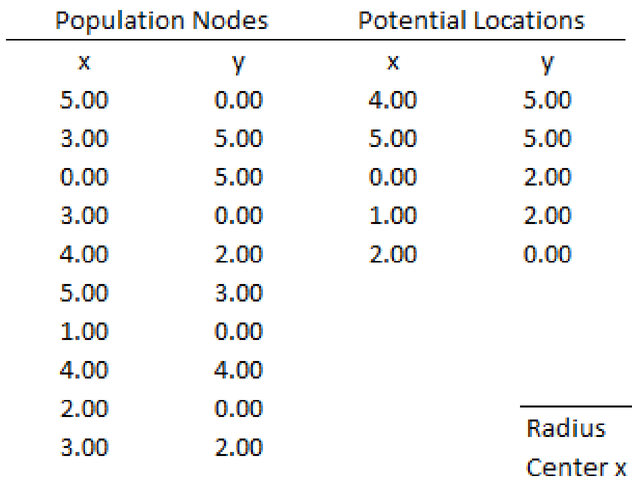
A solution or plan to this problem includes the chosen permanent combination and a temporary combination for each scenario. For example, the solution consists of the right permanent combination and the left temporary combination for both scenarios. If a disaster occurs that is more similar to the left than the right scenario, the temporary shelters of the left scenario are built and the distances of the population nodes to the available shelters are (8, 0).

2.2. Scenarios

The affected population nodes of each scenario are selected either manually or at random. When a scenario is generated randomly, a circle is randomly placed on a predefined area and any population node that lies inside the circle is affected. The radius of this circle is also chosen randomly but lies within a predefined range.

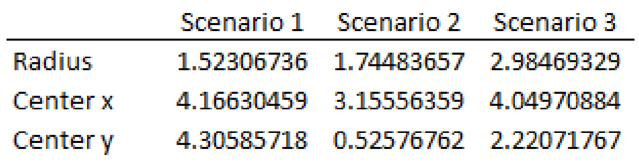
In order to visualize the randomly generated scenarios, the centers and radiuses can be exported to Excel. Of course, the population nodes and potential locations must also be exported, and it is optional to add a weight to a population node, equal to the number of households. A relatively easy way to export data to Excel, and how this is done in this part of the code, is to use the external library “LibXL” (XLware, n.d.). Except for the case study, this library is not used in the “Results” section because it is a trial version and has limitations. Instead, the Excel files are converted to CSV files. An example of what the visualization looks like and how it is set up in Excel is given below.

The following two tables, Table 2-1 and Table 2-2, are copied from the Excel file and contain the imported coordinates of the population nodes and potential locations, as well as the center point coordinates and radiuses of the scenarios.



Population Nodes		Potential Locations	
x	y	x	y
5.00	0.00	4.00	5.00
3.00	5.00	5.00	5.00
0.00	5.00	0.00	2.00
3.00	0.00	1.00	2.00
4.00	2.00	2.00	0.00
5.00	3.00		
1.00	0.00		
4.00	4.00		
2.00	0.00		
3.00	2.00		

Table 2-1 Excel: Imported Coordinates



	Scenario 1	Scenario 2	Scenario 3
Radius	1.52306736	1.74483657	2.98469329
Center x	4.16630459	3.15556359	4.04970884
Center y	4.30585718	0.52576762	2.22071767

Table 2-2 Excel: Imported Scenario Data

Figure 2-2 shows an area with population nodes and potential locations. This area represents a map. The first map does not have a scenario but each of the other identical maps visualizes a different scenario. Every population node that lies within or on the circle is affected.
The coordinate systems must be created in advance as this is not possible with the external library. There is a detailed explanation of how a circle is visualized in an Excel chart on the website of Tushar Mehta (n.d.), with the small adjustment that the X and Y coordinates of the circle centers, which would otherwise be at (0|0), are added to the circle equation.

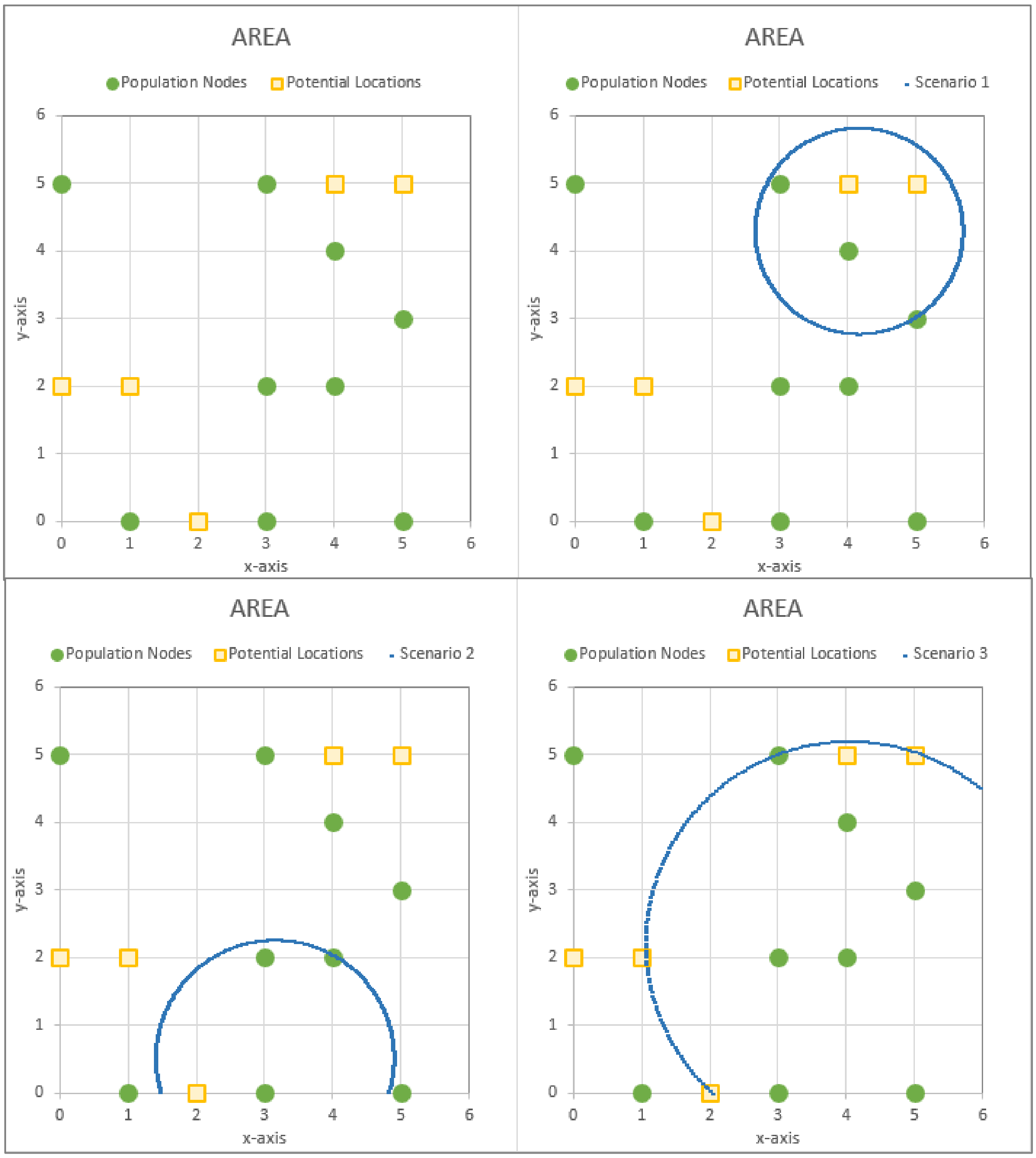


Figure 2-2 Scenarios in Excel

2.3. P-Center Problem

A cost vector in Figure 2-1 of subsection 2.1 “Model” contains the distance of each population node to its nearest built shelter. Unaffected population nodes have a cost of 0. These distances are Euclidian distances, and the p-center problem is used to select the locations for shelters where the distances to the population nodes are minimized.

The idea of the p-center problem is as follows: There is a set of M potential locations. A chosen number of p shelters is built at some of the M potential locations so that the maximum distance of the population nodes that are assigned to the nearest shelters is minimized (Elloumi, Labbé, and Pochet, 2004).

In Figure 2-3, the general concept of the p-center problem is demonstrated.

1) At first, each of the population nodes is assigned to the nearest built shelter.
2) After that, all distances are compared, and the longest distance is marked in red. This means that the population node that is connected to this red arrow is further away from its nearest shelter than all other population nodes and therefore has the maximum distance of this problem.
3) Now this distance must be minimized by building an additional shelter. The distances between this population node and all remaining potential locations are compared.
4) The nearest location is chosen and marked, and a shelter is built. Simply put, the maximum distance is minimized.

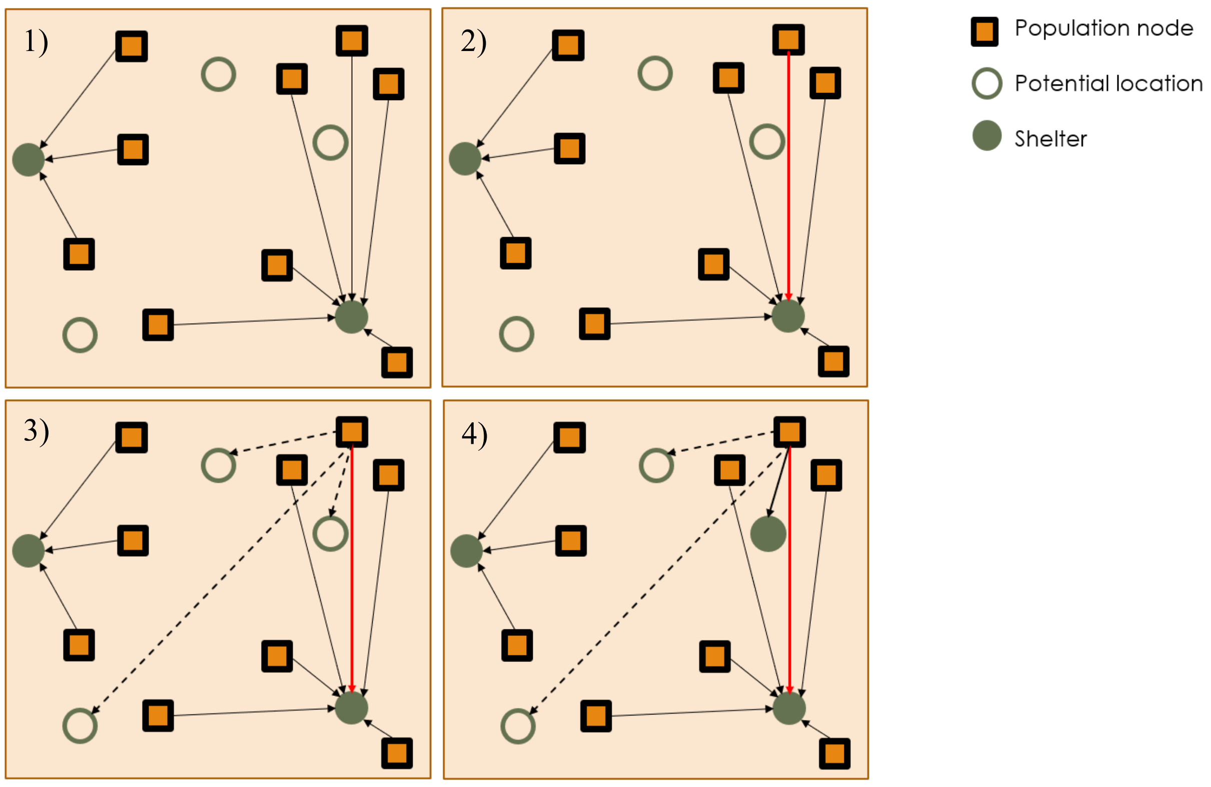


Figure 2-3 P-Center Problem

As the complete enumeration is performed in Figure 2-1 (Two-Stage Model) to get the best solution, all possible combinations are calculated, and the steps shown in the previous figure are a bit different in the implementation.

The already built shelters in 1) represent one permanent combination of the first-stage decision. Instead of following the further steps, the shelters of the first temporary combination of the model are built, and the distances of the population nodes to the nearest shelters are calculated. This calculation is done for each temporary combination separately and then repeated for the next permanent combination. The best solution minimizes the maximum distance.

3. Evaluation Methods

3.1. Conception of Fairness

A study conducted by Mazepus and Leeuwen (2020) shows that people want decision-makers to distribute relief goods fairly, for example by following fair procedures. Contrary to this view there is the concept of utility, first introduced by Bentham in 1781.

In Bentham's work, the utility (e.g. happiness or the prevention of unhappiness) of a community corresponds to the combined utility of all its members. He further writes that an option should be executed if it increases the utility of the community, even if that means that the utility of one or more members decreases.

John Rawls (1957) argues in his paper "Justice as Fairness", that the utilitarian conception cannot be applied to all social issues without complications. He then gives an example that clarifies how differently the two concepts approach the same problem:

- Concept of utility: Slavery should not exist because the gains of the slaveholder are smaller than the losses of the slave and society in general due to an inefficient labor system.
- Concept of fairness: Slavery must be recognized by all involved, otherwise it will be rejected. This leads to an improvement for the people who are worst off, since they do not have to accept a principle that is bad for them. Therefore, the slaveholder's gains are not even considered in this concept (John Rawls, 1957).

The aim of this work is to compare two evaluation approaches, ex-ante and ex-post. The ex-ante evaluation is carried out from the point of view before the disaster occurs, which minimizes the inequity of the expected outcome, while the ex-post evaluation is carried out from the point of view after the disaster has occurred, which minimizes the expected inequity of the outcome (Mostajabdaveh et al., 2020).

Experimental results of Andreoni, Aydin, Barton, Bernheim, and Naecker (2020) give an insight into how people perceive fairness:

- Ex-ante: There are 10 red and 10 blue lottery tickets. After all tickets have been assigned to two households (A and B), one ticket is randomly drawn (future event) and the winner receives some money. A owns all 10 red tickets without earning them. People are then asked how many of the blue tickets they would give to household A or B. Most people give all 10 blue tickets to B to give both households an equal chance of winning, which maximizes the fairness of chances (equal opportunity).

- Ex-post: After the blue tickets have been given to B, a ticket is randomly drawn (past event), and it is blue. People will now be asked if they want to reassign the blue tickets. Most people give half the tickets to A, which maximizes the chances of fairness (equal outcome).

Andreoni et al. (2020) point out that most people, including decision-makers, tend to choose the fairest ex-ante solution ex-ante, but switch to the fairest ex-post solution ex-post. This time inconsistency is a known problem for the ex-ante approach and is discussed in more detail in Gutjahr (2020). Since calculating an optimal ex-ante strategy is pointless if the decision-maker deviates in the second stage anyway, Gutjahr proposes a recursive ex-ante policy to ensure that the strategy is executed. The author bases his policy on the concept of consistent planning which, applied to this model, means that for each scenario, the fairest temporary shelter combination is chosen. Because of this approach, the calculation of the second-stage decision is the same for ex-ante and ex-post.

The results of these two evaluation approaches cannot be directly compared as it depends on the perspective (before or after the event) from which the solution is considered fair (Gutjahr, 2020).

In this thesis a decision is made in a way that the best or "fairest" outcome is achieved for all by minimizing the cost of the worst-off population node, which is done by applying the ex-ante and ex-post approach proposed in Gutjahr (2020). Based on Rawls (1957) conception of fairness, the number of households at each population node is ignored when calculating the ex-ante and ex-post values, because every population node counts the same. However, when the utilitarian values are calculated, the number of households is included to get the utility of the community. In this work, the utility of a population node equals the distance to its nearest shelter multiplied by the number of households.

3.2. Calculation

In the following three subsections the calculations of the ex-ante evaluation, the ex-post evaluation, and the utilitarian objective function are explained. The solutions are calculated from bottom to top. The model used in the figures has two population nodes, two temporary shelter combinations per scenario, two scenarios, and two permanent shelter combinations. It depends on the scenario whether a population node is affected or not. Unaffected population nodes have a cost (Euclidean distance) of 0. To make it easier to find the corresponding part of the code in Appendix A, the numbers of the lines where the code starts are also given.

3.3. Ex-Ante Evaluation

Figure 3-1 shows the calculation of the ex-ante solution.

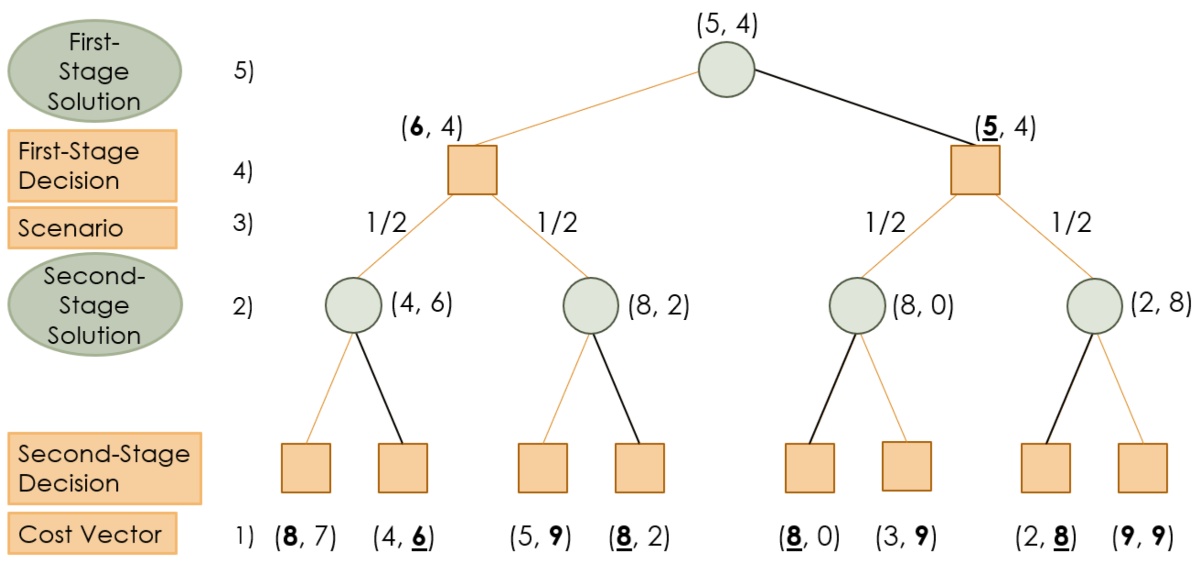


Figure 3-1 Ex-Ante Calculation, adapted from Gutjahr (2020)

1) Lines 42+59: The first step of the ex-ante evaluation is to mark the largest number of each cost vector in bold to obtain the population node of each vector that is worst off. If there are several largest numbers, they are all marked. This means that in the first cost vector 8 is marked, in the second 6, and so on. After that, all bold numbers that belong to the same scenario are compared and the smallest cost is underlined, which minimizes the maximum cost. The first two cost vectors belong to the same scenario. Therefore, 6 is underlined and the second decision node (temporary combination) is chosen. If there are multiple smallest costs, all are underlined and one temporary combination is randomly selected, unlike Gutjahr (2020), who then selects the combination with the better utilitarian value and if they are still equally good, uses lexicographic precedence.
2) Line 127: The next step is to place the entire cost vector containing the underlined number (4, 6) next to the circle and connect the circle with the square. This is now the second-stage solution in case the first permanent combination is chosen and the first scenario occurs.
3) In this model, both scenarios have a probability of occurrence of 50%.
4) Line 137: To obtain the first index of the cost vector of the left decision node (permanent combination), the first index of the left cost vector in 2) is multiplied by the probability of its scenario and then the first index of the right cost vector is multiplied by the probability of its scenario. These two results are then added together. In this example, to obtain the cost

vector (6, 4) of the left decision node, 4 is multiplied by 1/2 and then 8 * 1/2 is added to this result. Thus, the first index (population node) has a cost of 6 and the second index has a cost of 6 * 1/2 + 2* 1/2 = 4.

Lines 150+165: After performing the same calculations for the right decision node, step 1) is repeated, which means that the largest number of each cost vector is marked in bold and then the smallest marked number is underlined.

5) Line 180: The cost vector with the underlined number (5, 4) is placed at the very top of the model. This is the first-stage solution and the edge connecting the circle with the decision node is marked.

Now the solution of the ex-ante evaluation is visible. The first-stage solution is the permanent combination with cost vector (5, 4). The marked edge to the right branch of the model is followed and this permanent combination is built. Which temporary combination is built depends on the scenario that occurs. If scenario 1 occurs, then the left temporary combination with cost vector (8, 0) is chosen and if scenario 2 occurs, then the left temporary combination with cost vector (2, 8) is chosen.
For this solution, the total cost or distance of each scenario is simply the summed cost of each vector in 2), which means 8 for scenario 1 and 10 for scenario 2. This calculation is important for the cost comparisons in the "Results" section.

3.4. Ex-Post Evaluation

Figure 3-2 shows how the ex-post solution is calculated.

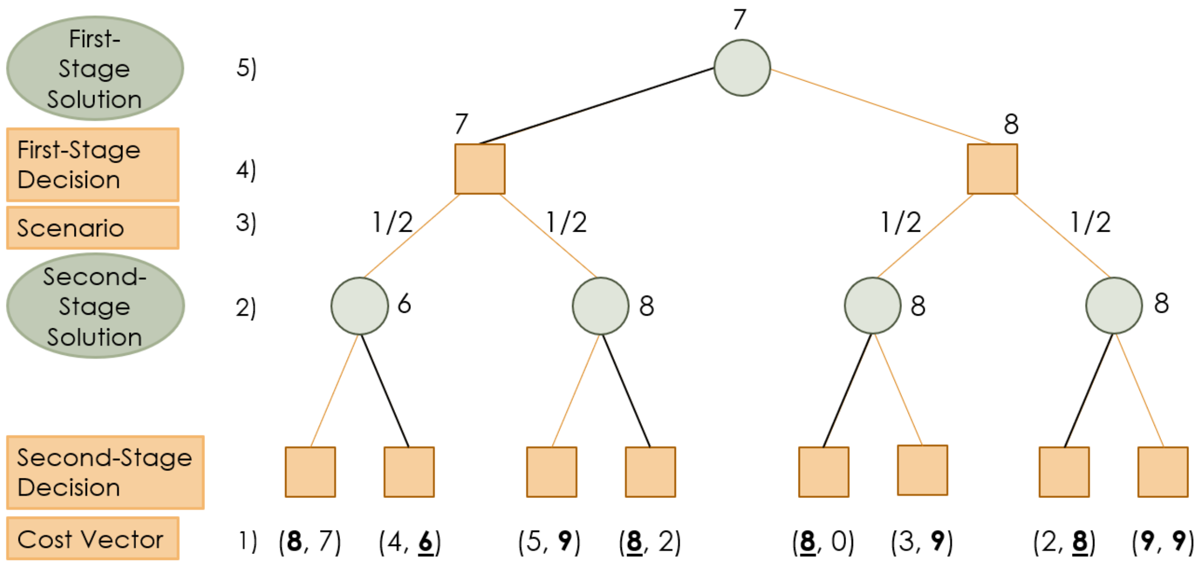


Figure 3-2 Ex-Post Calculation, adapted from Gutjahr (2020)

1) Lines 42+59: The first step of the ex-post calculation is identical to the first step of the ex-ante calculation. First, the highest cost of each vector is marked in bold and then the smallest marked cost of the vectors that are connected to the same scenario is underlined.
2) Line 59: Now, instead of placing the whole vector next to the circle (second-stage solution), only the underlined cost is taken. Again, the edge connecting the circle with the chosen square is marked, and this temporary combination is the second-stage solution.
3) Each scenario has a probability of occurrence of 50%.
4) Line 81: To obtain the value of the left permanent combination, the value of the left second-stage solution is multiplied by the probability of its scenario (1/2) and the value of the right second-stage solution is also multiplied by the probability of its scenario (1/2). These two results are added together. In this case, the calculation is 6 * 1/2 + 8 * 1/2 = 7.
   Line 92: The costs of the permanent combinations are compared and the smallest value (7) is selected.
5) Line 107: This value (7) is placed at the very top of the model (first-stage solution) and the edge connecting the circle with the decision node is marked.

The solution is obtained by following the marked edges. This means that the first-stage solution is the left permanent combination with a value of 7. Now the second-stage solution depends on which scenario occurs. Scenario 1 has a value of 6 and the right temporary combination is chosen. Scenario 2 has a value of 8 and the right temporary combination is chosen.
For this solution, the total cost or distance of each scenario is simply the summed cost of the chosen vector in 1), which is 10 (4+6 and 8+2) for both scenarios. This calculation is important for the cost comparisons in the "Results" section.

3.5. Utilitarian Objective Function

In contrast to the ex-ante and ex-post objective functions that maximize fairness, the utilitarian objective function minimizes total cost. Although the model of Figure 3-3 does not change, the first two steps of calculating the utilitarian solution are quite different compared to ex-ante and ex-post.

1) Line 206: The cost vectors given in the figure are now weighted. This means that the cost vectors of the previous figures, e.g. Figure 3-2, are multiplied by the number of households (weight) that exist at a population node. In this example, population nodes 1 and 2 have a weight of 3 and 2, respectively. The first cost vector of the previous figures is (8, 7). Therefore, the calculations for the first weighted cost vector are:
   3 (weight) * 8 (cost) = 24 and 2 (weight) * 7 (cost) = 14.

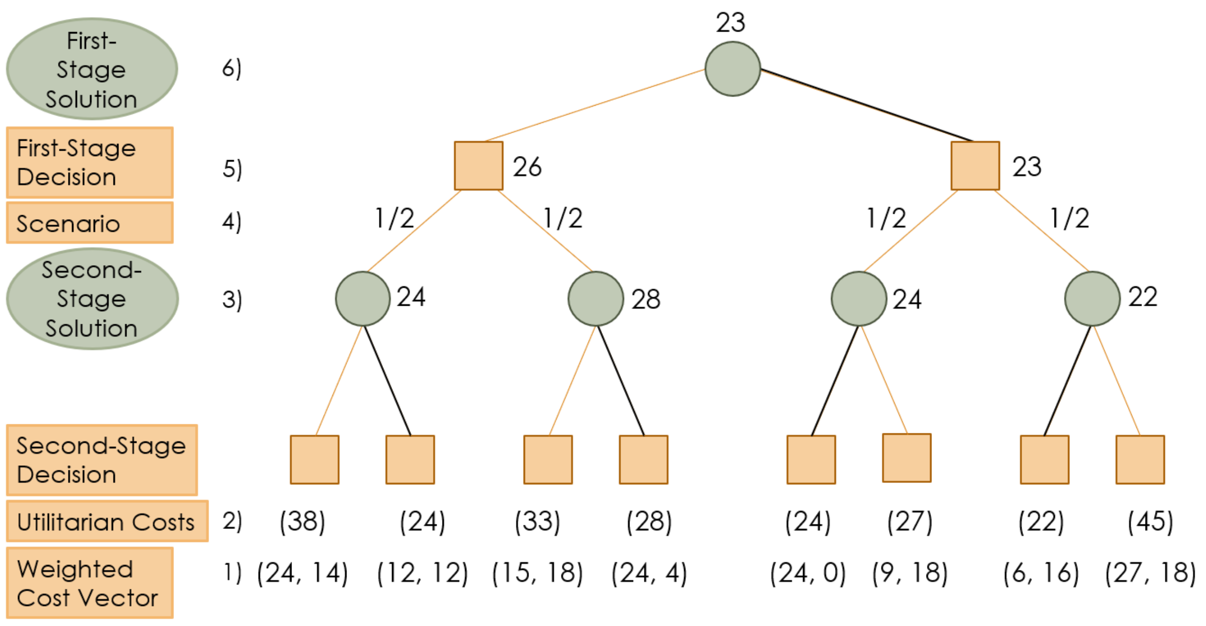


Figure 3-3 Utilitarian Calculation, adapted from Gutjahr (2020)

2) Line 206: To obtain the utilitarian cost of the community, the utilitarian cost of each population node in the weighted cost vector is simply summed.
3) Line 212: Then the utilitarian costs of 2) that belong to the same scenario are compared and the smallest cost is placed next to the circle. This temporary combination is now the second-stage solution and the edge is marked.
4) Both scenarios have a probability of occurrence of 50%.
5) Line 221: This step is identical to step 4) of the ex-post evaluation: To obtain the value of the left decision node (permanent combination), the values of the left and right scenarios that are connected to this decision node are multiplied by 1/2 (probability of occurrence). These two results are then added together. In this case, the calculation is 24 * 1/2 + 28 * 1/2 = 26.
6) Line 226: The last step is to compare the values, take the smallest one, and put it at the top of the model. This is now the first-stage solution. Then the edge that connects the circle with the permanent combination is marked.

The utilitarian solution is obtained by following the marked edges. For this example, this means that the right permanent combination is built with utilitarian costs of 23. When scenario 1 occurs, the left temporary combination is chosen leading to total utilitarian costs of 24, and when scenario 2 occurs, the left temporary combination is chosen leading to total utilitarian costs of 22.

4. Complete Enumeration

When the problem shown in Figure 4-1 is solved by complete enumeration (CE), each branch of the model is computed and the best solution is chosen. This calculation can take decades or even longer for large problems (see section 6.5.2 “Time Measurement”). For this reason, only a small problem is solved by complete enumeration and the solutions of the larger problems are obtained by a genetic algorithm.

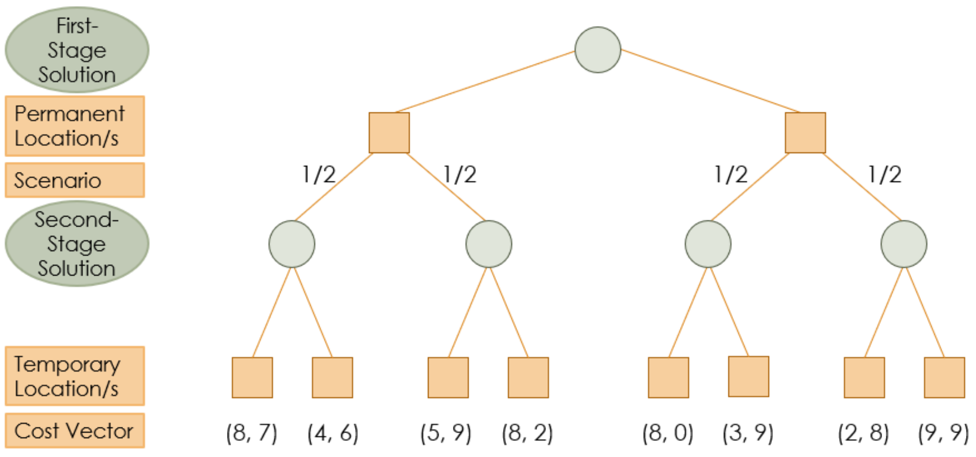


Figure 4-1 Complete Enumeration, adapted from Gutjahr (2020)

The complete enumeration algorithm is explained in Figure 4-2 via pseudocode and the implementation of this pseudocode is given in Appendix A. To make it easier to find the corresponding part of the code in the appendix, the numbers of the lines where the code starts are also given.

1 Lines 6+17: First, all possible permanent combinations are calculated. The underlying mathematical formula of this function is the binomial coefficient

$$(1) \quad \binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}$$

where n represents the number of potential locations and k represents the number of permanent shelters that must be built. This formula means that choosing the same location more than once is not allowed and the order in which the locations are selected is not important (Weisstein, n.d.).

2 Lines 6+17: Then, all possible temporary combinations are calculated for each permanent combination, using the same formula as before. This time, n represents the number of potential locations that are not already used for permanent shelters, and k represents the number of temporary shelters that must be built.

```
1   determine all possible permanent combinations
2   determine all possible temporary combinations
3   for all possible permanent combinations
4       for all scenarios
5           for all possible temporary combinations
6               calculate cost vector
7           determine fairness of temporary combinations
8           find fairest temporary combination
9   calculate fairness of permanent combinations
10  find fairest permanent combination
```

Figure 4-2 Pseudocode Complete Enumeration

4 The scenarios and their probabilities of occurrence, as well as the affected population nodes, are either manually selected or randomly generated before the algorithm starts.

6 Line 22: The cost vectors of all permanent combinations, scenarios, and temporary combinations are calculated. To compute the cost of each affected population node, the Euclidian distance to the nearest permanent or temporary shelter is determined.

7 Lines 42: The fairness of each temporary combination is calculated regardless of whether the ex-ante or ex-post evaluation approach is used.

8 Line 59: In this step, the fairest temporary combination of each scenario is found.

9 Lines 137 (ex-ante) + 81 (ex-post): For each permanent combination, the fairness value is calculated depending on the selected evaluation approach.

10 Lines 150+165 (ex-ante) + 81 (ex-post): Depending on the selected evaluation approach, the fairest permanent combination is chosen.

Lines 180 (ex-ante) + 107 (ex-post): The algorithm then has all the relevant data for the solution, which consists of the fairest permanent combination and, for each scenario, one (the fairest) temporary combination.

5. Genetic Algorithm

5.1. Introduction

At the time Charles Darwin published his abstract “On the origin of species” in 1859, most naturalists still believed that every species was created independently from one another and that parts considered important for a species do not vary for each individual member of that species. Darwin observed that in fact even important parts do vary slightly and if this variation is an advantage for the individual in relation to current living conditions, then it is more likely to survive and consequently has a higher chance of being naturally selected for reproduction. This advantageous variation is then passed on to its offspring which again will have a higher chance to survive (Darwin, 1859).

The genetic algorithm is based on Darwin’s theory of evolution and was first introduced in the 1960s by John Holland with the aim of studying natural adaptation (Mitchell, 1998). In his book “Adaptation in Natural and Artificial Systems”, he proposed a mathematical framework to generalize factors of adaptive processes (Holland, 1992). He notes that adaptive processes are important to several computational fields such as psychology (learning), artificial intelligence, and economics (optimal planning). When Holland first published his book in 1975, it did not receive much attention. However, this changed rapidly about ten years later, when learning was again seen as the key to artificial intelligence, and because genetic algorithms are capable of solving problems that could not be solved by standard approaches (Holland, 1992).

Genetic algorithms (GAs) belong to a field called evolutionary computing and, like other metaheuristics, are often used for optimization purposes (Reeves, 2003). Although (meta)heuristics do not necessarily deliver optimal solutions (Reeves, 2003), metaheuristics do provide robust results and can be applied in many different fields (Hussain, Salleh, Cheng, and Shi, 2019).

Since computing all possible combinations by complete enumeration heavily affects the computation time and is therefore “foredoomed in all but the simplest cases” (Holland, 1992), the first-stage decision is now replaced by a genetic algorithm as shown in Figure 5-1. The solution of the second-stage decision is still determined by complete enumeration.

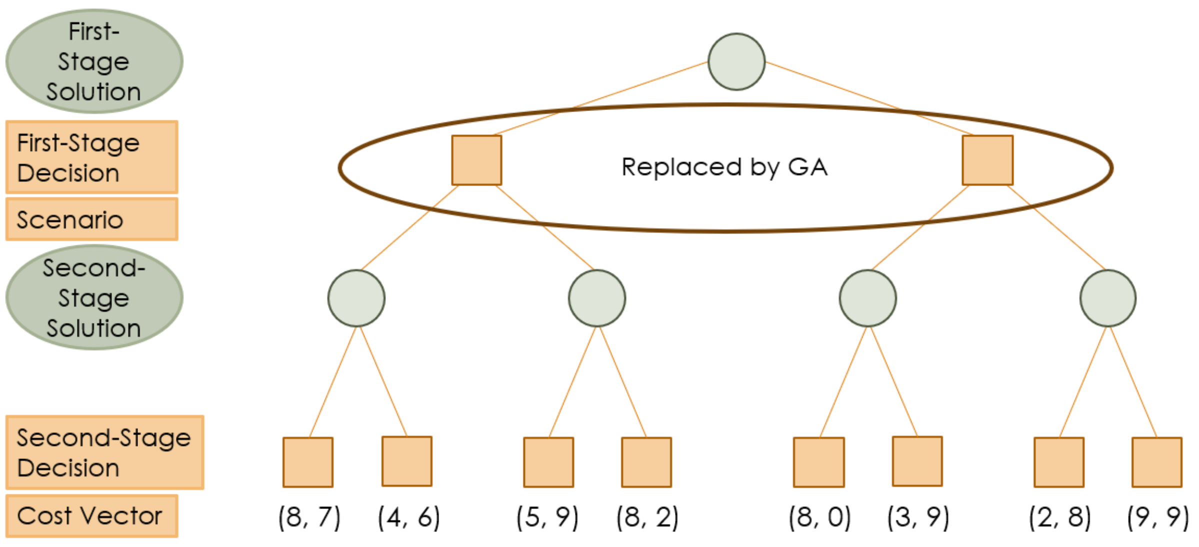


Figure 5-1 Genetic Algorithm Model, adapted from Gutjahr (2020)

In the next subsections, first a brief overview of the structure of the genetic algorithm used in this work is given and then each part is explained in more detail.

5.2. Structure

Figure 5-2 contains the pseudocode of the implemented genetic algorithm. The implementation of the genetic algorithm is given in Appendix B, and the numbers of the lines where each part of the code starts are given. The fitness values are calculated by a slightly adapted version of the complete enumeration algorithm and the code is therefore not included again.

```
1    create initial population
2    calculate fitness
3    for number of generations/iterations
4        select parents
5        perform crossover
6        apply repair mechanism
7        perform mutation
8        calculate fitness
9        if termination criterion is met
10           get best chromosome
```

Figure 5-2 Pseudocode Genetic Algorithm

1 Line 6: The first function creates the initial population. A population consists of a chosen number of individuals called “chromosomes”.

2 Then each of the chromosomes is evaluated by calculating its fitness value. If a chromosome has a high fitness value relative to the other chromosomes it means that this chromosome is a good solution to the given problem.

3 At each iteration of the algorithm, a new population, also called “generation”, is delivered.

4 Line 23 or 58: Each generation starts with selecting and pairing up the parent chromosomes. This selection is based on the fitness values calculated either at the beginning of the algorithm (step 2), or in step 8 of the previous generation.

5 Lines 94+114 or 100+136: The next step is to perform the crossover, which basically means that one part of the chromosome of the first parent and a different part of the chromosome of the other parent are joined together and a new chromosome, called “child”, is created. These children replace the old population.

6 Line 162: As the crossover could cause problems in the form of duplicate permanent shelters, a repair mechanism must be used to fix invalid chromosomes.

7 Line 204: A mutation function is implemented that replaces a permanent shelter with a potential location since diversity is crucial for this kind of heuristic.

8 Fitness values are calculated for the new generation, to either select the parents in the next generation or, if the

9 termination criterion of the algorithm is met,

10 determine the current best individual.

5.3. Chromosome Representation

Before the initial population can be created, a decision must be made about the representation of the chromosome. A chromosome is represented by a string and a string consists of several elements called “genes” that can take on values called “alleles” (Reeves, 2003).
Holland (1992) claims that from a computational point of view, many genes with few alleles are preferred to few genes with many alleles, which means that a binary representation is usually preferred to an integer representation. Reeves (2003) states that some problems are naturally integer or real-valued and must be transformed into a binary string, while the chromosomes of other problems, such as the travelling salesman problem, are defined via a permutation. An example of such a permutation is given by the author: The solution (653478) specifies the order in which jobs are completed on a machine: Job 6 is done first, then job 5, and so on.

As mentioned in the introduction of this section, the first-stage decision is replaced by a GA, which means that the chromosomes in this work contain the locations where the permanent shelters are built. Each chromosome contains one permanent combination. For this type of problem there are two obvious representations of the chromosomes, binary and integer.

Figure 5-3 shows these two representations and both store the same information. In the given example, five potential locations are available and two permanent shelters are built.

The binary chromosome consists of five genes, one for each potential location. Each gene has two alleles, 0 and 1. When a shelter is built, the gene is given the value 1, otherwise 0. Here, permanent shelters 1 and 4 are built. In the code, the chromosome is stored in a vector or array.

The integer chromosome is much shorter and the genes contain the permanent shelters to be built. Each gene has five alleles, one for each potential location. Again, permanent shelters 1 and 4 are built.

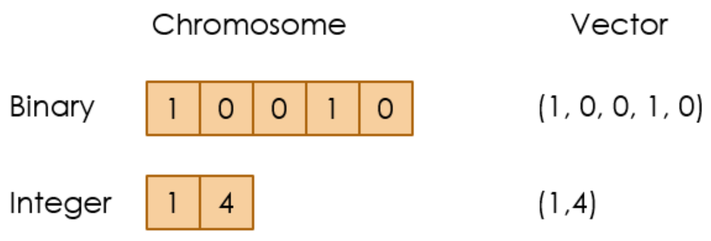


Figure 5-3 Binary and Integer Chromosomes

Now the best representation for this problem has to be determined. To do this, both representations are implemented and tested in terms of performance (in microseconds). The results are compared, and the faster representation is chosen.
For this purpose, only very simple versions of the functions (e.g. initial population, mutation, selection, …) are used in the code. The fitness function is the same for both representations and is therefore omitted. Two different problem sizes, given in Table 5-1, are generated completely randomly and used for testing.

	Permanent Locations	Temporary Locations	Potential Locations	Population Nodes	Chromosomes
Small Problem Size	2	2	5	10	20
Large Problem Size	3	3	10	30	100

Table 5-1 Problem Sizes

Since the runtime varies due to the scheduling algorithm of the operating system (Walls, 2016), and also depends on the selected seed, it is very likely that the measured time of each run is slightly different. To obtain more stable results, each problem size is run with 100 seeds and the arithmetic means are given in Table 5-2.

	1) Small Problem Size			2) Large Problem Size		
	Binary	Integer	%Δ	Binary	Integer	%Δ
1 Initial Population	321.60	308.76	-3.99	1940.27	1904.65	-1.84
2 Mutation	77.34	84.17	8.83	694.65	960.62	38.29
3 Selection	16.24	16.85	3.76	60.35	63.30	4.89
4 Crossover	314.47	260.06	-17.30	1976.58	1281.20	-35.18
5 Cross. Repair	143.38	82.06	-42.77	1277.04	692.64	-45.76
Total Runtime	873.03	751.90	-13.87	5948.89	4902.41	-17.59

Table 5-2 GA: Performance Comparison in Microseconds

The performance of the mutation operator of the small problem size is 8.83% worse for the integer chromosome (IntC) compared to the binary chromosome (BinC) and it becomes a lot worse when the IntC and the BinC of the large problem size are compared. Since the mutation operator works directly with the alleles, it seems that Holland's (1992) claim applies and that indeed a long string with few alleles is faster than a short string with many alleles.

However, this does not apply to the implemented crossover operator since it only stores the elements of the parent vectors in the vectors of the children and does not change any alleles. Here, the long binary string has strong disadvantages compared to the short integer string, which worsen with increasing problem size.

Overall, the total runtime of the IntC is faster than the BinC by 13.87% for the small problem and by 17.59% for the large problem. Therefore, an integer representation is chosen for this thesis.

5.4. Initial Population

Reeves (2003) argues that the number of chromosomes or size of the initial population should be chosen in a way that the search space can be explored effectively, but also efficiently in terms of computation time. The author further states that the initial population is usually randomly generated. However, reports show that the GA can find better solutions if a good solution from another heuristic is included in the initial population (Reeves, 2003).

In this work, the chromosomes are randomly generated as explained with the pseudocode in Figure 5-4. The code is available in Appendix B, line 6. A vector is created (2) that stores the numbers of all the enumerated potential locations (3). Then a random location is selected from this vector (5), added to the chromosome (6), and deleted from the vector (7). This is done for the number of permanent shelters to be built (4) and repeated for the number of chromosomes in the population (1). Since the chromosomes are randomly generated, it is possible that the best solution is already present in the initial population, or that some of the chromosomes are identical.

```
1   for the number of chromosomes
2       create vector for potential locations
3       enumerate locations and store numbers in vector
4       for the number of permanent shelters
5           get a random location from the vector
6           add this location to the chromosome
7           delete this location from the vector
```

Figure 5-4 Pseudocode Initial Population

The size of the initial population of this implementation is rather small and usually has about 20 chromosomes. A larger population would not be a problem as far as the computation time is concerned, but then it is quite likely that the best solution is already generated in the initial population since the chromosomes have few genes (integer representation). When the best solution is present in the initial population, a comparison of the number of generations needed to obtain the best result for ex-ante and ex-post would not be possible.

5.5. Fitness Values and Diversity

Fitness values are needed for the selection operator that chooses individuals, called parents, for mating (Vidal, Crainic, Gendreau, Lahrichi, and Rei, 2011). Vidal et al. (2011) explain that a fitness value is assigned to each individual in the population and this value shows how well an individual performs relative to the other individuals in that population.
Although the authors emphasize that diversity is crucial for this field of metaheuristics and therefore propose a fitness function based on the cost and diversity of an individual to maintain variation, the experimental results of Bersano-Begey (1997) show that including factors other than just the value of the individual itself in the fitness function does not necessarily lead to better solutions. He states that, even though his initial approach helped the GA avoid local optima, it did not find good solutions, probably because the fitness value assigned to an individual differed a lot from its true value. Nevertheless, the author is convinced that variation is important because it helps avoid local optima and early convergence.

The results of Bersano-Begey (1997) suggest that if additional factors are included in the fitness function, they should be used with caution. Therefore, in this thesis, only the costs/distances are considered in the fitness function and other methods (survivors in the crossover function and constraints in the selection function) are used to maintain diversity. Survivors are individuals of the current population that are directly transferred to the new generation (Vidal et al., 2011).

```
1   determine all possible temporary combinations
2   for all chromosomes
3       for all scenarios
4           for all possible temporary combinations
5               calculate cost vector
6           determine fairness of temporary combinations
7           find fairest temporary combination
8   calculate fairness of permanent combinations
9   find fairest permanent combination
10  calculate reciprocal of chosen combination
```

Figure 5-5 Pseudocode GA Fitness

As already mentioned in the introduction of this section, only the first-stage decision is replaced by the genetic algorithm, but the second-stage decision of each chromosome is still computed by complete enumeration. This means that the fitness value of each chromosome is calculated by executing all the functions of the complete enumeration algorithm as shown in Figure 5-5. The major difference is that now the chromosomes replace the possible combinations of permanent shelters (Figure 4-2, line 1).
Instead of fairness values, which are equal to costs and minimized in this work, fitness values are calculated. The fitness value of an individual is equal to the reciprocal of its cost (1/cost) since the fitness function is maximized. As the fitness function is an only slightly adapted version of the complete enumeration algorithm, the code is not included.

5.6. Selection

Individuals from the current population are selected, put into a mating pool, and used for breeding (Miller & Goldberg, 1995). As it is important to have “good” individuals in the mating pool, the individuals with a higher fitness value have a higher chance of being selected (Miller & Goldberg, 1995). The selection pressure, that determines how strongly the fitter chromosomes are favored, must be chosen carefully since a selection pressure that is too low will take a long time to find a good solution and a selection pressure that is too high delivers a sub-optimal solution more easily (Miller & Goldberg, 1995).

Two of the most commonly used selection methods are the roulette wheel and the tournament selection (Jebari & Madiafi, 2013). These two methods are implemented in the code and are therefore explained in more detail.

For the roulette wheel selection, a probability distribution is used, and for each individual, the probability of being selected depends on its fitness value (Reeves, 2003). Figure 5-6 illustrates the general idea of this method using the data provided in Table 5-3. Six individuals, their percentages, and the cumulated percentages are given in the table. The percentages are calculated by dividing the fitness of an individual by the sum of all fitness values of the whole population and multiplying the result by 100. The greater the selection probability of an individual, the larger its area on the roulette wheel (Reeves, 2003). This is illustrated in the figure: Individual B has the largest percentage and thus the largest area on the wheel.

Now a random number [0, 100] is generated and the individual in whose area this number lies is put into the mating pool. For example, if the random number is 60, then individual C is selected.

Individuals	Percentages	Cumulated Percentages
A	12.5	12.5
B	34.5	47.0
C	15.5	62.5
D	12.5	75.0
E	18.0	93.0
F	7.0	100

Table 5-3 GA: Percentages of Individuals

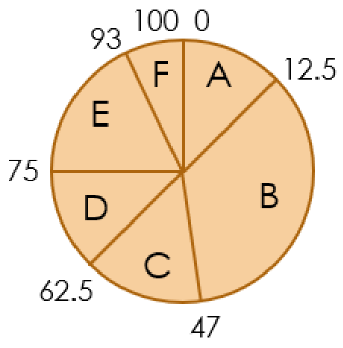


Figure 5-6 Roulette Wheel Selection

In the tournament selection method, a tournament is held among a chosen number of individuals. The individual with the best fitness value wins the tournament and is put into the mating pool (Miller & Goldberg, 1995). The number of participants in the tournament has a large impact on the selection pressure, as the fitness of the winner is on average higher the more individuals are competing (Miller & Goldberg, 1995).

Figure 5-7 illustrates the tournament selection process. Here, three individuals (A, D, and E) are randomly selected to participate in a tournament. Using the percentages from Table 5-3 as fitness values, the values for individuals A, D, and E are 12.5, 12.5, and 18, respectively. This means that individual E wins the tournament and is put into the mating pool.

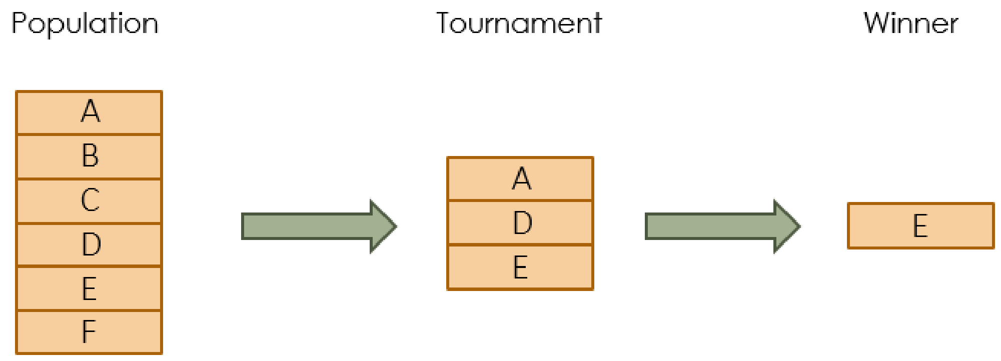


Figure 5-7 Tournament Selection

As already mentioned in the previous subsection, diversity is ensured in the selection functions. In the roulette wheel selection function, given in Figure 5-8, diversity is maintained in the first five lines. The code can be found in Appendix B, line 23.

```
1   put worst and best chromosome into mating pool
2   for chosen number of chromosomes with low fitness
3       get a random chromosome and put it into mating pool
4   for chosen number of chromosomes with low or medium fitness
5       get a random chromosome and put it into mating pool
6   sum up all fitness values
7   divide each fitness value by this sum
8   store cumulated percentages in a vector
9   for each remaining slot of the mating pool
10      get random real number [0, 1]
11      check cumulated vector and get respective chromosome
12      put chromosome into mating pool
```

Figure 5-8 Pseudocode Roulette Wheel Selection

In line 1 of the figure, the worst and the best chromosome are put into the mating pool. These chromosomes are later put directly into the new generation by the crossover algorithm and are therefore called survivors. Then a predefined number of chromosomes with low fitness is put into the mating pool (2). In the implementation, “low” corresponds to the worst 30% of chromosomes. Afterwards, again a predefined number of chromosomes with low or medium fitness is put into the mating pool (4). “Low or medium” corresponds to the worst 60% of chromosomes in the code. Both percentages were tested and seem to work well for this problem.

In line 6, the roulette wheel selection starts. First, the shares or percentages of the chromosomes on the roulette wheel are calculated. This is done by summing up all the fitness values. Then each of the values is divided by this sum to get the percentage of each individual (7). After that, the percentages are cumulated and the results are stored in a vector (8). The first element of the vector is 0 and the last element is the sum of all percentages and must be 1.
Now the empty slots of the mating pool are filled up (9). A random real number [0, 1] is generated (10) and the chromosome in whose share the random number lies wins (11+12). The same chromosome can win several times.

Figure 5-9 shows the pseudocode of the tournament selection. The first five lines ensure diversity and are identical to the first five lines of the roulette wheel selection function. The code is given in Appendix B, line 58.

```
1    put worst and best chromosome into mating pool
2    for chosen number of chromosomes with low fitness
3        get a random chromosome and put it into mating pool
4    for chosen number of chromosomes with low or medium fitness
5        get a random chromosome and put it into mating pool
6    for each remaining slot of the mating pool
7        store enumerated chromosomes in a vector
8        for the number of tournament participants
9            get a random chromosome from the vector
10           put it in the tournament
11           delete this chromosome from the vector
12       for each tournament participant
13           compare fitness values of participants
14       put winner into mating pool
```

Figure 5-9 Pseudocode Tournament Selection

The tournament selection starts with line 6. For each empty slot in the mating pool, the chromosomes are enumerated, and these numbers are stored in a vector (7). Then, for the predefined number of tournament participants (8), a random chromosome from the vector (9) is put into the tournament (10) and deleted from the vector (11). The fitness values of all participants are compared (12+13) and the chromosome with the highest value is put into the mating pool (14). The same chromosome can win more than once.

5.7. Crossover

The crossover operator creates, in general, one or more children (offspring) from two parents (Reeves, 2003). Several crossover approaches are given in Reeves (2003) and four of them, the one-point, two-point, m-point, and completely random crossover, are discussed in the following paragraphs.

Figure 5-10 illustrates the one-point and two-point crossover. Each gene represents a potential location where a permanent shelter is built. "X" indicates the position where the two parent chromosomes are crossed over. This position is usually chosen randomly (Reeves, 2003).

First, the one-point crossover is explained. Here, one crossover position is chosen randomly, and the results are child 1 and 2. Child 1 receives the first two genes (up to the crossover position) from parent 1 and the last three genes from parent 2. Vice versa for child 2.

The two-point crossover follows the same concept with the difference that now two crossover positions are chosen randomly as shown in the figure. Child 1 thus receives the first gene from parent 1, the second and third from parent 2, and the last two again from parent 1.

The resulting new individuals (children) replace the old population (Miller & Goldberg, 1995).

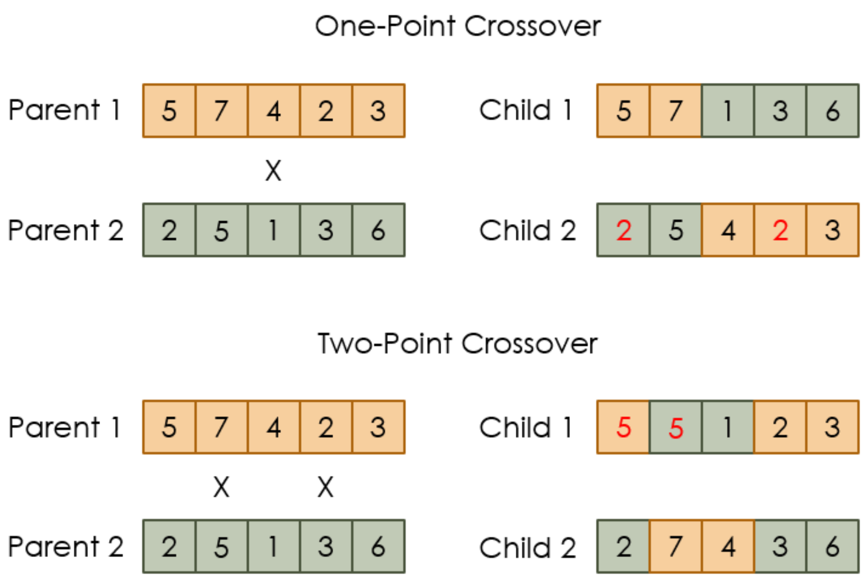


Figure 5-10 One-Point and Two-Point Crossover

Reeves (2003) states that any number of crossover positions (m-point crossover, where m > 1) can be chosen. He further mentions a completely random crossover, called uniform crossover, where the operator is implemented as a binary string, e.g. (1, 0, 0, 1, 1, 1). This string means that the second and third genes are taken from the second parent and the other genes are taken from the first parent, which is equivalent to a two-point crossover in this case.

The red numbers in Figure 5-10 indicate duplicate locations, which is forbidden in this type of problem because there is no value in building two shelters at the same location. These infeasible chromosomes need to be repaired. Pongcharoen, Khadwilard, and Hicks (2008) state that repair mechanisms are problem specific because the algorithm is different for each problem, e.g. binary or integer chromosomes and existing constraints.
An interesting approach is used by Mostajabdaveh et al. (2020), who implement three different crossover operators and choose one at random when a crossover is required.

In this work, a one-point and a two-point crossover operator are implemented, run separately, and the results are compared. Since the crossover mechanism may lead to infeasible solutions, the applied repair mechanism is also explained.
Figure 5-11 shows the pseudocode of the one-point crossover. The code is available in Appendix B, lines 94+114. First, the crossover position is randomly chosen for each chromosome pair. As already mentioned in the previous subsection, survivors (the best and the worst chromosome) are put directly into the new generation without being changed (3).
In line 4 of the figure, the crossover is performed for all pairs of chromosomes. A pair consists of two chromosomes (parents) that are directly adjacent in the mating pool. Two children (offspring) are generated from each pair. The first child receives all genes up to the crossover position from the first parent (6) and the remaining genes from the second parent (9). The reverse is the case for the second child.
In the last line, the children (new generation) replace the old generation.

```
1     for each chromosome pair
2         randomly choose crossover position
3     put survivors directly into new generation
4     for all chromosome pairs
5         for all genes up to the crossover position
6             first child gets the gene of the first parent
7             second child gets the gene of the second parent
8         for all remaining genes
9             first child gets the gene of the second parent
10            second child gets the gene of the first parent
11    new generation replaces old generation
```

Figure 5-11 Pseudocode One-Point Crossover

Almost the same steps are executed for the two-point crossover, given in Figure 5-12. The code can be found in Appendix B, lines 100+136. Again, the crossover positions are chosen (2) and the survivors are put into the new generation (3). But now, the first child gets the first (6) and last (12) sequence of genes from the first parent and only the middle sequence from the second parent. The reverse is true for the second child.
The old generation is again replaced by the new generation.

```
1    for each chromosome pair
2        randomly choose crossover position
3    put survivors directly into the new generation
4    for all chromosome pairs
5        for all genes up to the first crossover position
6            first child gets the gene of the first parent
7            second child gets the gene of the second parent
8        for all genes between the first and the second crossover position
9            first child gets the gene of the second parent
10           second child gets the gene of the first parent
11       for all remaining genes
12           first child gets the third part of the first parent
13           second child gets the third part of the second parent
14   new generation replaces old generation
```

Figure 5-12 Pseudocode Two-Point Crossover

After the crossover is performed, a repair mechanism is applied to check if each solution is feasible. This repair function is given in Figure 5-13, and begins at line 162 in Appendix B. The function searches for one or more duplicate locations in line 2 of the figure. If there are duplicates (3), then the function finds the affected locations (4).
For the number of duplicate locations (5), the still available locations are stored in a vector (6). Then a random duplicate location is selected (7) and replaced by a random available location from the vector (8+9).

```
1   for all chromosomes
2       check if there are duplicate locations
3       if there are duplicates
4           find affected locations
5           for the number of duplicate locations
6               store still available locations in vector
7               get random duplicate locations
8               get random available location
9               replace duplicate with available location
```

Figure 5-13 Pseudocode Repair-Mechanism

5.8. Mutation

Most authors point out that mutation is necessary to prevent the algorithm from getting stuck in a local optimum by keeping the population diverse (Reeves, 2003). Experimental results of Grefenstette (1986) even indicate that the performance of algorithms is worse when mutation is not implemented as lost values are not recovered.
Reeves (2003) lists different approaches that are used to implement mutation: One approach is to generate a random number for each gene of each chromosome and compare it to a mutation rate. The drawback of this idea is that for long chromosomes/strings and a large population, the computation time could be strongly affected. A more efficient approach for a long string is to use a Poisson distribution with $\lambda=1$ to get a random variable that decides how many genes are mutated. Then a uniform distribution is used to determine which genes are mutated.

When a gene of a binary chromosome is mutated, it is simply flipped e.g. from 0 to 1, and for non-binary strings, a value can either be chosen randomly from a pool or in case there are ordinal relations between the genes, constraints or biased probabilities can be implemented (Reeves, 2003).
In the implementation, a random number is generated for each gene since the chromosomes are usually very short and the population size is small. Furthermore, the mutation rate is set to 5%, which according to Grefenstette (1986) is the maximum percentage that does not harm the performance.
The pseudocode of the implementation is given in Figure 5-14, and the code in Appendix B, line 204. For all chromosomes except the last one (1) and all permanent shelters/genes (2), a random number [0, 99] is generated (3). If this number is smaller than the mutation rate (4),

then the gene is mutated. The last chromosome is not changed because the current best solution is stored at this position and excluding it from possibly being mutated ensures that the overall highest fitness value does not get worse.

The process of mutating a gene is as follows: First, a vector is created to store potential locations (5). Afterwards, for all locations (6), it is checked if this location is already present in the chromosome and if this is not the case (7), then it is stored in the vector (8). A random location is selected from the vector (9) and this new location replaces the old permanent shelter (10).

```
1    for all chromosomes except the last
2        for all permanent shelters
3            generate random number [0, 99]
4            if number smaller than mutation rate
5                create vector for potential locations
6                for all locations
7                    if location does not exist in chromosome
8                        add location to vector
9                get random location from vector
10               replace old location with new random location
```

Figure 5-14 Pseudocode Mutation

5.9. Termination

If no termination criterion is implemented, GAs would never stop running (Reeves, 2003). Reeves (2003) lists some common approaches: Terminate the algorithm after a certain number of generations, a preset time, or when the diversity of the population falls below a threshold. Another approach is to stop the algorithm when no improvement is achieved after a preset number of iterations (Leung & Wang, 2001). Hedar, Ong, and Fukushima (2007) claim that these standard termination criteria are a drawback as they are set by the user. Therefore, the authors propose a method where the GA itself decides when to terminate the algorithm.

In Mostajabdaveh et al. (2020), two termination criteria are combined, which is also done in this work. The algorithm is stopped either after 50 generations without improvement of the best chromosome or after a total of 300 generations. These numbers have been tested and seem to work well for this algorithm.

6. Results

6.1. General Settings and Overview

In each of the subsections, the ex-ante and ex-post fitness values, the costs, and the utilitarian costs are discussed. Note that the fitness values cannot be directly compared due to the different objective functions. As it is interesting to know how well the best ex-ante/ex-post solution performs in terms of fitness values when plugged into the ex-post/ex-ante objective function, the percentage difference is also calculated in each subsection. The same is done for the utilitarian costs: the best ex-ante/ex-post solution is plugged into the utilitarian objective function and the percentage difference is compared. The underlying data (e.g. the coordinates of the population nodes) of both problem sizes is completely randomly generated. The algorithms are implemented in C++ and run on Windows 10, using an Intel Core i7-7700HQ and 8 GB RAM.

Subsection 6.3.1.1 (“Increasing Population Nodes”) first shows and explains the results of a single seed and compares these results to the arithmetic mean of 100 randomly generated seeds. A single seed is discussed only in that subsection to keep the thesis at a reasonable length. For all other subsections the arithmetic means of 100 seeds are used. Additionally, mathematical formulas, that apply to all other subsections, are given and explained in this subsection. All results are rounded to two decimal places and the performance is measured in milliseconds.

After discussing the solutions of the complete enumeration and the genetic algorithm of the small problem size (6.3 “Small Problem Size”), the genetic algorithm is applied to a large problem (6.4 ”Large Problem Size”) and different combinations of selection and crossover algorithms are compared.

The next subsection (6.6 ”Case Study: Nepal”) focuses on a case study of Nepal. Based on probabilistic seismic hazard assessment and real data of major cities in central Nepal, the genetic algorithm is used to find a solution in case a disaster occurs. This solution is then compared to the data of a real disaster, an earthquake that occurred in Nepal in April 2015.

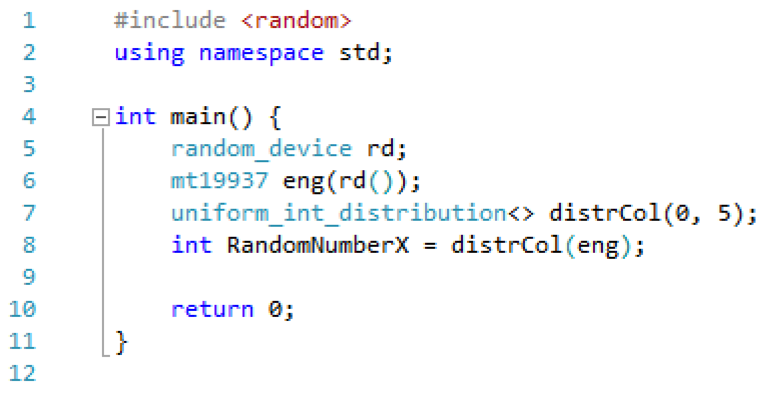
A budget is introduced in the last subsection (6.7 ”Budget”). It is investigated how a different number of permanent and temporary shelters with the same budget affects the fitness values. This is important for decision-makers because temporary shelters are built where they are really needed on the one hand, but on the other hand they are much more expensive than permanent shelters because they have to be available as soon as possible.

6.2. Input Data and Randomness

Most of the required input data is either manually selected or randomly generated. It is also possible to import latitude and longitude of population nodes and the number of households from an Excel file.

Since randomness is an important part of the code, it is briefly discussed in this subsection. According to Steve Ward, professor at MIT, deterministic algorithms produce identical results every time they are executed with the same seed (Rubin, 2011). He further states that the numbers generated are not truly random and are referred to as pseudo-random numbers. These numbers are sufficient for most purposes and are therefore used in this thesis. Truly random numbers are crucial when it matters that the outcome cannot be predicted, for example on an online poker site (Rubin, 2011).

Detailed information about the following code can be found on the cplusplus website (The C++ Resources Network, n.d.). The standard library added to the code is called <random>. Two uniform distributions, the uniform discrete distribution and the uniform real distribution, are used. These distributions use a generator to create random numbers. The generator chosen is called “Mersenne twister random number engine”, which creates pseudo-random numbers by plugging a seed into an algorithm. This seed is produced by a generator called random_device, which creates non-deterministic random numbers.



```
1     #include <random>
2     using namespace std;
3
4     int main() {
5         random_device rd;
6         mt19937 eng(rd());
7         uniform_int_distribution<> distrCol(0, 5);
8         int RandomNumberX = distrCol(eng);
9
10        return 0;
11    }
12
```

Figure 6-1 Generating Random Numbers

A small example is given in Figure 6-1. In line 5, a random number rd is generated, which is used to seed the Mersenne Twister engine in line 6. A huge series of numbers can be generated by this engine’s algorithm. Then the uniform discrete distribution is created with range [0, 5]. Line 8 shows how a random integer number of range [0, 5] is produced.

6.3. Small Problem Size

Table 6-1 shows the default settings of the small and large problem sizes. For each problem size, the number of permanent shelters, temporary shelters, potential locations, and population nodes are independently increased by a constant number after each run and presented in separate subsections. For example, to observe how additional permanent shelters affect the result, only the permanent shelters are increased but the other settings remain constant. Table 6-2 contains the additional settings necessary for the genetic algorithm.

	Permanent Locations	Temporary Locations	Potential Locations	Scenarios	Population Nodes
Small Problem Size	2	2	8	3	10
Large Problem Size	3	1	50	2	20

Table 6-1 Default Settings

	Chromosomes	Generations	Mutation Rate
Small Problem Size	10	300	5
Large Problem Size	20	300	5

Table 6-2 Additional Settings for GA

6.3.1. Complete Enumeration

Although the complete enumeration uses fairness values (based on distances or costs), these values are changed to fitness values (1/fairness) in all "Complete Enumeration" subsections to facilitate the comparison with the results of the genetic algorithm. Therefore, the objective function of the CE is no longer minimized (costs), but maximized (fitness).

6.3.1.1. Increasing Population Nodes

6.3.1.1.1. Fitness Values and Performance

6.3.1.1.1.1. Single Seed

In this subsection the population nodes increase by a constant number of additional nodes after each run, while all other settings stay the same. Table 6-3 shows the fitness values of the best ex-ante and ex-post solutions. Ex-ante fitness values λ_A cannot be directly compared with the ex-post fitness values ψ_P because their calculation is different. Instead, the best solution of each approach is plugged into the objective function of the other approach (λ_P and ψ_A).

		Population Nodes 10	13	16	19	22	25
λ_A	Ex-Ante Fitness of Solution with Best Ex-Ante Fitness	5.48	10.81	9.09	8.10	6.96	7.93
λ_P	Ex-Ante Fitness of Solution with Best Ex-Post Fitness	5.48	10.19	9.09	7.19	6.73	6.43
Δ_A	Difference Ante in %	0.00	5.73	0.00	11.23	3.28	18.96
ψ_P	Ex-Post Fitness of Solution with Best Ex-Post Fitness	4.72	7.61	6.54	5.49	5.45	5.43
ψ_A	Ex-Post Fitness of Solution with Best Ex-Ante Fitness	4.50	6.45	6.54	5.08	4.21	4.66
Δ_P	Difference Post in %	4.50	15.25	0.00	7.54	22.72	14.20

Table 6-3 CE: Population Nodes: Single Seed: Fitness Values

$$(2) \qquad \Delta_A = \frac{\lambda_A - \lambda_P}{\lambda_A} \times 100 \qquad\qquad (3) \qquad \Delta_P = \frac{\psi_P - \psi_A}{\psi_P} \times 100$$

The fitness values of the table are visualized in Figure 6-2. The left graph displays the fitness values of the ex-ante objective (λ_A and λ_P) and the right graph shows the values of the ex-post objective (ψ_P and ψ_A). Both graphs decrease when the number of population nodes increases, which is easily explained by considering that some of these additional nodes are also affected by the disaster, leading to higher costs (distances) and since fitness = 1/costs, the fitness values decrease.

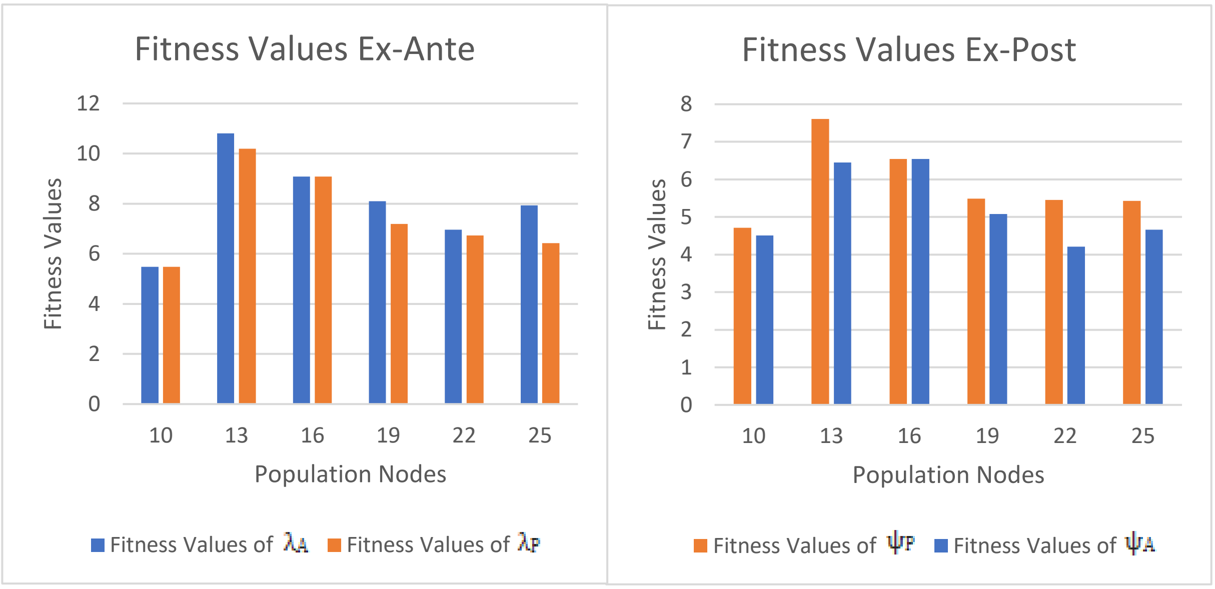


Figure 6-2 CE: Population Nodes: Single Seed: Fitness Values

The percentage differences (Δ_A and Δ_P) are computed by the two mathematical formulas given above, where (2) computes the differences of the solutions that are plugged into the ex-ante objective function and (3) computes the differences of the solutions that are plugged into the ex-post objective function.

For the data in Table 6-3 this means that to determine the ex-ante percentage difference Δ_A of the second column (computed by equation (2)), the best ex-post solution is plugged into the ex-ante objective function, leading to a fitness value of $\lambda_P = 10.19$. This value is subtracted from the ex-ante fitness of the solution with the best ex-ante fitness ($\lambda_A = 10.81$). The result is then divided by λ_A and multiplied by 100, which corresponds to 5.73%. Equation (3) shows that the same is done to calculate the ex-post percentage difference Δ_P but this time the ex-post objective function is used.

The Δ_A results in the table show how much worse the best ex-post solution is when plugged into the ex-ante objective function (λ_P) compared to the best ex-ante solution that is also plugged into the ex-ante objective function (λ_A). The differences range from 0.00% to 18.96%. A difference of 0.00% means that the fitness values (λ_A and λ_P) are identical and therefore both solutions are equally good. λ_P is almost 19% worse than λ_A in the last column (25 population nodes).

Similarly for Δ_P, the differences range from 0.00% to 22.72% and are worse than the Δ_A results. This means that for the seed used, plugging the best ex-post solution into the ex-ante objective function (λ_P) often results in smaller percentage differences than plugging the best ex-ante solution into the ex-post objective function (ψ_A). Δ_A and Δ_P are visualized in the first diagram of Figure 6-3.

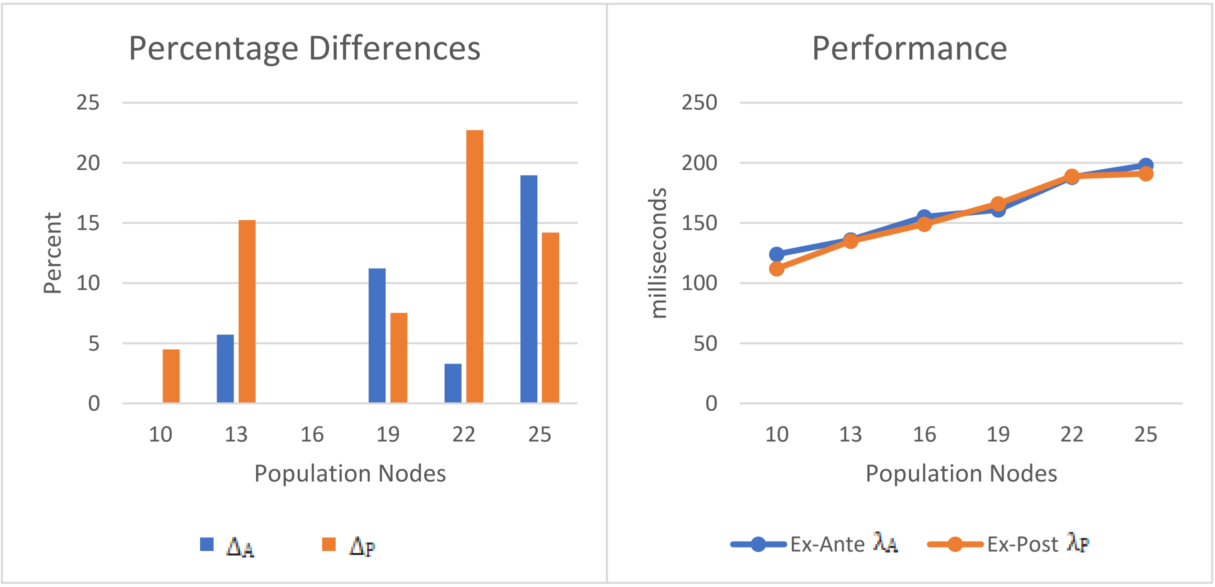


Figure 6-3 CE: Population Nodes: Single Seed: Percentage Differences and Performance

The percentage differences of both evaluation approaches tend to increase as more population nodes are added, although no clear pattern is observed.

Ex-post (ψ_P) usually performs slightly better than ex-ante (λ_A), as indicated by the right graph of Figure 6-3. A linear increase in computation time is observed as more population nodes are added.

6.3.1.1.1.2. Multiple Seeds

The $\bar{\Delta}_A$ and $\bar{\Delta}_P$ values of Table 6-4 are calculated by applying the following mathematical formulas. These formulas are very similar to the formulas of the single seed, but instead of plugging in just one fitness value, e.g. λ_A, the arithmetic mean of 100 randomly generated seeds, e.g. $\bar{\lambda}_A$, is now calculated and then plugged into the formula.

$$(4) \qquad \bar{\Delta}_A = \frac{\bar{\lambda}_A - \bar{\lambda}_P}{\bar{\lambda}_A} \times 100 \qquad\qquad (5) \qquad \bar{\Delta}_P = \frac{\bar{\psi}_P - \bar{\psi}_A}{\bar{\psi}_P} \times 100$$

		Population Nodes					
		10	13	16	19	22	25
$\bar{\lambda}_A$	Ex-Ante Fitness of Solution with Best Ex-Ante Fitness	10.38	8.81	8.13	7.81	6.45	6.79
$\bar{\lambda}_P$	Ex-Ante Fitness of Solution with Best Ex-Post Fitness	10.24	8.66	7.95	7.65	6.32	6.60
$\bar{\Delta}_A$	Difference Ante in %	1.32	1.76	2.22	2.07	1.99	2.80
$\bar{\psi}_P$	Ex-Post Fitness of Solution with Best Ex-Post Fitness	6.69	5.73	5.33	5.10	4.81	4.73
$\bar{\psi}_A$	Ex-Post Fitness of Solution with Best Ex-Ante Fitness	6.61	5.63	5.24	4.98	4.67	4.62
$\bar{\Delta}_P$	Difference Post in %	1.11	1.82	1.80	2.30	2.79	2.22

Table 6-4 CE: Population Nodes: Fitness Values

While Δ_P showed differences of up to 22.72% in Table 6-3, the largest difference in Table 6-4 for $\bar{\Delta}_P$ is only 2.79%, which reveals how much a single seed can deviate from the arithmetic mean of many. This is also shown by comparing Figure 6-2 (single seed) with Figure 6-4: the ex-ante fitness values (λ_A and λ_P) of Figure 6-2 do vary (greatly), but the ex-ante values ($\bar{\lambda}_A$ and $\bar{\lambda}_P$) of Figure 6-4 are almost the same. Another difference to Figure 6-2 is that it is now very clear that that fitness values decrease as the population nodes increase.

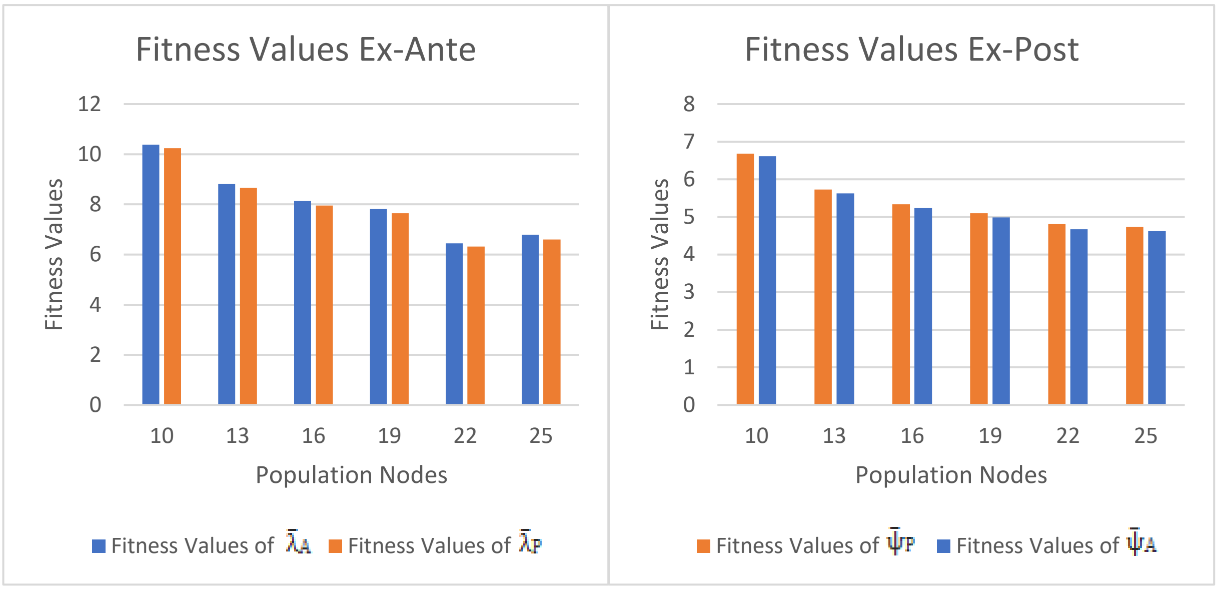


Figure 6-4 CE: Population Nodes: Fitness Values

The first diagram of Figure 6-5 shows the percentage differences $\bar{\Delta}_A$ and $\bar{\Delta}_P$. Compared to Figure 6-3 (single seed), $\bar{\Delta}_P$ is now mostly smaller than $\bar{\Delta}_A$, which means that plugging the best ex-ante solution into the ex-post objective function ($\bar{\psi}_A$) usually leads to results that are closer to the best result than plugging the best ex-post solution into the ex-ante objective function ($\bar{\psi}_P$). Both percentage differences, $\bar{\Delta}_P$ more than $\bar{\Delta}_A$, increase as the population nodes increase. The performance of both evaluation methods is linear and again, ex-post ($\bar{\psi}_P$) performs slightly better than ex-ante ($\bar{\lambda}_A$).

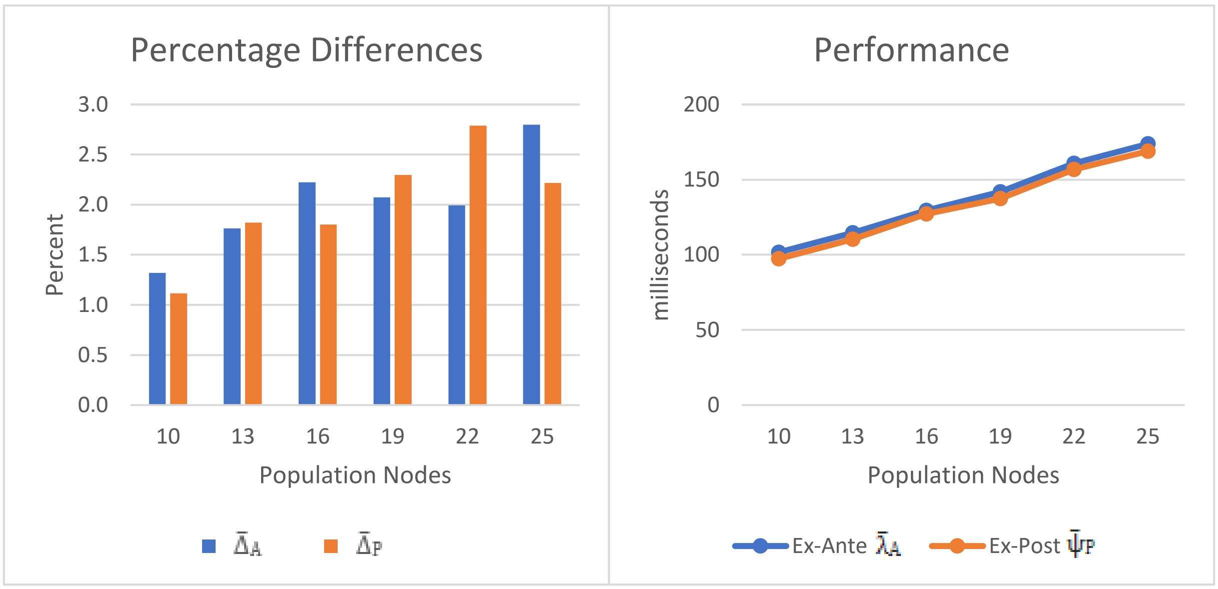


Figure 6-5 CE: Population Nodes: Percentage Differences and Performance

6.3.1.1.2. Comparison of Costs

6.3.1.1.2.1. Single Seed

The costs of the ex-ante, ex-post, and utilitarian solutions are compared and given as percentages in Table 6-5. For the ex-ante and ex-post objective functions, the costs are simply the summed distances of all affected population nodes to their nearest shelter. For the utilitarian objective function, the distances of the affected population nodes to their nearest shelters are first multiplied by the number of households (weight) and then summed. The second column in the table specifies which costs are being compared:

1) Costs of the solution with best ex-ante fitness (θ_A) are compared to the costs of the solution with best ex-post fitness (θ_P),
2) Weighted costs (utilitarian costs) of the solution with best ex-ante fitness (γ_A) are compared to the utilitarian costs of the utilitarian objective function (γ_U), and
3) Weighted costs (utilitarian costs) of the solution with best ex-post fitness (γ_P) are compared to the utilitarian costs of the utilitarian objective function (γ_U).

Since this problem has three scenarios and the affected population nodes of each scenario can be completely different, the scenarios are compared separately. This means that for scenario 3 and 10 population nodes in 1), the result is obtained by calculating the ex-ante costs θ_A and the ex-post costs θ_P (summed distances of all affected population nodes to their nearest shelter) of scenario 3 using formula (6). Similarly for scenario 3 and 10 population nodes in 2), but here the costs of the ex-ante solution of scenario 3 are multiplied by the number of households (utilitarian costs) and then compared to scenario 3 of the utilitarian solution γ_U using formula (7).

Table 6-5 first compares θ_A with θ_P separately for each scenario. Positive numbers (red color) mean that the ex-ante costs θ_A are worse than the ex-post costs θ_P by that percentage. Zero (white color) means that both approaches are equally good and negative numbers (green color) mean that the ex-ante costs θ_A are better than the ex-post costs θ_P by that percentage. Considering the colors, it seems that the ex-ante costs θ_A are mostly worse than the ex-post costs θ_P, sometimes equally good, and only twice better for that specific seed. The numbers are percentage differences $\Delta_{1)}$ and they are calculated with mathematical formula (6). The results range from -20.75% to 66.13%, which means that for scenario 2 and 19 population nodes, θ_A saves 20.75% of the costs compared to θ_P, but for scenario 3 and 10 nodes, the cost of θ_A is almost 2/3 higher than the cost of θ_P.

			Population Nodes					
			10	13	16	19	22	25
$\Delta_{1)}$	1) Ex-Ante (θ_A)	Scenario 1	0.00	0.00	0.00	6.09	43.39	12.58
	vs.	Scenario 2	20.00	37.75	0.00	-20.75	15.44	7.66
	Ex-Post (θ_P)	Scenario 3	66.13	-18.07	0.00	4.27	6.72	0.11
$\Delta_{2)}$	2) Ex-Ante (γ_A)	Scenario 1	0.00	0.00	0.00	10.84	26.47	12.59
	vs.	Scenario 2	18.30	36.38	1.17	0.00	17.24	9.81
	Utilitarian (γ_U)	Scenario 3	48.59	5.44	29.06	31.27	11.74	10.25
$\Delta_{3)}$	3) Ex-Post (γ_P)	Scenario 1	0.00	0.00	0.00	0.00	-6.61	0.00
	vs.	Scenario 2	0.00	0.00	1.17	32.08	0.00	1.23
	Utilitarian (γ_U)	Scenario 3	0.00	46.15	29.06	29.16	11.31	8.02

Table 6-5 CE: Population Nodes: Single Seed: Comparison of Costs (Ex-Ante, Ex-Post, and Utilitarian) in %

$$(6) \quad \Delta_{1)} = \frac{\theta_A - \theta_P}{\theta_P} \times 100 \qquad (7) \quad \Delta_{2)} = \frac{\gamma_A - \gamma_U}{\gamma_U} \times 100 \qquad (8) \quad \Delta_{3)} = \frac{\gamma_P - \gamma_U}{\gamma_U} \times 100$$

Next, γ_A and γ_U are compared. Unlike before, the costs are now weighted, that is, the number of households is taken into account. Again, the positive/negative numbers indicate by what percentage the ex-ante utilitarian costs γ_A are worse/better than the utilitarian costs γ_U. Since the utilitarian objective function chooses the solution with the minimum utilitarian costs, contrary to the ex-ante and ex-post objective functions that focus on maximizing fairness, it is no surprise that the ex-ante solution is mostly worse or, at best, equally good as the utilitarian solution. Here, from the perspective of the ex-ante solution, the worst percentage difference $\Delta_{2)}$, calculated with mathematical formula (7), is 48.59%, and the best is 0%.

Lastly, γ_P and γ_U are compared. The percentage differences $\Delta_{3)}$ are computed with formula (8). These differences are overall better than the differences of 2) and there is even a reduction of 6.61% for scenario 1 and 22 population nodes. However, due to scenario 3, the total costs (costs of all scenarios added together) of this ex-post solution are still higher than the total costs of the utilitarian solution.

Figure 6-6 shows the costs of the ex-ante and ex-post solutions. Just as expected, considering that costs increase with each additional affected population node, both diagrams show an upward movement when the number of (affected) population nodes increases. At 25 population nodes in the ex-ante costs (θ_A) graph, the costs of scenario 1 decrease compared to 22 population nodes. The reason for this decrease is that as more population nodes are affected, the solution (combination of permanent and temporary locations) may change, which can reduce the costs of a scenario.

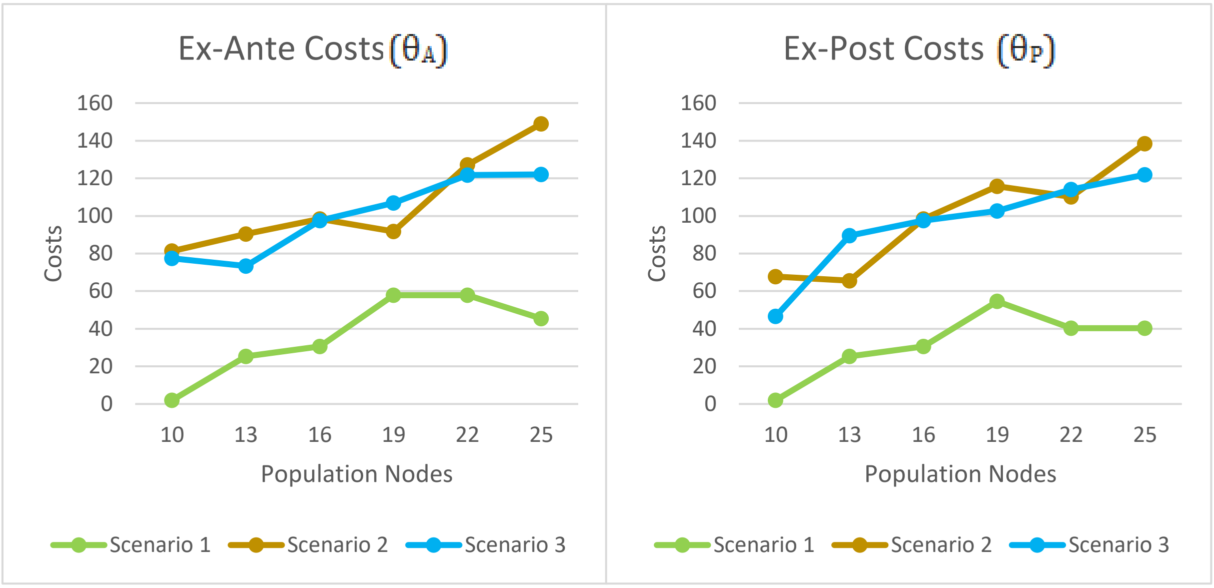


Figure 6-6 CE: Population Nodes: Single Seed: Ex-Ante and Ex-Post Costs

Therefore, it is even possible that the total costs (costs of all scenarios added together) decrease when more population nodes are added. This can be observed in the ex-post costs (θ_P) graph at 22 population nodes compared to 19. The total costs at 19 nodes are 272.85 units, while the total costs at 22 nodes are 264.59 units.

6.3.1.1.2.2. Multiple Seeds

The values given in Table 6-6 are percentages calculated with the mathematical formulas below. These formulas are very similar to those for the single seed with the difference that the arithmetic means of 100 seeds are calculated first and the results are then plugged into the equations.

$$(9)\quad \bar{\Delta}_{1)} = \frac{\bar{\gamma}-\bar{\rho}}{\bar{\rho}} \qquad (10)\quad \bar{\Delta}_{2)} = \frac{\bar{\gamma}-\bar{\tau}_2}{\bar{\tau}} \qquad (11)\quad \bar{\Delta}_{3)} = \frac{\bar{\rho}-\bar{\tau}}{\bar{\tau}}$$

			Population Nodes					
			10	13	16	19	22	25
$\bar{\Delta}_{1)}$	1) Ex-Ante ($\bar{\theta}_A$) vs. Ex-Post ($\bar{\theta}_P$)	Scenario 1	0.00	-0.93	1.01	-1.52	0.99	0.43
		Scenario 2	-0.22	0.75	0.83	-0.32	1.33	1.59
		Scenario 3	-0.46	-0.11	0.80	1.60	2.27	1.84
$\bar{\Delta}_{2)}$	2) Ex-Ante ($\bar{\gamma}_A$) vs. Utilitarian ($\bar{\gamma}_U$)	Scenario 1	0.00	8.88	9.40	11.47	11.72	15.18
		Scenario 2	11.82	13.23	16.56	15.03	15.52	18.71
		Scenario 3	13.89	14.56	13.06	14.90	13.59	15.96
$\bar{\Delta}_{3)}$	3) Ex-Post ($\bar{\gamma}_P$) vs. Utilitarian ($\bar{\gamma}_U$)	Scenario 1	0.00	9.45	6.14	11.43	10.61	10.47
		Scenario 2	10.28	11.78	12.86	11.77	11.88	12.03
		Scenario 3	9.82	12.18	12.82	12.13	9.47	8.28

Table 6-6 CE: Population Nodes: Comparison of Costs (Ex-Ante, Ex-Post, and Utilitarian) in %

For $\bar{\Delta}_{1j}$, the percentage differences, ranging from -1.52 to 2.27, are much smaller than the differences of the single seed in Table 6-5, that range from -20.75 to 66.13, and even improved for the ex-ante approach, as there are now more negative percentages (highlighted in green) than before. To repeat, negative percentages mean that the ex-ante costs ($\bar{\theta}_A$) are lower than the ex-post costs ($\bar{\theta}_P$) by this percentage. However, the ex-post costs ($\bar{\theta}_P$) are still mostly lower than the ex-ante costs ($\bar{\theta}_A$).

Comparing 2) and 3) of Table 6-6 with Table 6-5, almost every entry is now highlighted in red, showing that the ex-ante costs ($\bar{\gamma}_A$) and ex-post costs ($\bar{\gamma}_P$) are worse than the utilitarian costs ($\bar{\gamma}_U$) by this percentage. These percentages are often significantly lower than the percentages of the single seed. While the largest percentage difference of the single seed is 48.59%, the largest difference of the arithmetic mean is only 18.71%.

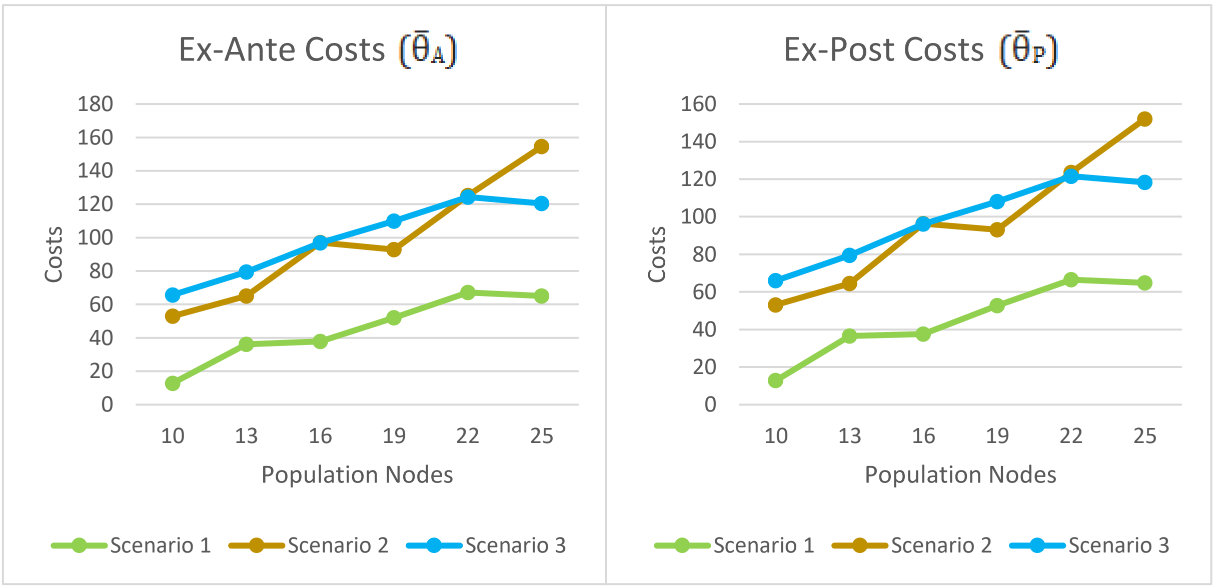


Figure 6-7 CE: Population Nodes: Ex-Ante and Ex-Post Costs

As already indicated by 1) in Table 6-6, and shown in Figure 6-7, the graphs of both evaluation methods are almost identical with slight deviations. Again, the costs increase when more affected population nodes are added.

6.3.1.2. Increasing Potential Locations

6.3.1.2.1. Fitness Values and Performance

Table 6-7 in combination with Figure 6-8 shows that the fitness values for both approaches increase as more potential locations are added.

		Potential Locations					
		8	11	14	17	20	23
$\bar{\lambda}_A$	Ex-Ante Fitness of Solution with Best Ex-Ante Fitness	9.08	10.38	11.64	12.76	13.65	14.93
$\bar{\lambda}_P$	Ex-Ante Fitness of Solution with Best Ex-Post Fitness	8.97	10.24	11.35	12.43	13.17	14.46
$\bar{\Delta}_A$	Difference Ante in %	1.20	1.32	2.53	2.60	3.50	3.11
$\bar{\psi}_P$	Ex-Post Fitness of Solution with Best Ex-Post Fitness	5.91	6.69	7.48	8.18	8.82	9.69
$\bar{\psi}_A$	Ex-Post Fitness of Solution with Best Ex-Ante Fitness	5.89	6.61	7.33	7.85	8.39	9.16
$\bar{\Delta}_P$	Difference Post in %	0.33	1.11	1.99	4.00	4.88	5.45

Table 6-7 CE: Potential Locations: Fitness Values

The explanation for this increase is that as more potential locations become available, the algorithm can select better locations to maximize fairness for all affected population nodes.

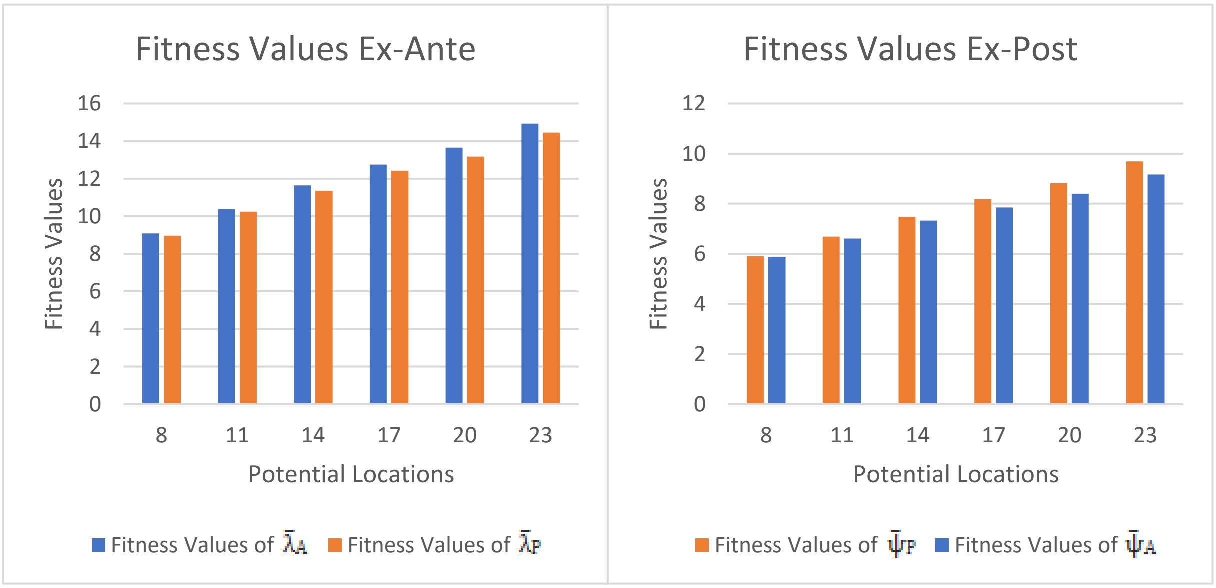


Figure 6-8 CE: Potential Locations: Fitness Values

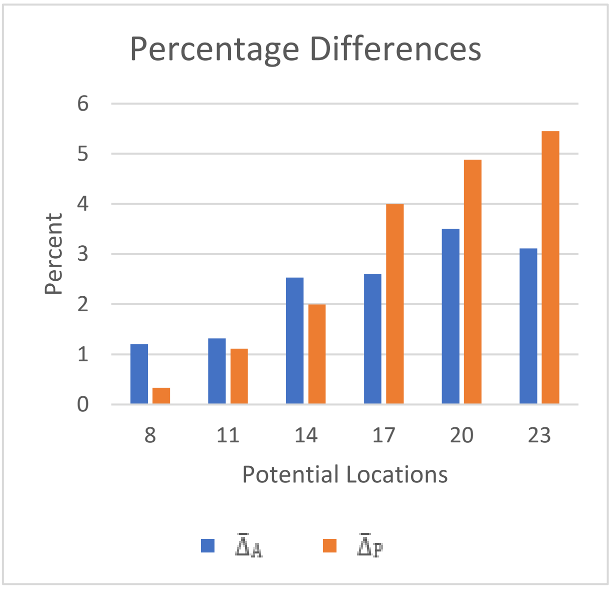


Figure 6-9 CE: Potential Locations: Percentage Differences

Figure 6-9 shows a strong positive correlation between increasing potential locations and the percentage differences of ex-post $\bar{\Delta}_P$, meaning that when plugging the best ex-ante solution into the ex-post objective function (ψ_A), the resulting fitness value becomes on average worse compared to the best ex-post solution (ψ_P) as more potential locations become available. The percentage differences of ex-ante $\bar{\Delta}_A$ also increase (with one exception), but to a lesser extent.

An additional table (Table 6-8) has been included to explain the behavior of Figure 6-10. In Table 6-8, all possible combinations are calculated using the binomial coefficient (formula (1)). For the permanent combinations in the first column this means that n = 8 and k = 2, since 8 potential locations are available and 2 permanent shelters are built (see 6.1 "General Settings and Overview"), resulting in 28 permanent combinations. For the temporary combinations, the same formula applies, except that now n = 6 (8 potential locations – 2 permanent shelters already selected) and k = 2 (2 temporary shelters are built), which results in 15 temporary combinations. As the temporary combinations are calculated separately for each permanent combination, they must be multiplied by 28 (permanent combinations), resulting in 420 possible combinations.

	Potential Locations					
	8	11	14	17	20	23
Permanent Combinations	28	55	91	136	190	253
Temporary Combinations	15	36	66	105	153	210
Possible Combinations	420	1980	6006	14280	29070	53130

Table 6-8 CE: Potential Locations: Permanent and Temporary Combinations

The right graph of Figure 6-10 illustrates the results of Table 6-8. Comparing the two graphs, it is clear that increasing the number of potential locations heavily affects the computation time. Here, the performance of ex-post is marginally better than the performance of ex-ante.

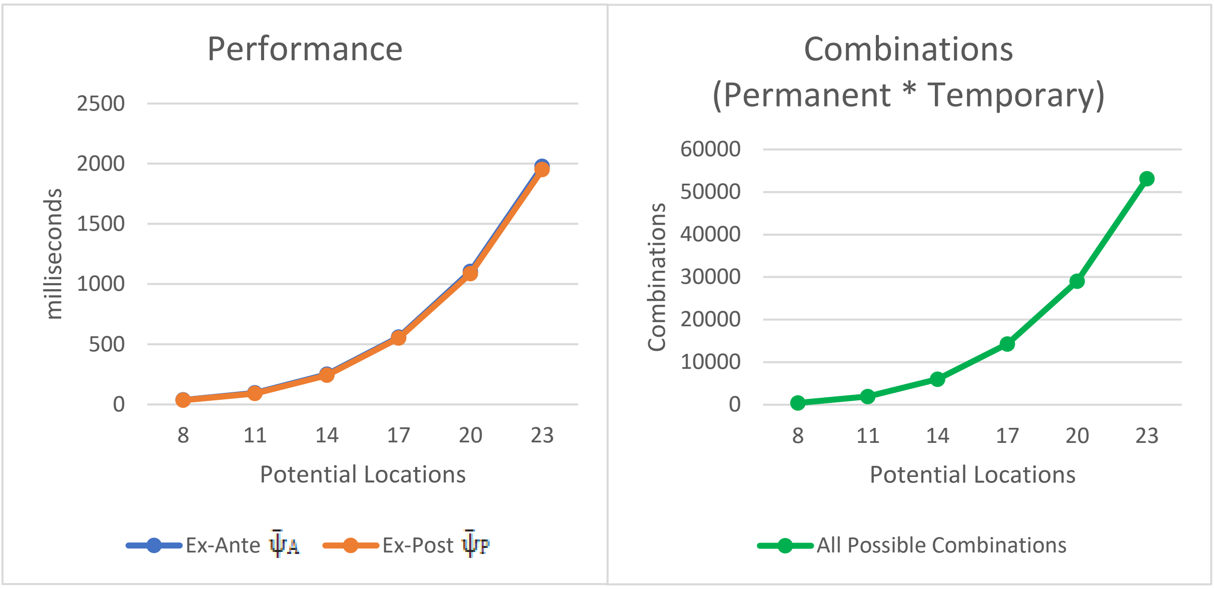


Figure 6-10 CE: Potential Locations: Performance and Combinations

6.3.1.2.2. Comparison of Costs

Considering the $\bar{\Delta}_{1)}$ values in Table 6-9, it appears that ex-ante has slightly lower costs than ex-post for up to 14 potential locations, but the costs grow rapidly as the number of potential locations increases.

$\bar{\Delta}_{2)}$ shows an even stronger correlation between utilitarian costs and potential locations. The costs of the ex-ante solution at scenario 3 is at first only 5.18% (column 1) worse than $\bar{\gamma}_U$, but this difference gets worse every time new locations are added, up to 34.84% (column 6).

$\bar{\Delta}_{3)}$ also shows a link between costs and locations. The difference of column 1 is at first higher than the difference of $\bar{\Delta}_{2)}$, however, it does not grow as fast and does not even reach half of the costs of $\bar{\Delta}_{2)}$ in column 6.

			Potential Locations					
			8	11	14	17	20	23
$\bar{\Delta}_{1)}$	1) Ex-Ante ($\bar{\theta}_A$)	Scenario 1	0.00	0.00	0.00	0.00	0.00	0.00
	vs.	Scenario 2	0.78	-0.22	1.22	2.73	5.78	10.33
	Ex-Post ($\bar{\theta}_P$)	Scenario 3	-0.72	-0.46	-0.37	4.41	4.87	5.84
$\bar{\Delta}_{2)}$	2) Ex-Ante ($\bar{\gamma}_A$)	Scenario 1	0.00	0.00	0.00	0.00	0.00	0.00
	vs.	Scenario 2	6.27	11.82	16.39	24.14	27.80	31.14
	Utilitarian ($\bar{\gamma}_U$)	Scenario 3	5.18	13.89	19.11	22.77	30.35	34.84
$\bar{\Delta}_{3)}$	3) Ex-Post ($\bar{\gamma}_P$)	Scenario 1	0.00	0.00	0.00	0.00	0.00	0.00
	vs.	Scenario 2	6.98	10.28	11.44	13.63	12.63	12.89
	Utilitarian ($\bar{\gamma}_U$)	Scenario 3	6.41	9.82	12.30	14.73	16.60	16.02

Table 6-9 CE: Potential Locations: Comparison of Costs (Ex-Ante, Ex-Post, and Utilitarian) in %

In Figure 6-11 the costs of both graphs decrease. When more locations are added, it is possible that these new locations are placed closer to affected population nodes than already available locations, improving the fitness value and thus leading to lower costs (fitness = 1/costs).

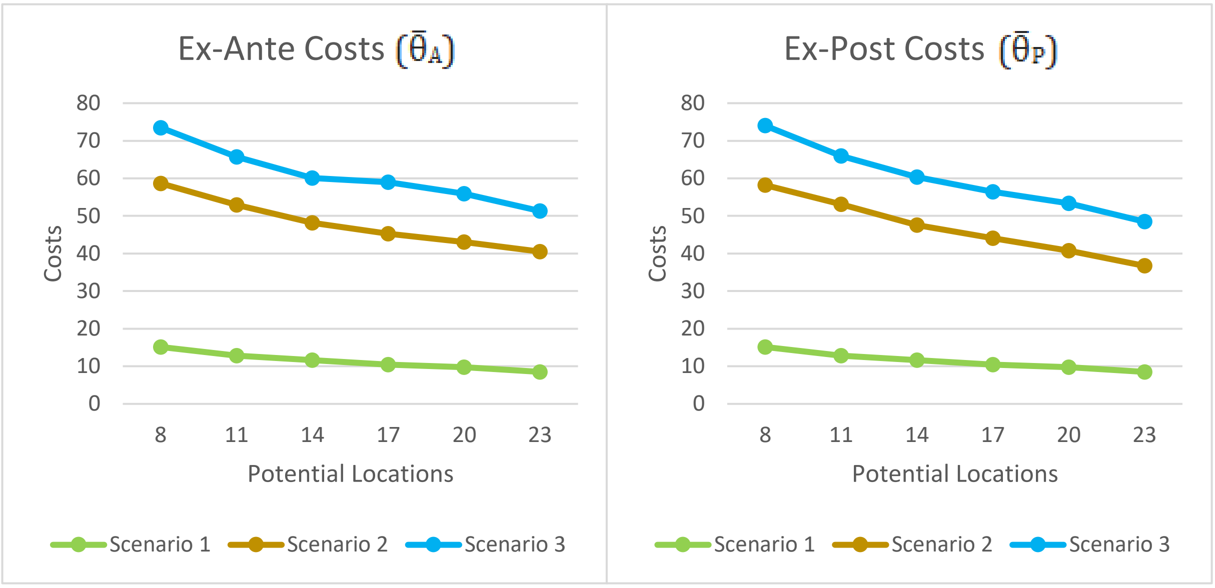


Figure 6-11 CE: Potential Locations: Ex-Ante and Ex-Post Costs

6.3.1.3. Increasing Permanent Shelters

6.3.1.3.1. Fitness Values and Performance

As illustrated in Figure 6-12, which uses the data from Table 6-10, fitness values increase when more permanent shelters are built. However, strongly depending on the problem size, adding more shelters may not improve the solution. In this case, the fitness values of both evaluation approaches only grow for up to two permanent shelters. Three or more shelters do not further improve these values. The reason for this outcome is the position of the potential locations; for the first two permanent shelters the best locations are chosen, but all other locations are positioned in a way that they cannot contribute to a better solution.

		Permanent Shelters					
		1	2	3	4	5	6
$\bar{\lambda}_A$	Ex-Ante Fitness of Solution with Best Ex-Ante Fitness	10.22	10.38	10.38	10.38	10.38	10.38
$\bar{\lambda}_P$	Ex-Ante Fitness of Solution with Best Ex-Post Fitness	9.94	10.24	10.32	10.26	10.31	10.38
$\bar{\Delta}_A$	Difference Ante in %	2.72	1.32	0.49	1.07	0.66	0.00
$\bar{\psi}_P$	Ex-Post Fitness of Solution with Best Ex-Post Fitness	6.57	6.69	6.69	6.69	6.69	6.69
$\bar{\psi}_A$	Ex-Post Fitness of Solution with Best Ex-Ante Fitness	6.47	6.61	6.66	6.68	6.68	6.69
$\bar{\Delta}_P$	Difference Post in %	1.59	1.11	0.40	0.02	0.08	0.00

Table 6-10 CE: Permanent Shelters: Fitness Values

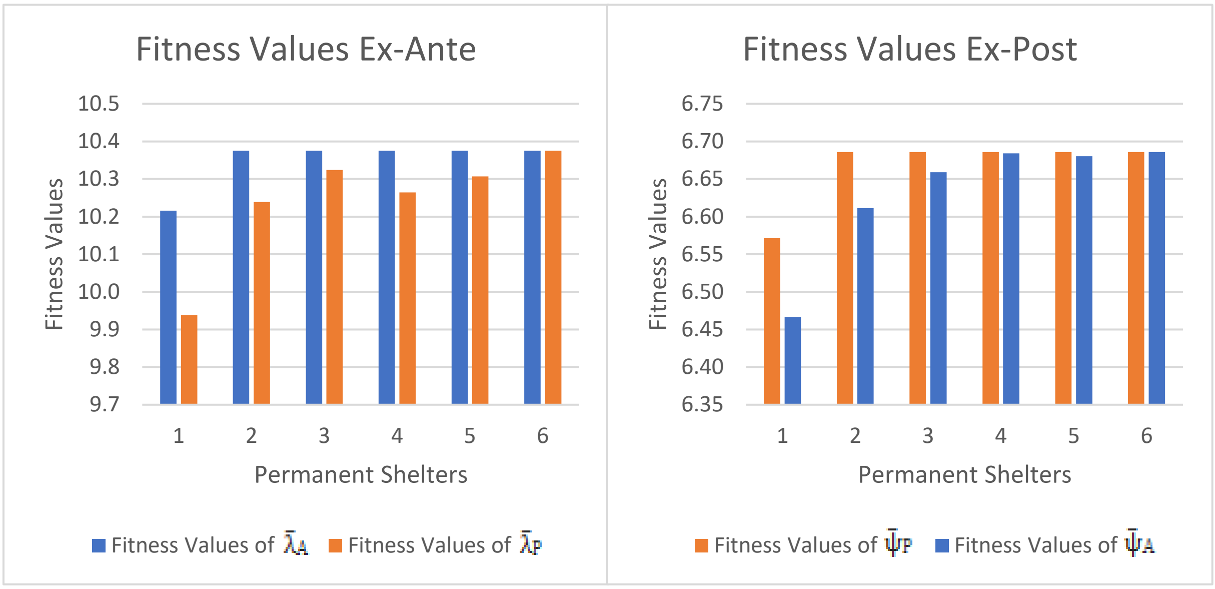


Figure 6-12 CE: Permanent Shelters: Fitness Values

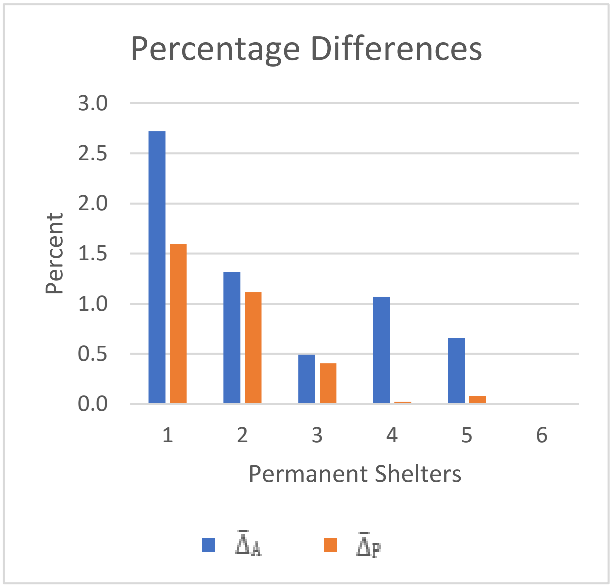


Figure 6-13 CE: Permanent Shelters: Percentage Differences

The percentage differences in Figure 6-13 are quite high at the beginning, but decrease as more permanent shelters are built, which is plausible because an increasing number of shelters and a constant number of potential locations lead to similar or even identical fitness values.

In the graph, the ex-post percentage differences $\bar{\Delta}_P$ are smaller than $\bar{\Delta}_A$, suggesting that the best ex-ante solution plugged into the ex-post objective function $\bar{\psi}_A$ delivers results that are (much) closer to the best ex-post solution $\bar{\psi}_P$ than the obtained results of the best ex-post solution plugged into the ex-ante objective function $\bar{\lambda}_P$ are to the best ex-ante solution $\bar{\lambda}_A$.

	Permanent Shelters					
	1	2	3	4	5	6
Permanent Combinations	8	28	56	70	56	28
Temporary Combinations	21	15	10	6	3	1
Possible Combinations	168	420	560	420	168	28

Table 6-11 CE: Permanent Shelters: Permanent and Temporary Combinations

Similar to the previous subsection (6.3.1.2 “Increasing Potential Locations”), the number of all possible combinations of each run with a different number of permanent shelters is given in Table 6-11 and visualized in the right diagram of Figure 6-14. Again, the computation time (left diagram) is heavily affected by the possible combinations since all of them are calculated. Ex-post performs slightly better than ex-ante.

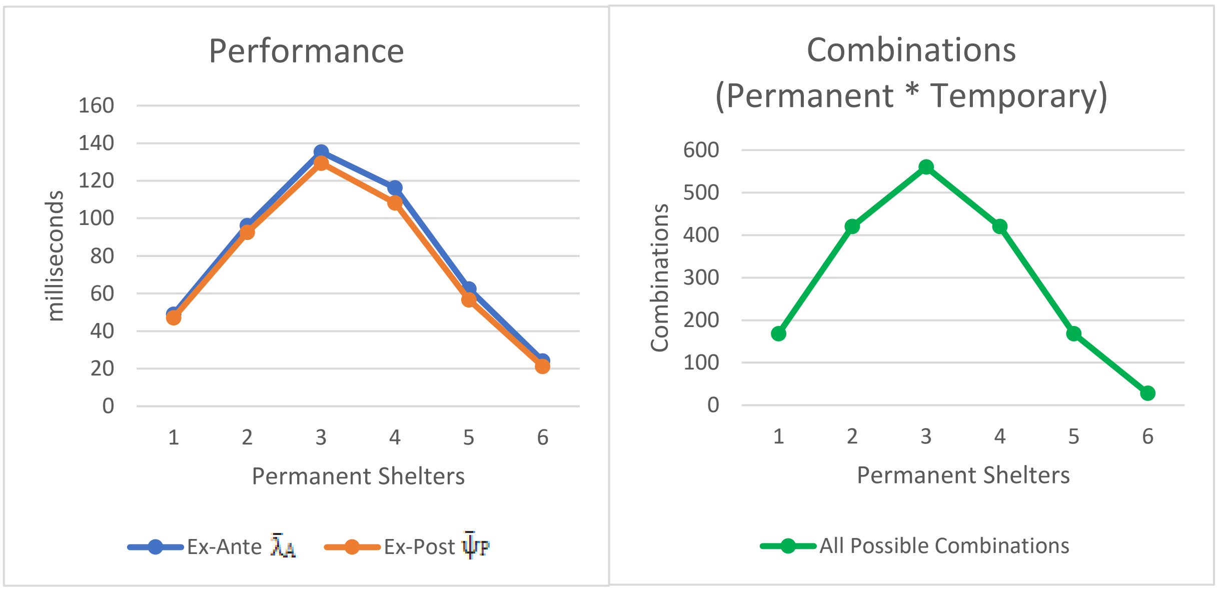


Figure 6-14 CE: Permanent Shelters: Performance and Combinations

6.3.1.3.2. Comparison of Costs

The ex-ante costs $\bar{\theta}_A$ of 1) in Table 6-12 only perform worse than the ex-post costs $\bar{\theta}_P$ when one permanent shelter is built. For all other numbers of shelters, the percentage difference $\bar{\Delta}_{1)}$ is mostly positive.

			Permanent Shelters					
			1	2	3	4	5	6
$\bar{\Delta}_{1)}$	1) Ex-Ante ($\bar{\theta}_A$)	Scenario 1	0.00	0.00	0.00	0.00	0.00	0.00
	vs.	Scenario 2	1.43	-0.22	-0.82	-1.03	0.03	0.00
	Ex-Post ($\bar{\theta}_P$)	Scenario 3	1.46	-0.46	0.19	0.25	-0.16	0.00
$\bar{\Delta}_{2)}$	2) Ex-Ante ($\bar{\gamma}_A$)	Scenario 1	0.00	0.00	0.00	0.00	0.00	0.00
	vs.	Scenario 2	17.29	11.82	7.63	5.08	1.94	0.00
	Utilitarian ($\bar{\gamma}_U$)	Scenario 3	14.13	13.89	6.82	4.48	1.98	0.00
$\bar{\Delta}_{3)}$	3) Ex-Post ($\bar{\gamma}_P$)	Scenario 1	0.00	0.00	0.00	0.00	0.00	0.00
	vs.	Scenario 2	15.02	10.28	7.87	3.78	2.95	0.00
	Utilitarian ($\bar{\gamma}_U$)	Scenario 3	10.85	9.82	7.38	4.49	1.98	0.00

Table 6-12 CE: Permanent Shelters: Comparison of Costs (Ex-Ante, Ex-Post, and Utilitarian) in %

For $\bar{\Delta}_{2)}$ and $\bar{\Delta}_{3)}$ a decrease in the percentage differences is observed when more permanent shelters are built. Six permanent shelters always result in a difference of 0% in this example because eight potential locations are available and two temporary shelters are built. Therefore, all locations are used and the costs are the same.

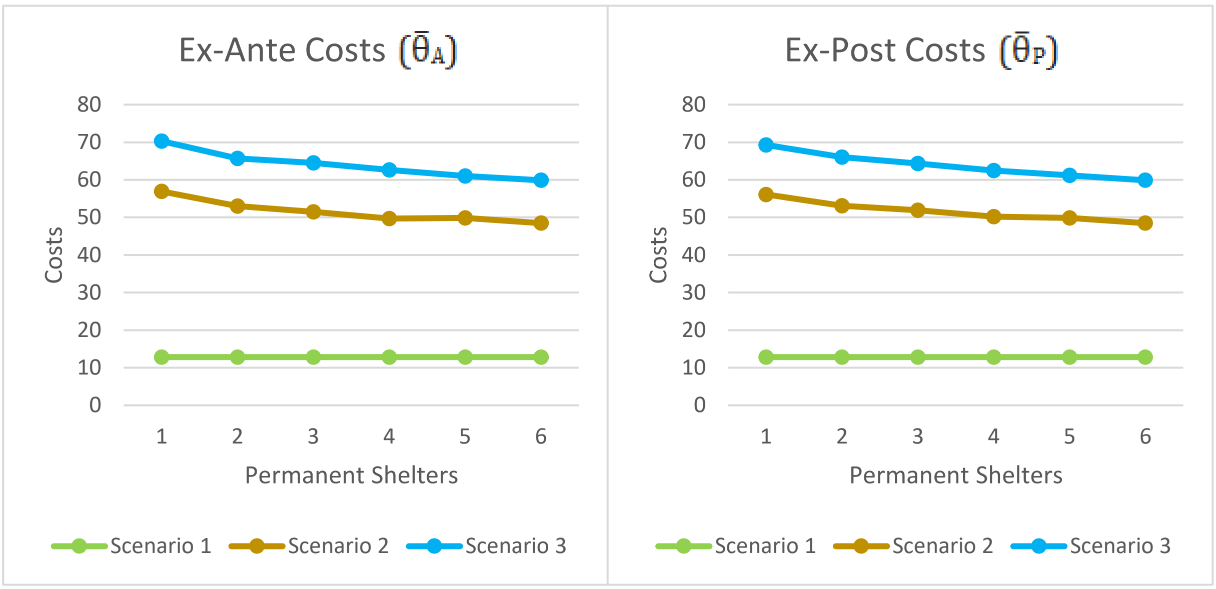


Figure 6-15 CE: Permanent Shelters: Ex-Ante and Ex-Post Costs

Figure 6-15 shows the costs of all three scenarios of both evaluation approaches. There are only minor differences between these two graphs. The costs decrease when the number of shelters increases, which is obvious because additional shelters can be located closer to some affected population nodes and therefore lead to shorter distances (=costs) for these nodes. The costs of scenario 1 do not decrease, which means that all added shelters are farther away from these affected population nodes than the already existing shelters.

6.3.1.4. Increasing Temporary Locations

6.3.1.4.1. Fitness Values and Performance

The results of this subsection are very similar to the results of subsection 6.3.1.3 "Increasing Permanent Shelters", as the total number of shelters (permanent and temporary) does not change and the problem size is rather small.

		Temporary Shelters					
		1	2	3	4	5	6
$\bar{\lambda}_A$	Ex-Ante Fitness of Solution with Best Ex-Ante Fitness	10.21	10.38	10.38	10.38	10.38	10.38
$\bar{\lambda}_P$	Ex-Ante Fitness of Solution with Best Ex-Post Fitness	9.97	10.24	10.17	10.32	10.33	10.38
$\bar{\Delta}_A$	Difference Ante in %	2.38	1.32	1.97	0.52	0.40	0.00
$\bar{\psi}_P$	Ex-Post Fitness of Solution with Best Ex-Post Fitness	6.55	6.69	6.69	6.69	6.69	6.69
$\bar{\psi}_A$	Ex-Post Fitness of Solution with Best Ex-Ante Fitness	6.31	6.61	6.68	6.69	6.69	6.69
$\bar{\Delta}_P$	Difference Post in %	3.64	1.11	0.08	0.00	0.00	0.00

Table 6-13 CE: Temporary Shelters: Fitness Values

Figure 6-16 visualizes the fitness values given in Table 6-13. These values increase when the number of temporary shelters increases. For ex-ante and ex-post, there is no further improvement after two temporary shelters are built.

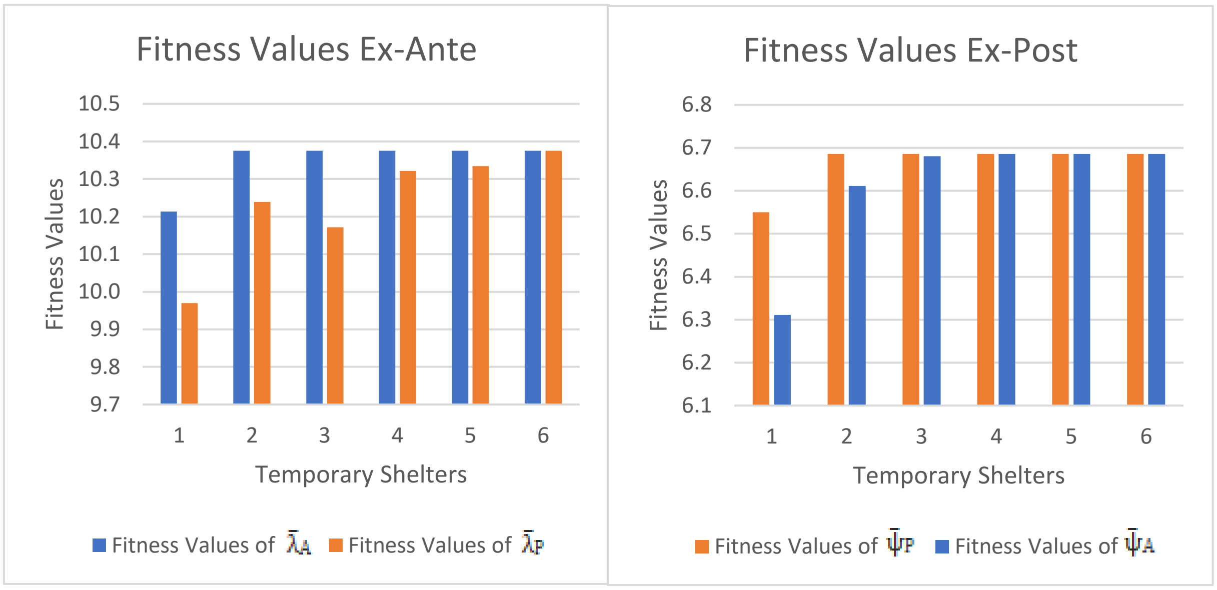


Figure 6-16 CE: Temporary Shelters: Fitness Values

The left diagram of Figure 6-17 shows that the percentage differences for both approaches decrease when more shelters are built and the right diagram shows the performance for each number of shelters, which again strongly depends on the number of possible combinations, as discussed in the previous subsection.

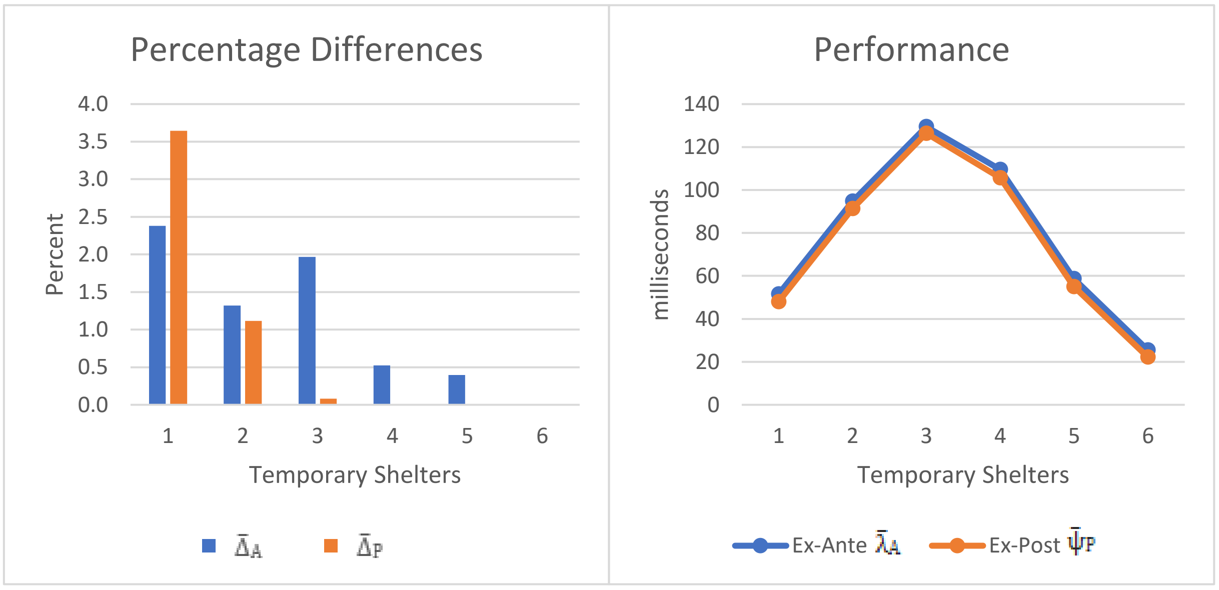


Figure 6-17 CE: Temporary Shelters: Percentage Differences and Performance

6.3.1.4.2. Comparison of Costs

The percentage differences $\bar{\Delta}_{1)}$ in Table 6-14 indicate that the best ex-ante solutions have higher costs than the best ex-post solutions when only one temporary shelter is built, but when the number of shelters increases, ex-ante tends to have lower costs than ex-post.

When the weighted ex-ante and ex-post costs are compared with the utilitarian costs ($\bar{\Delta}_{2)}$ and $\bar{\Delta}_{3)}$), it appears that ex-ante causes much higher costs than ex-post for column 1, scenario 2 (29.38%) and 3 (23.26%). However, as the number of shelters grows, the percentage differences decrease for both approaches.

			Temporary Shelters					
			1	2	3	4	5	6
$\bar{\Delta}_{1)}$	1) Ex-Ante ($\bar{\theta}_A$) vs. Ex-Post ($\bar{\theta}_P$)	Scenario 1	0.00	0.00	0.00	0.00	0.00	0.00
		Scenario 2	6.12	-0.22	-2.19	-0.85	-0.91	0.00
		Scenario 3	5.49	-0.46	-1.29	0.04	-0.28	0.00
$\bar{\Delta}_{2)}$	2) Ex-Ante ($\bar{\gamma}_A$) vs. Utilitarian ($\bar{\gamma}_U$)	Scenario 1	0.00	0.00	0.00	0.00	0.00	0.00
		Scenario 2	29.38	11.82	8.34	5.24	2.69	0.00
		Scenario 3	23.26	13.89	7.71	6.08	2.56	0.00
$\bar{\Delta}_{3)}$	3) Ex-Post ($\bar{\gamma}_P$) vs. Utilitarian ($\bar{\gamma}_U$)	Scenario 1	0.00	0.00	0.00	0.00	0.00	0.00
		Scenario 2	10.06	10.28	10.01	4.66	3.57	0.00
		Scenario 3	8.67	9.82	7.71	4.44	2.10	0.00

Table 6-14 CE: Temporary Shelters: Comparison of Costs (Ex-Ante, Ex-Post, and Utilitarian) in %

For scenarios 2 and 3 in Figure 6-18, the costs of the best ex-ante solution are a bit higher than the costs of the best ex-post solution when only one temporary shelter is built. Similar to the cost graphs of Figure 6-15 (permanent shelters), the costs decrease when the number of shelters increases.

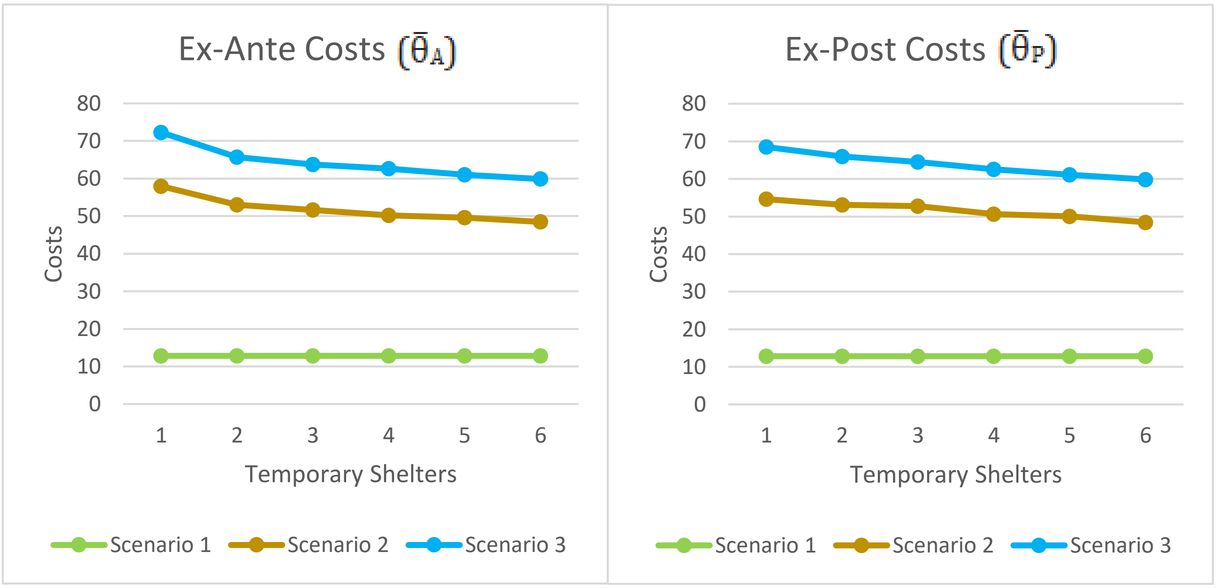


Figure 6-18 CE: Temporary Locations: Ex-Ante and Ex-Post Costs

6.3.2. Genetic Algorithm

The purpose of this subsection is to compare the genetic algorithm (GA) with the complete enumeration (CE). Ideally, the fitness values of the GA are identical to the values of the CE.

In subsection 6.3.2.3 “Increasing Permanent Shelters”, especially in Figure 6-26 (performance graph), it will become clear that the GA finds a solution much faster than the CE. However, this only applies to large problem sizes as the GA executes much more code than the CE, leading to a worse performance for small problem sizes. Furthermore, the GA may not find the best solution every time, as it can get stuck in a local optimum. In contrast to the performance graph of the permanent shelters, the performance graph of the temporary shelters (Figure 6-30) does not change because a complete enumeration is still applied to calculate all possible combinations of temporary shelters.

The input data of the GA is given in 6.3 “Small Problem Size”. The two-point crossover is used in combination with the roulette wheel selection because these settings deliver the best fitness values, which will be shown later in section 6.4 “Large Problem Size”.

6.3.2.1. Increasing Population Nodes

6.3.2.1.1. Fitness Values and Performance

Table 6-15 shows the fitness values of ex-ante and ex-post. The underlying calculations (explained in 6.3.1.1.1.2 “Multiple Seeds”) are the same as for the CE .These fitness values ($\bar{\lambda}_A$ and $\bar{\psi}_P$) are identical to the fitness values of the CE, which means that the GA found the best solutions. $\bar{\lambda}_P$ and $\bar{\psi}_A$ vary slightly when compared to the CE values. This occurs when multiple solutions have the same fitness value for one objective function but lead to different fitness values when the solutions are plugged into the other objective function. When multiple

		Population Nodes					
		10	13	16	19	22	25
$\bar{\lambda}_A$	Ex-Ante Fitness of Solution with Best Ex-Ante Fitness	10.38	8.81	8.13	7.81	6.45	6.79
$\bar{\lambda}_P$	Ex-Ante Fitness of Solution with Best Ex-Post Fitness	10.20	8.75	7.91	7.58	6.31	6.60
$\bar{\Delta}_A$	Difference Ante in %	1.65	0.68	2.71	2.95	2.07	2.71
$\bar{\psi}_P$	Ex-Post Fitness of Solution with Best Ex-Post Fitness	6.69	5.73	5.33	5.10	4.81	4.73
$\bar{\psi}_A$	Ex-Post Fitness of Solution with Best Ex-Ante Fitness	6.64	5.65	5.21	4.99	4.66	4.61
$\bar{\Delta}_P$	Difference Post in %	0.71	1.41	2.33	2.22	3.12	2.47

Table 6-15 GA: Population Nodes: Fitness Values

solutions have the same best fitness value, one of them is randomly chosen, and therefore it is quite likely that this results in slightly different values ($\bar{\lambda}_P$ and $\bar{\psi}_A$) for the CE and the GA.

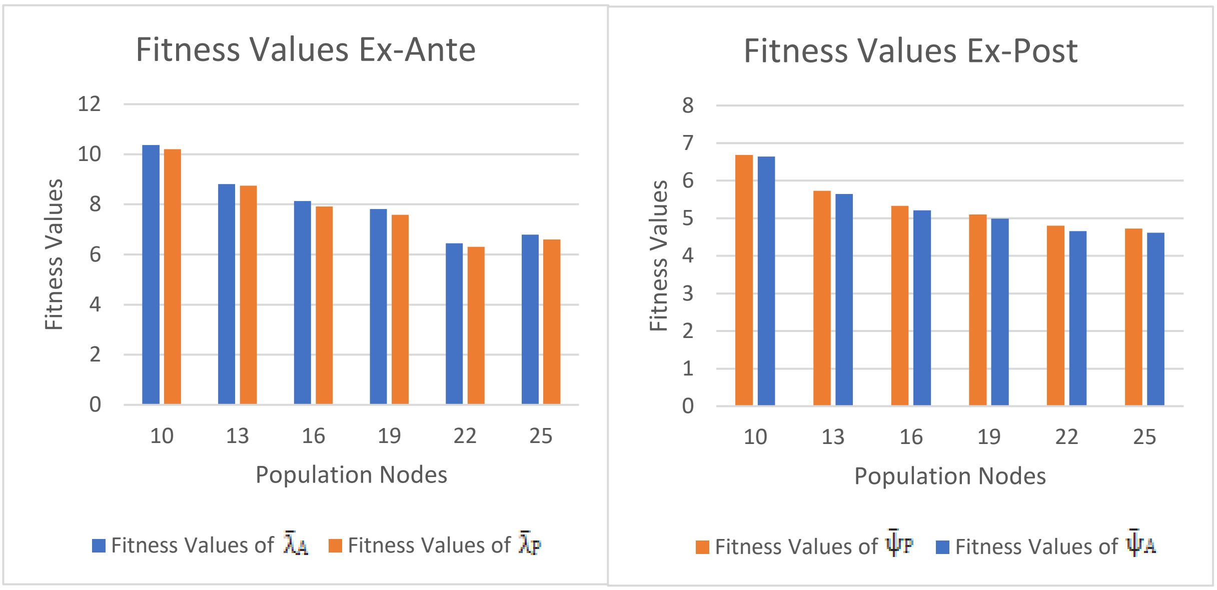


Figure 6-19 GA: Population Nodes: Fitness Values

Figure 6-19 displays the fitness values of the previous table. For both approaches, the fitness values decrease as more population nodes are added due to increasing costs/distances that are caused by the additional nodes that are affected by the disaster.

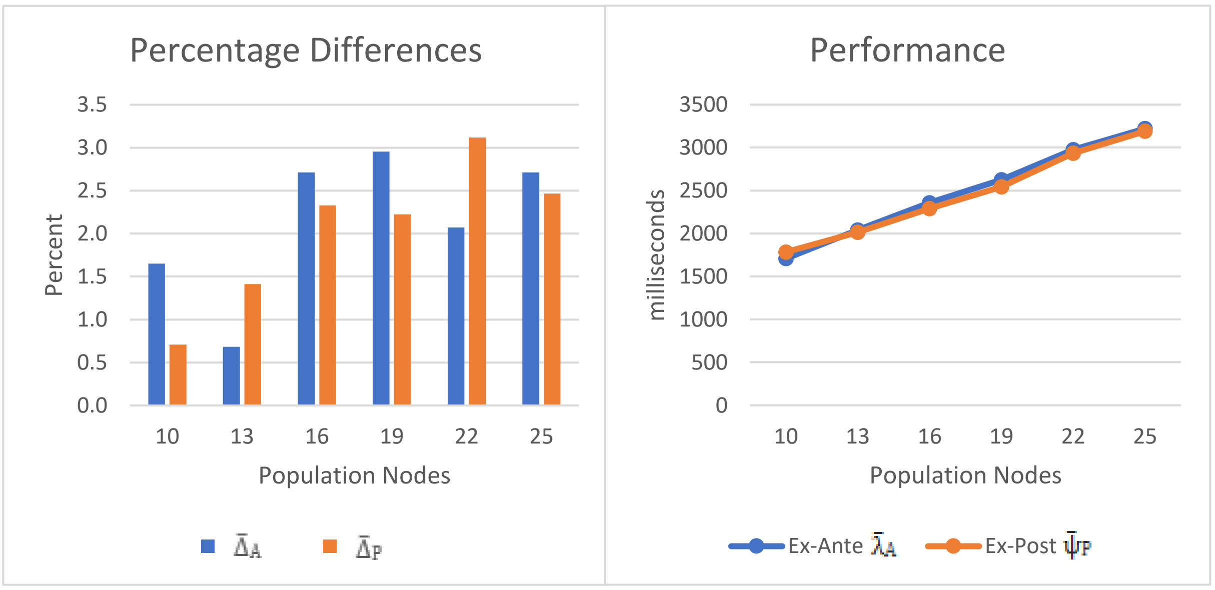


Figure 6-20 GA: Population Nodes: Percentage Differences and Performance

The left diagram of Figure 6-20 illustrates the percentage differences given in the Table 6-15. For both approaches, these differences tend to increase as more population nodes are added. The performance in the right diagram is almost the same for ex-ante and ex-post. Compared to the CE, it takes much more time for the GA to execute the code. For example, it takes the CE

100 milliseconds to run the algorithm with 10 population nodes, whereas the GA needs over 1,700 milliseconds. This is normal for small problem sizes, because the GA has much more code to execute than the CE. However, as the problem size grows, the GA is much faster and, given a particular problem (see section 6.5 ”Performance CE vs. GA”), provides a solution in e.g. about an hour, while it would take the CE many decades.

6.3.2.1.2. Comparison of Costs

Based on the rather small percentage differences (-1.87 to 2.95) in 1) in Table 6-16, it appears that the ex-ante and the ex-post solutions perform almost equally well in terms of costs. 2) and 3) are very similar, implying that both approaches perform equally well when plugged into the utilitarian objective function. Most cost values in this table are not much different from the values of the CE.

			Population Nodes					
			10	13	16	19	22	25
$\bar{\Delta}_{1)}$	1) Ex-Ante ($\bar{\theta}_A$)	Scenario 1	0.00	-1.87	-0.34	-0.03	1.76	2.21
	vs.	Scenario 2	1.41	2.24	1.81	0.26	1.92	1.09
	Ex-Post ($\bar{\theta}_P$)	Scenario 3	-0.23	2.95	0.75	1.63	1.76	1.46
$\bar{\Delta}_{2)}$	2) Ex-Ante ($\bar{\gamma}_A$)	Scenario 1	0.00	7.07	8.56	8.70	9.77	10.89
	vs.	Scenario 2	10.92	12.39	13.09	9.50	12.22	12.51
	Utilitarian ($\bar{\gamma}_U$)	Scenario 3	7.88	11.86	12.16	11.22	10.93	9.54
$\bar{\Delta}_{3)}$	3) Ex-Post ($\bar{\gamma}_P$)	Scenario 1	0.00	8.56	8.49	8.80	7.66	8.35
	vs.	Scenario 2	9.61	9.89	10.49	9.21	10.53	11.51
	Utilitarian ($\bar{\gamma}_U$)	Scenario 3	8.41	8.54	10.88	9.17	9.47	7.83

Table 6-16 GA: Population Nodes: Comparison of Costs (Ex-Ante, Ex-Post, and Utilitarian) in %

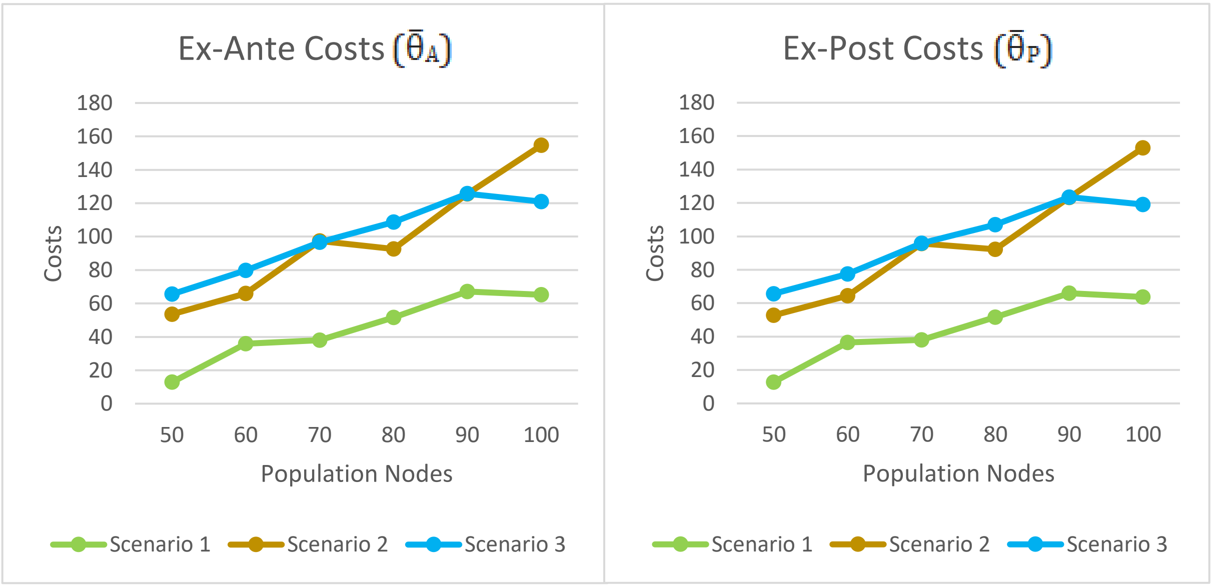


Figure 6-21 GA: Population Nodes: Ex-Ante and Ex-Post Costs

Figure 6-21 displays the costs of the three scenarios. Costs increase as more population nodes are added. Both graphs are very similar to the graphs of the CE.

6.3.2.2. Increasing Potential Locations

6.3.2.2.1. Fitness Values and Performance

Comparing the fitness values of Table 6-17 with Table 6-7 (CE), some values of $\bar{\lambda}_A$ and $\bar{\psi}_P$ of the GA are found to be slightly lower than those of the CE. In these cases, the best solution could not be found for all 100 seeds. However, this is negligible as the differences are very small.

		Potential Locations					
		8	11	14	17	20	23
$\bar{\lambda}_A$	Ex-Ante Fitness of Solution with Best Ex-Ante Fitness	9.08	10.38	11.64	12.76	13.63	14.93
$\bar{\lambda}_P$	Ex-Ante Fitness of Solution with Best Ex-Post Fitness	8.91	10.20	11.39	12.13	13.26	14.28
$\bar{\Delta}_A$	Difference Ante in %	1.85	1.65	2.13	4.93	2.72	4.33
$\bar{\psi}_P$	Ex-Post Fitness of Solution with Best Ex-Post Fitness	5.91	6.69	7.48	8.17	8.81	9.67
$\bar{\psi}_A$	Ex-Post Fitness of Solution with Best Ex-Ante Fitness	5.89	6.64	7.35	7.92	8.48	9.09
$\bar{\Delta}_P$	Difference Post in %	0.25	0.71	1.81	3.14	3.76	6.01

Table 6-17 GA: Potential Locations: Fitness Values

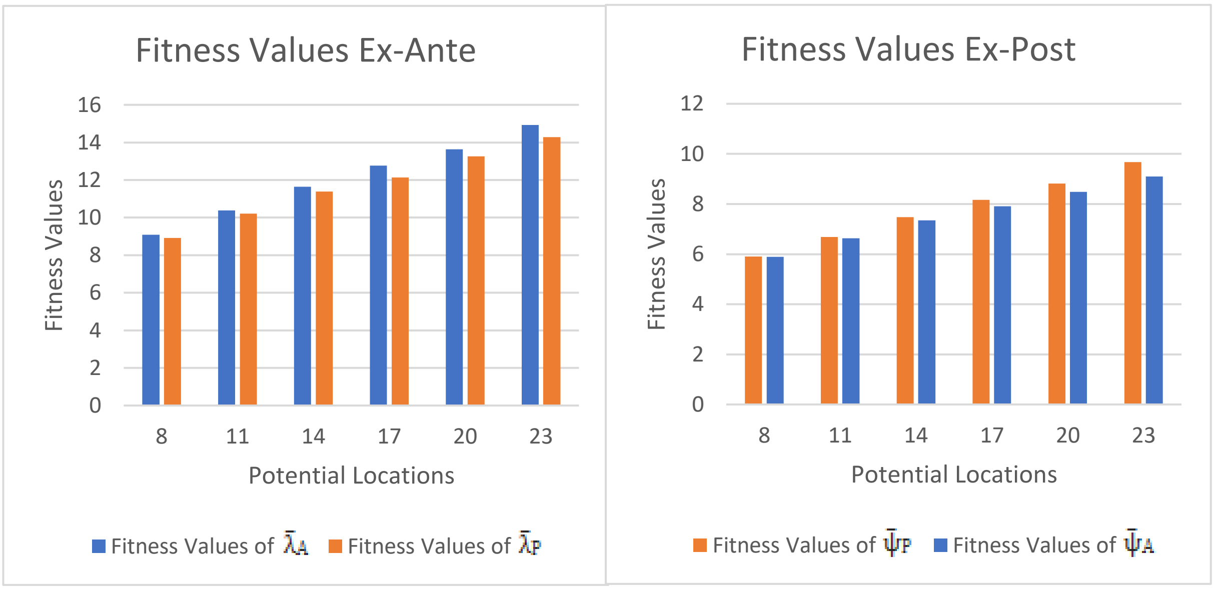


Figure 6-22 GA: Potential Locations: Fitness Values

Figure 6-22 displays the fitness values of the table above. These values increase for ex-ante and ex-post when more potential locations are available.

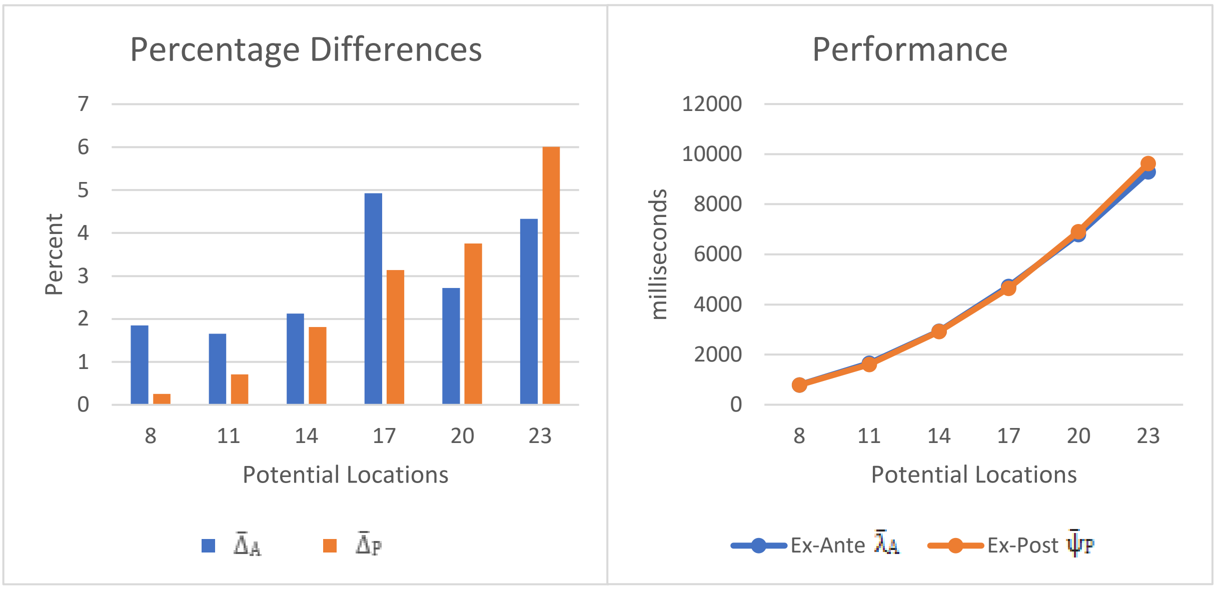


Figure 6-23 GA: Potential Locations: Percentage Differences and Performance

Although the percentage differences of ex-ante in Figure 6-23 do not show the same clear pattern as in Figure 6-9 (CE), they still increase as the number of potential locations grows. While the performance graph of the CE is clearly exponential, the graph of the GA is much flatter, but its code takes a lot more time to execute.

6.3.2.2.2. Comparison of Costs

The percentage differences given in Table 6-18 are quite similar to those of the CE, except for 2), which has much smaller differences. This means that the GA happens to deliver ex-ante solutions that lead to lower utilitarian costs than the ex-ante solutions of the CE.

			Potential Locations					
			8	11	14	17	20	23
$\bar{\Delta}_{1)}$	1) Ex-Ante ($\bar{\theta}_A$)	Scenario 1	0.00	0.00	0.00	0.00	0.00	0.00
	vs.	Scenario 2	-0.27	1.41	0.05	0.89	6.79	9.63
	Ex-Post ($\bar{\theta}_P$)	Scenario 3	-0.22	-0.23	2.14	1.73	3.09	6.96
$\bar{\Delta}_{2)}$	2) Ex-Ante ($\bar{\gamma}_A$)	Scenario 1	0.00	0.00	0.00	0.00	0.00	0.00
	vs.	Scenario 2	5.17	10.92	10.80	14.98	19.01	23.15
	Utilitarian ($\bar{\gamma}_U$)	Scenario 3	5.28	7.88	11.96	13.63	15.33	19.96
$\bar{\Delta}_{3)}$	3) Ex-Post ($\bar{\gamma}_P$)	Scenario 1	0.00	0.00	0.00	0.00	0.00	0.00
	vs.	Scenario 2	5.92	9.61	10.55	13.65	11.09	11.30
	Utilitarian ($\bar{\gamma}_U$)	Scenario 3	5.55	8.41	9.76	11.96	12.87	11.40

Table 6-18 GA: Potential Locations: Comparison of Costs (Ex-Ante, Ex-Post, and Utilitarian) in %

For all three scenarios and both evaluation approaches, the costs of the GA (Figure 6-24) and the CE (Figure 6-11) decrease as the number of potential locations increases.

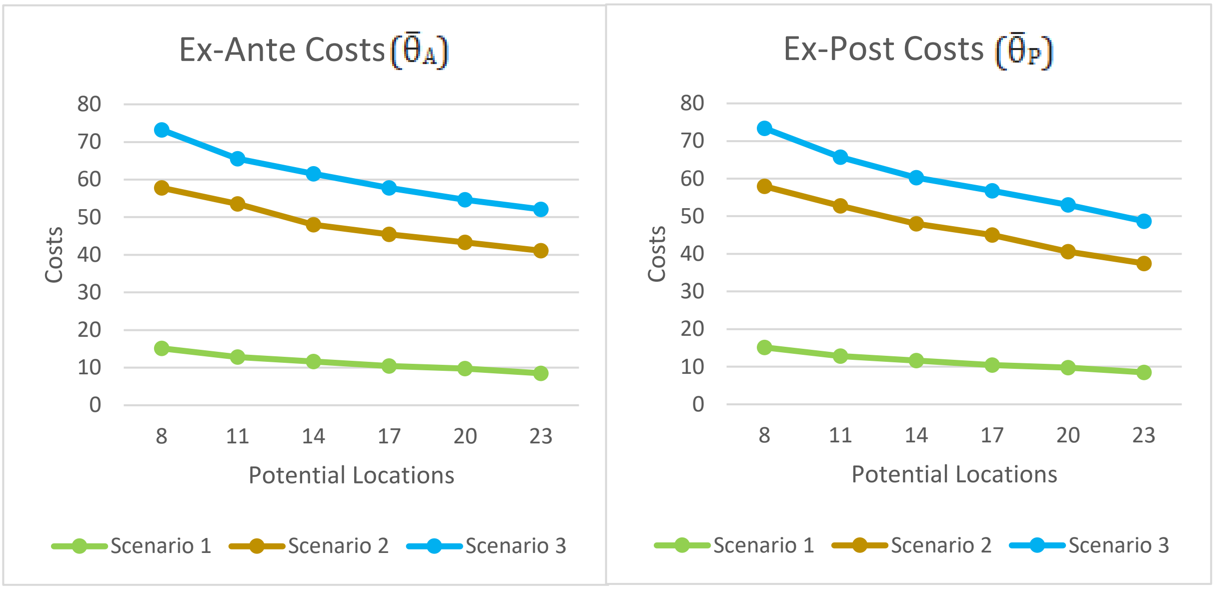


Figure 6-24 GA: Potential Locations: Ex-Ante and Ex-Post Costs

6.3.2.3. Increasing Permanent Shelters

6.3.2.3.1. Fitness Values and Performance

Again, the fitness values of $\bar{\lambda}_A$ and $\bar{\psi}_P$ are the same when Table 6-19 (GA) is compared with Table 6-10 (CE), while $\bar{\lambda}_P$ and $\bar{\psi}_A$ are slightly different.

		Permanent Shelters					
		1	2	3	4	5	6
$\bar{\lambda}_A$	Ex-Ante Fitness of Solution with Best Ex-Ante Fitness	10.22	10.38	10.38	10.38	10.38	10.38
$\bar{\lambda}_P$	Ex-Ante Fitness of Solution with Best Ex-Post Fitness	10.04	10.20	10.27	10.25	10.37	10.38
$\bar{\Delta}_A$	Difference Ante in %	1.88	1.65	1.05	1.23	0.02	0.00
$\bar{\psi}_P$	Ex-Post Fitness of Solution with Best Ex-Post Fitness	6.57	6.69	6.69	6.69	6.69	6.69
$\bar{\psi}_A$	Ex-Post Fitness of Solution with Best Ex-Ante Fitness	6.48	6.64	6.67	6.68	6.69	6.69
$\bar{\Delta}_P$	Difference Post in %	1.43	0.71	0.24	0.03	0.00	0.00

Table 6-19 GA: Permanent Shelters: Fitness Values

The diagrams of Figure 6-25 display the fitness values of the previous table and are quite similar to the diagrams of the CE. These values do not increase after two permanent shelters are built, and the values of $\bar{\lambda}_P$ and $\bar{\psi}_A$ are as good as the values of $\bar{\lambda}_A$ and $\bar{\psi}_P$, respectively, when five or six shelters are built.

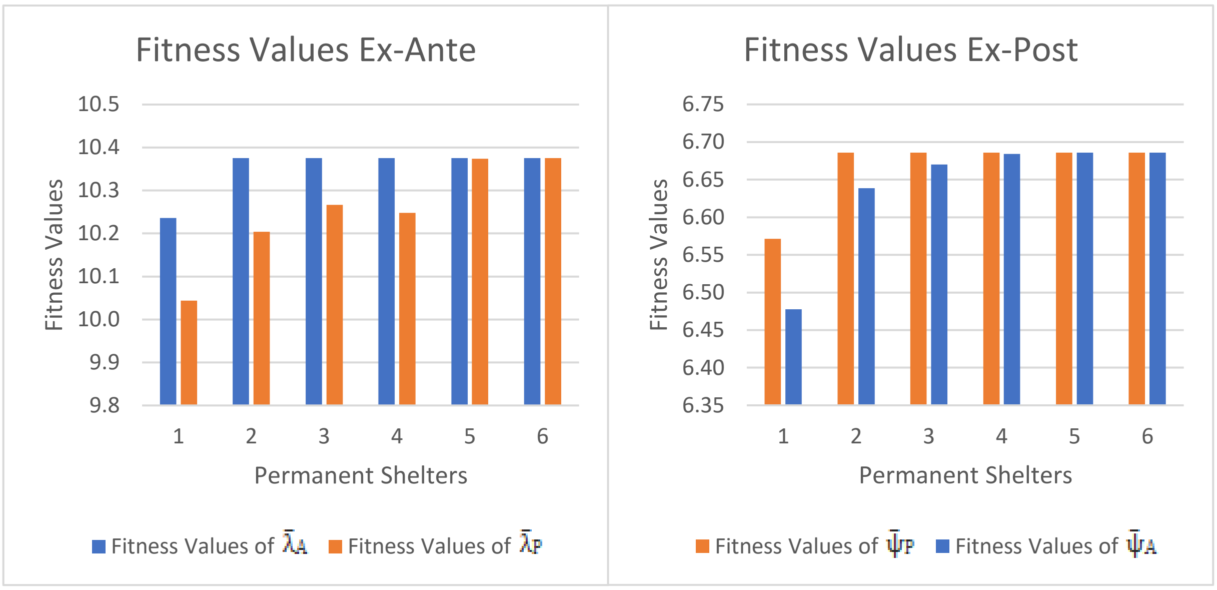


Figure 6-25 GA: Permanent Shelters: Fitness Values

The percentage differences $\bar{\Delta}_A$, illustrated in Figure 6-26, are mostly higher for the GA than for the CE, but for $\bar{\Delta}_P$, the differences are mostly lower for the GA. As more shelters are built, the percentage differences decrease.

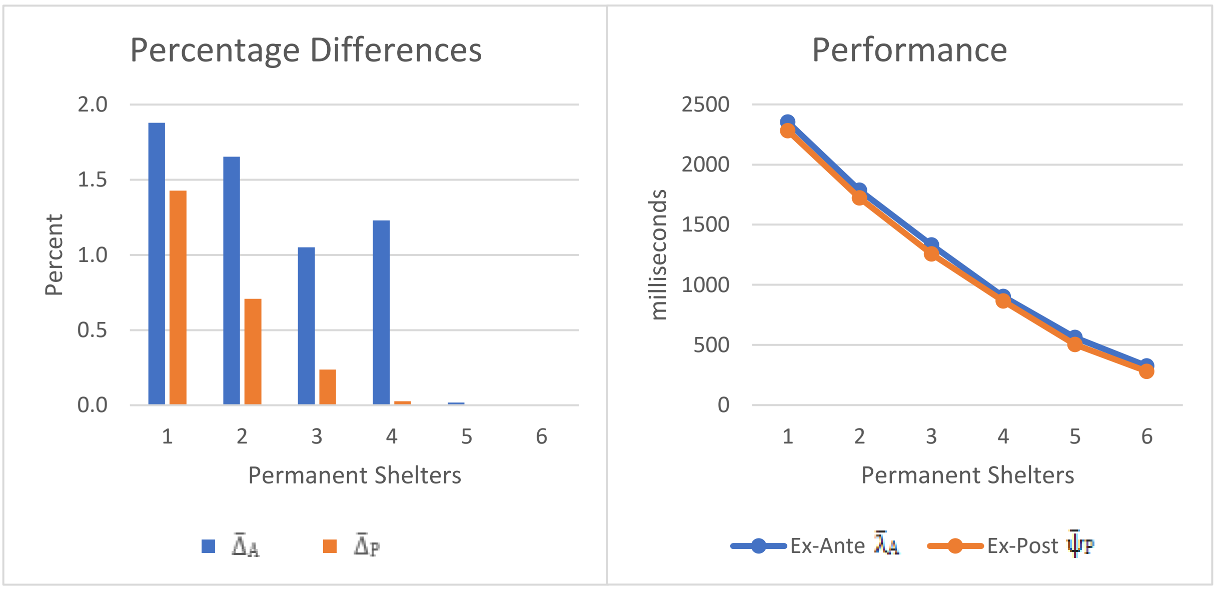


Figure 6-26 GA: Permanent Shelters: Percentage Differences and Performance

When comparing the performance graph of the GA and the CE (Figure 6-14), it becomes clear that the underlying calculations given in Table 6-20 (GA) and Table 6-11 (CE) are different. Since one chromosome equals one permanent combination and 10 chromosomes are used, the first row of Table 6-20 is always 10, which illustrates that not all possible permanent combinations are calculated for the GA.

	Permanent Shelters					
	1	2	3	4	5	6
Permanent Combinations	10	10	10	10	10	10
Temporary Combinations	21	15	10	6	3	1
Possible Combinations	210	150	100	60	30	10

Table 6-20 GA: Permanent Shelters: Permanent and Temporary Combinations

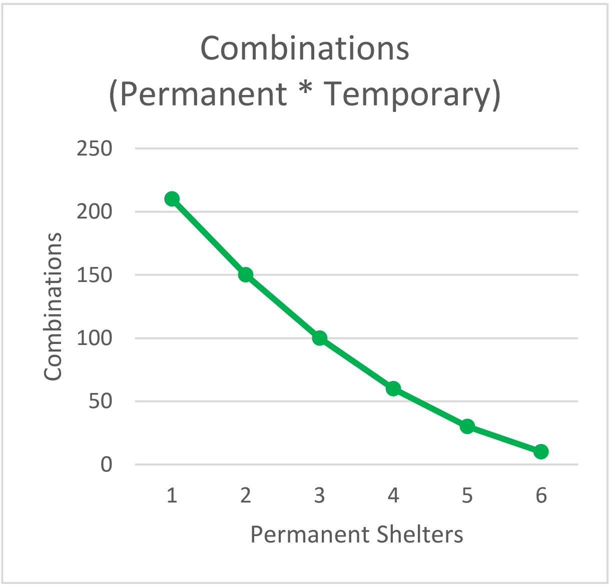


Figure 6-27 GA: Permanent Shelters: Combinations

Figure 6-27 visualizes the results of the table above. Again, the performance strongly depends on the number of combinations.

6.3.2.3.2. Comparison of Costs

The data of Table 6-21 is very similar to the data of the CE. In 1), the ex-ante evaluation performs slightly worse than the ex-ante evaluation of the CE. In 2) and 3), the worst percentage differences of the utilitarian costs are smaller than those of the CE. The differences decrease when more shelters are added.

			Permanent Shelters					
			1	2	3	4	5	6
$\bar{\Delta}_{1)}$	1) Ex-Ante ($\bar{\theta}_A$)	Scenario 1	0.00	0.00	0.00	0.00	0.00	0.00
	vs.	Scenario 2	1.28	1.41	0.70	-0.22	-0.02	0.00
	Ex-Post ($\bar{\theta}_P$)	Scenario 3	2.81	-0.23	-0.70	0.86	-0.47	0.00
$\bar{\Delta}_{2)}$	2) Ex-Ante ($\bar{\gamma}_A$)	Scenario 1	0.00	0.00	0.00	0.00	0.00	0.00
	vs.	Scenario 2	13.89	10.92	8.30	5.39	1.21	0.00
	Utilitarian ($\bar{\gamma}_U$)	Scenario 3	11.86	7.88	7.48	4.85	2.07	0.00
$\bar{\Delta}_{3)}$	3) Ex-Post ($\bar{\gamma}_P$)	Scenario 1	0.00	0.00	0.00	0.00	0.00	0.00
	vs.	Scenario 2	11.81	9.61	7.72	5.83	1.22	0.00
	Utilitarian ($\bar{\gamma}_U$)	Scenario 3	9.44	8.41	8.07	4.05	2.49	0.00

Table 6-21 GA: Permanent Shelters: Comparison of Costs (Ex-Ante, Ex-Post, and Utilitarian) in %

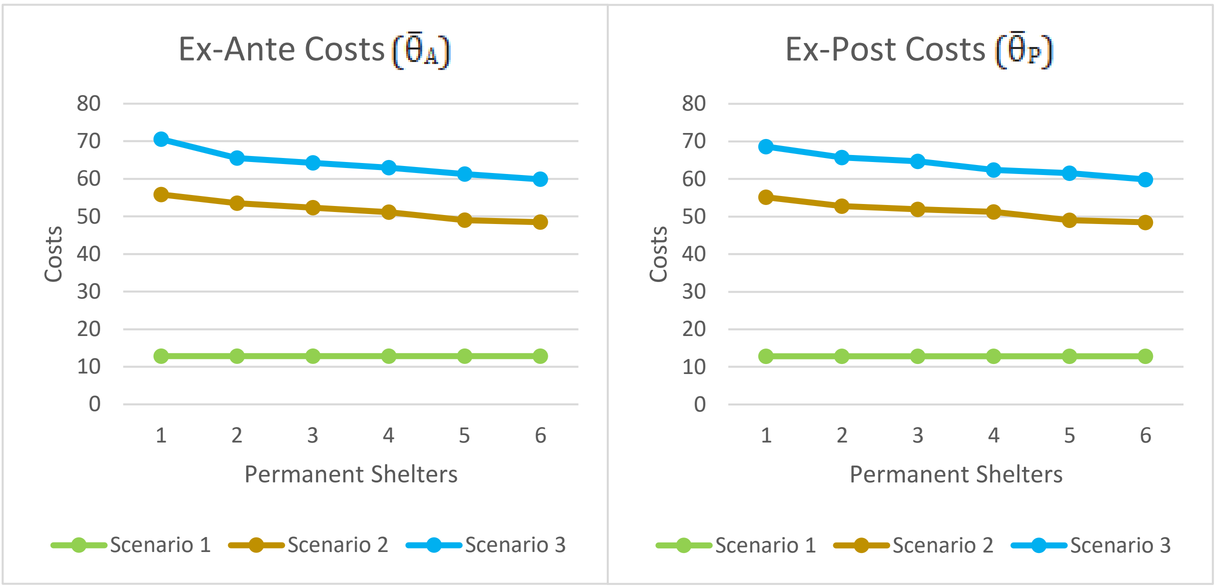


Figure 6-28 GA: Permanent Shelters: Ex-Ante and Ex-Post Costs

Figure 6-28 displays the distances/costs of ex-ante and ex-post. The graphs of both diagrams tend to decrease when more permanent shelters are added, and are almost identical to the graphs of the CE.

6.3.2.4. Increasing Temporary Shelters

6.3.2.4.1. Fitness Values and Performance

The GA again found the best solutions for $\bar{\lambda}_A$ and $\bar{\psi}_P$ as given in Table 6-22. $\bar{\lambda}_P$ and $\bar{\psi}_A$ show negligible differences compared to the CE.

		Temporary Shelters					
		1	2	3	4	5	6
$\bar{\lambda}_A$	Ex-Ante Fitness of Solution with Best Ex-Ante Fitness	10.21	10.38	10.38	10.38	10.38	10.38
$\bar{\lambda}_P$	Ex-Ante Fitness of Solution with Best Ex-Post Fitness	9.88	10.20	10.22	10.27	10.33	10.38
$\bar{\Delta}_A$	Difference Ante in %	3.23	1.65	1.51	1.02	0.48	0.00
$\bar{\psi}_P$	Ex-Post Fitness of Solution with Best Ex-Post Fitness	6.55	6.69	6.69	6.69	6.69	6.69
$\bar{\psi}_A$	Ex-Post Fitness of Solution with Best Ex-Ante Fitness	6.25	6.64	6.68	6.69	6.69	6.69
$\bar{\Delta}_P$	Difference Post in %	4.52	0.71	0.04	0.00	0.00	0.00

Table 6-22 GA: Temporary Shelters: Fitness Values

Both diagrams of Figure 6-29 and the first diagram of Figure 6-30 are very similar to the CE. The fitness values increase as more temporary shelters are built. The percentage differences decrease because the ex-ante and ex-post solutions become more similar as more shelters are built and when only eight potential locations are available.

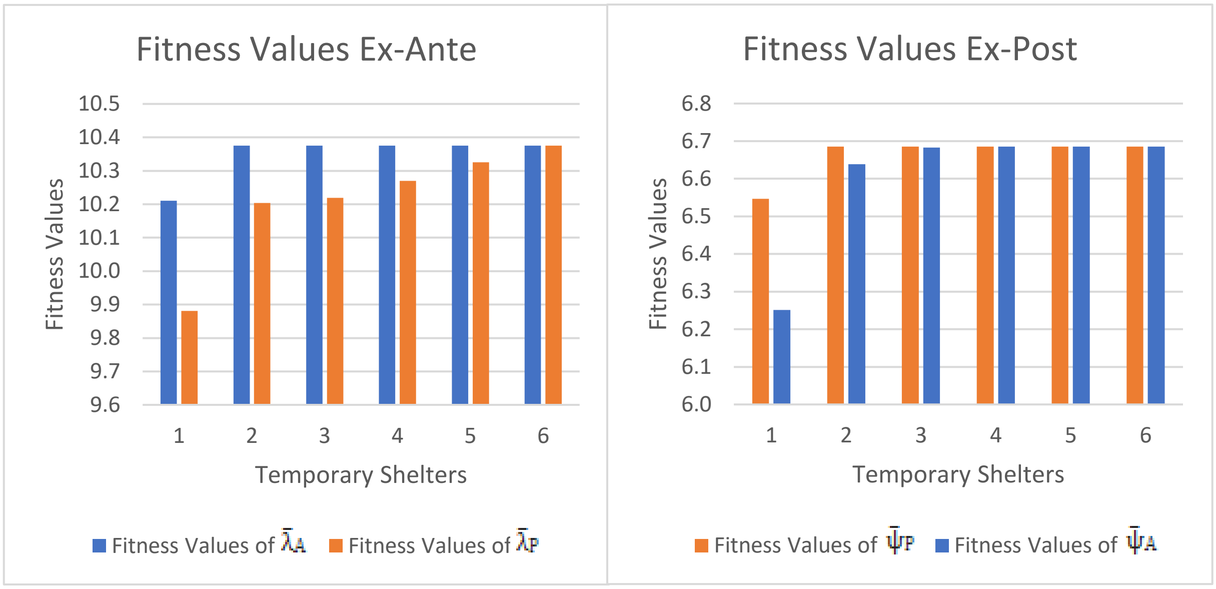


Figure 6-29 GA: Temporary Shelters: Fitness Values

Unlike the performance graph of Figure 6-26 (increasing permanent shelters), the performance graph of the GA in Figure 6-30 has the same shape as the graph of the CE (Figure 6-17) because the complete enumeration still has to be applied to calculate all combinations of temporary shelters.

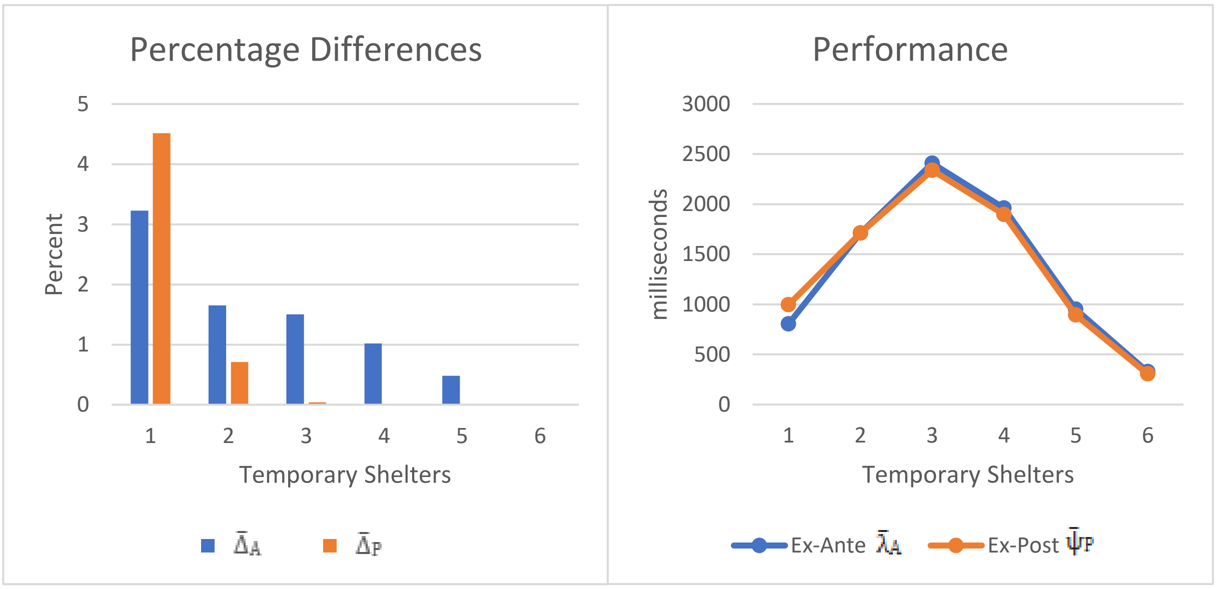


Figure 6-30 GA: Temporary Shelters: Percentage Differences and Performance

6.3.2.4.2. Comparison of Costs

The costs given in Table 6-23 are very similar to those of the CE for the most part, except for 2): the ex-ante utilitarian costs for one shelter are much lower for the GA than for the CE.

			Temporary Shelters					
			1	2	3	4	5	6
$\bar{\Delta}_{1)}$	1) Ex-Ante ($\bar{\theta}_A$)	Scenario 1	0.00	0.00	0.00	0.00	0.00	0.00
	vs.	Scenario 2	7.65	1.41	0.17	-0.39	-0.23	0.00
	Ex-Post ($\bar{\theta}_P$)	Scenario 3	2.88	-0.23	-0.74	-0.37	-0.28	0.00
$\bar{\Delta}_{2)}$	2) Ex-Ante ($\bar{\gamma}_A$)	Scenario 1	0.00	0.00	0.00	0.00	0.00	0.00
	vs.	Scenario 2	16.29	10.92	7.92	4.38	1.36	0.00
	Utilitarian ($\bar{\gamma}_U$)	Scenario 3	12.37	7.88	6.27	4.07	1.87	0.00
$\bar{\Delta}_{3)}$	3) Ex-Post ($\bar{\gamma}_P$)	Scenario 1	0.00	0.00	0.00	0.00	0.00	0.00
	vs.	Scenario 2	7.93	9.61	7.14	4.94	1.56	0.00
	Utilitarian ($\bar{\gamma}_U$)	Scenario 3	8.07	8.41	7.04	4.42	2.18	0.00

Table 6-23 GA: Temporary Shelters: Comparison of Costs (Ex-Ante, Ex-Post, and Utilitarian) in %

For both evaluation approaches and for all three scenarios, the costs of the GA given in Figure 6-31 are nearly identical to the costs of the CE.

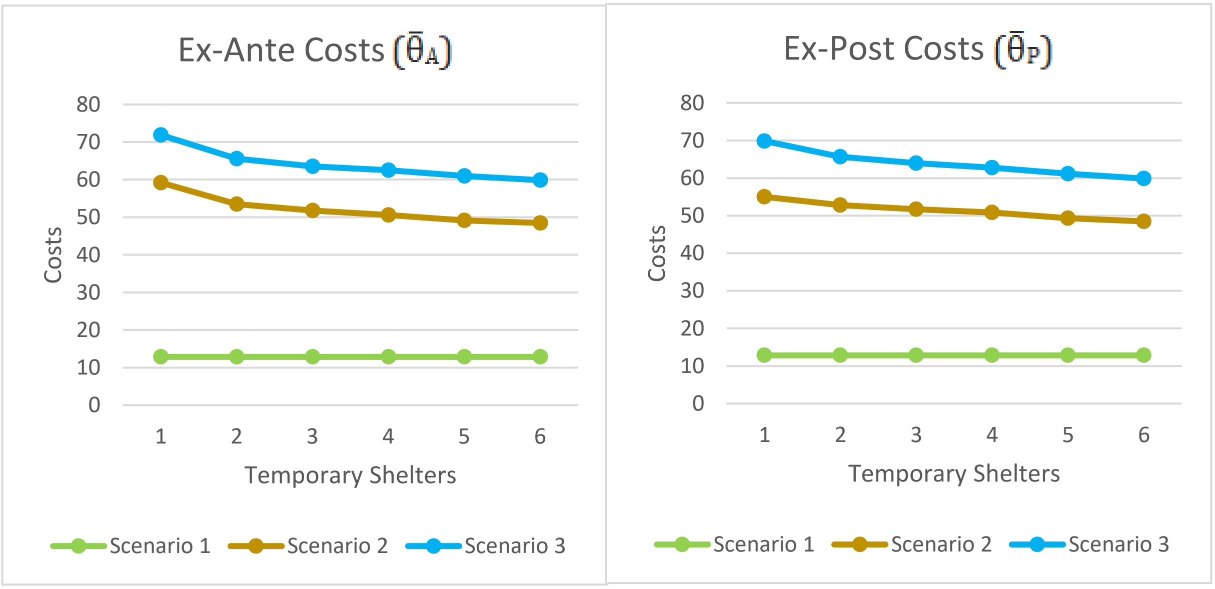


Figure 6-31 GA: Temporary Shelters: Ex-Ante and Ex-Post Costs

In conclusion, the fitness values of the GA are identical to the fitness values of the CE and only slightly worse in some cases. This proves that the GA succeeds in finding the best ex-ante and ex-post solution for almost every seed when the problem size is small.

6.4. Large Problem Size

In this section, the genetic algorithm (GA) is applied to a larger problem size. The settings are given in 6.3 “Small Problem Size”. All four combinations of the two crossover operators (one-point and two-point) and the two selection methods (roulette wheel and tournament) are compared:

a) One-point crossover and roulette wheel selection,
b) one-point crossover and tournament selection,
c) two-point crossover and roulette wheel selection, and
d) two-point crossover and tournament selection.

After either 300 generations or 50 consecutive generations without improvement, the algorithm terminates. Since the behavior of increasing population nodes, permanent shelters, and temporary shelters has already been observed in the previous section, only the number of potential locations is discussed for the large problem size, as increasing the latter causes high additional computation times, allowing the GA to demonstrate its efficiency.

6.4.1. Fitness Values and Performance

According to Table 6-24, the best fitness values for the ex-ante and ex-post evaluation approaches for an increasing number of potential locations are obtained when the two-point crossover in combination with the roulette wheel selection (c)) is chosen. The second-best results for both approaches are obtained with the one-point crossover and the roulette wheel selection (a)), the third place is given to table d) (two-point crossover and tournament selection) and the fourth to table b) (one-point crossover and tournament selection). This ranking implies that the tournament selection is generally worse than the roulette wheel selection for this particular problem.

a)		Potential Locations					
		50	60	70	80	90	100
$\bar{\lambda}_A$	Ex-Ante Fitness of Solution with Best Ex-Ante Fitness	2.49	2.62	2.61	2.70	2.70	2.74
$\bar{\lambda}_P$	Ex-Ante Fitness of Solution with Best Ex-Post Fitness	2.27	2.34	2.40	2.44	2.38	2.43
$\bar{\Delta}_A$	Difference Ante in %	9.01	10.74	8.07	9.77	11.94	11.19
$\bar{\psi}_P$	Ex-Post Fitness of Solution with Best Ex-Post Fitness	1.67	1.76	1.76	1.81	1.80	1.81
$\bar{\psi}_A$	Ex-Post Fitness of Solution with Best Ex-Ante Fitness	1.54	1.64	1.60	1.68	1.68	1.71
$\bar{\Delta}_P$	Difference Post in %	7.78	6.64	9.13	7.00	6.25	5.77

b)

		Potential Locations					
		50	60	70	80	90	100
$\bar{\lambda}_A$	Ex-Ante Fitness of Solution with Best Ex-Ante Fitness	2.42	2.51	2.57	2.63	2.56	2.65
$\bar{\lambda}_P$	Ex-Ante Fitness of Solution with Best Ex-Post Fitness	2.17	2.28	2.32	2.39	2.34	2.41
$\bar{\Delta}_A$	Difference Ante in %	10.39	9.33	9.98	8.90	8.69	9.08
$\bar{\psi}_P$	Ex-Post Fitness of Solution with Best Ex-Post Fitness	1.62	1.70	1.70	1.78	1.75	1.76
$\bar{\psi}_A$	Ex-Post Fitness of Solution with Best Ex-Ante Fitness	1.50	1.59	1.59	1.64	1.61	1.67
$\bar{\Delta}_P$	Difference Post in %	7.52	6.34	6.55	8.15	7.92	5.09

c)

		Potential Locations					
		50	60	70	80	90	100
$\bar{\lambda}_A$	Ex-Ante Fitness of Solution with Best Ex-Ante Fitness	2.52	2.65	2.66	2.79	2.75	2.79
$\bar{\lambda}_P$	Ex-Ante Fitness of Solution with Best Ex-Post Fitness	2.29	2.39	2.40	2.53	2.44	2.51
$\bar{\Delta}_A$	Difference Ante in %	9.08	9.76	9.77	9.42	11.40	9.92
$\bar{\psi}_P$	Ex-Post Fitness of Solution with Best Ex-Post Fitness	1.69	1.77	1.77	1.84	1.84	1.84
$\bar{\psi}_A$	Ex-Post Fitness of Solution with Best Ex-Ante Fitness	1.55	1.64	1.63	1.69	1.69	1.74
$\bar{\Delta}_P$	Difference Post in %	7.87	7.59	7.97	8.21	8.09	5.78

d)

		Potential Locations					
		50	60	70	80	90	100
$\bar{\lambda}_A$	Ex-Ante Fitness of Solution with Best Ex-Ante Fitness	2.45	2.54	2.61	2.62	2.60	2.68
$\bar{\lambda}_P$	Ex-Ante Fitness of Solution with Best Ex-Post Fitness	2.21	2.28	2.35	2.39	2.32	2.44
$\bar{\Delta}_A$	Difference Ante in %	9.69	10.40	10.01	8.74	10.67	9.15
$\bar{\psi}_P$	Ex-Post Fitness of Solution with Best Ex-Post Fitness	1.64	1.72	1.72	1.76	1.77	1.80
$\bar{\psi}_A$	Ex-Post Fitness of Solution with Best Ex-Ante Fitness	1.51	1.60	1.59	1.64	1.63	1.69
$\bar{\Delta}_P$	Difference Post in %	7.73	7.45	7.70	6.95	8.07	6.08

Table 6-24 Large GA: Fitness Values

Figure 6-32 displays the ex-ante fitness values ($\bar{\lambda}_A$) of Table 6-24. These values increase slightly as the number of potential locations increases. As mentioned before, the settings of c) provide the best results.

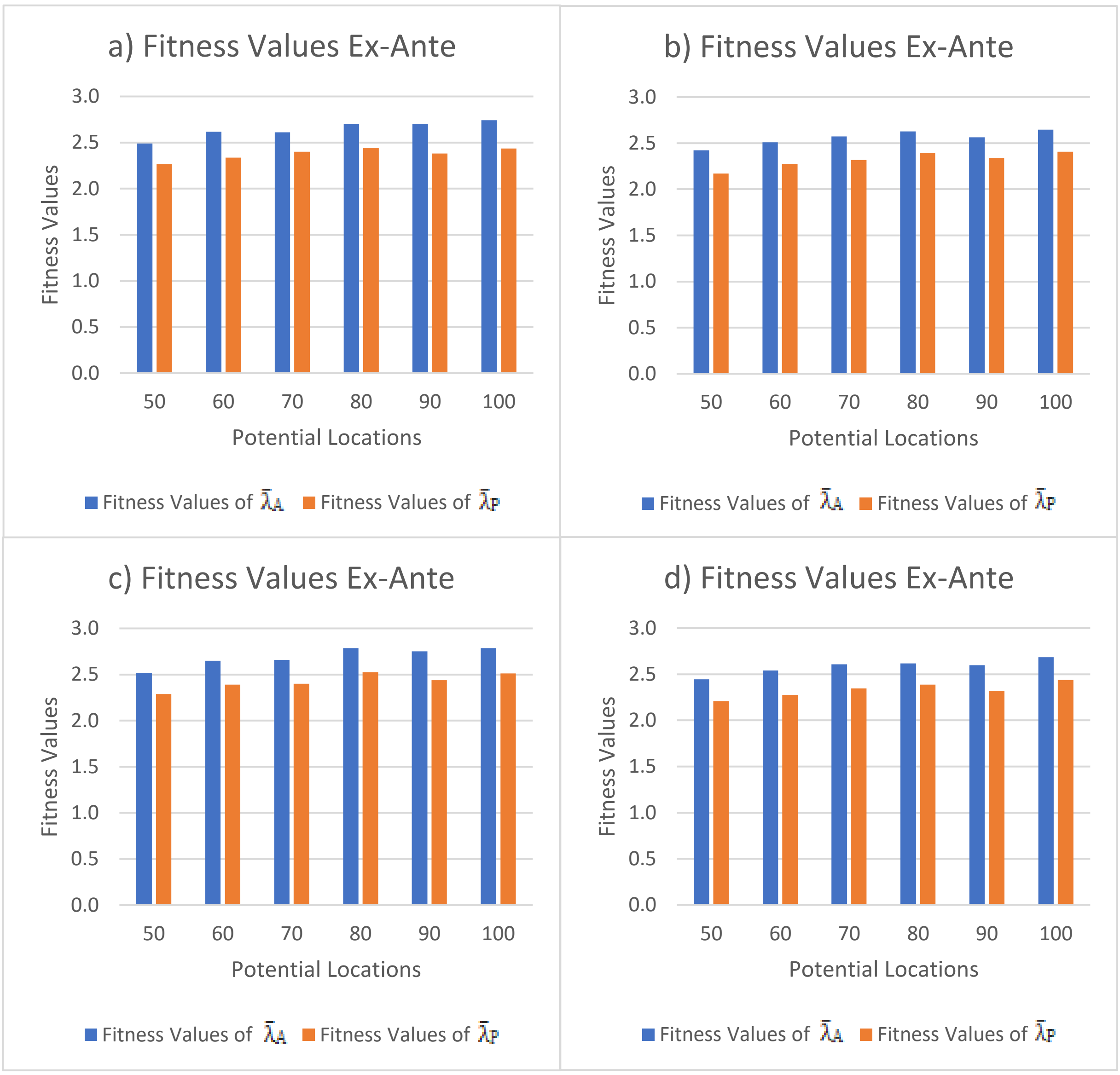
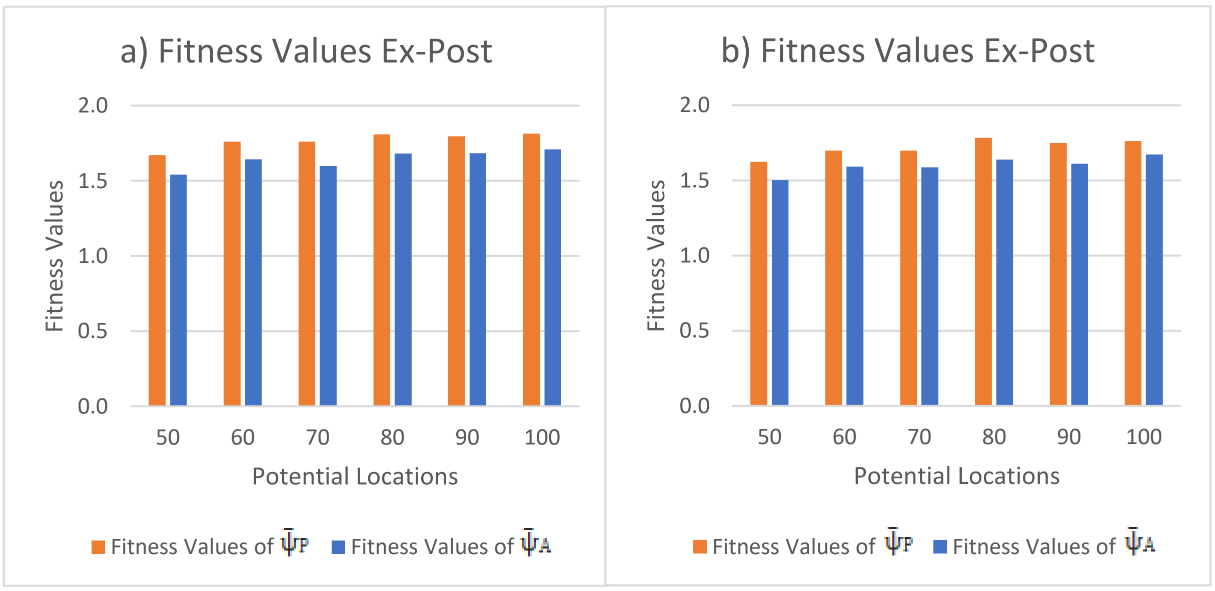


Figure 6-32 Large GA: Fitness Values Ex-Ante, a)-d) refer to Settings a)-d) given at the beginning of section 6.4



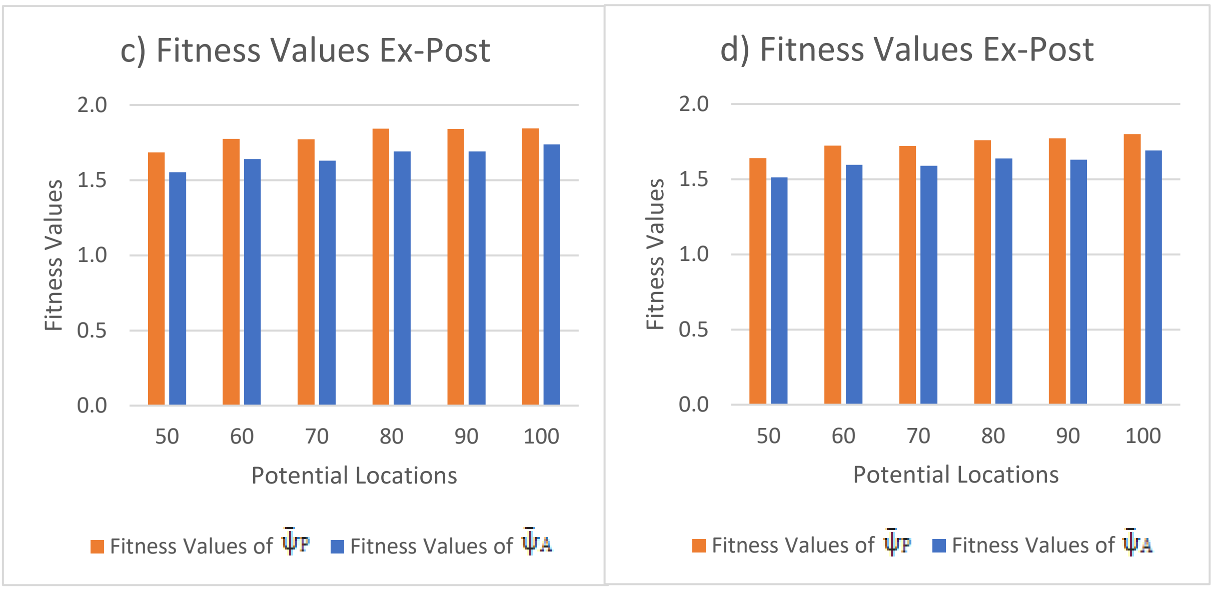
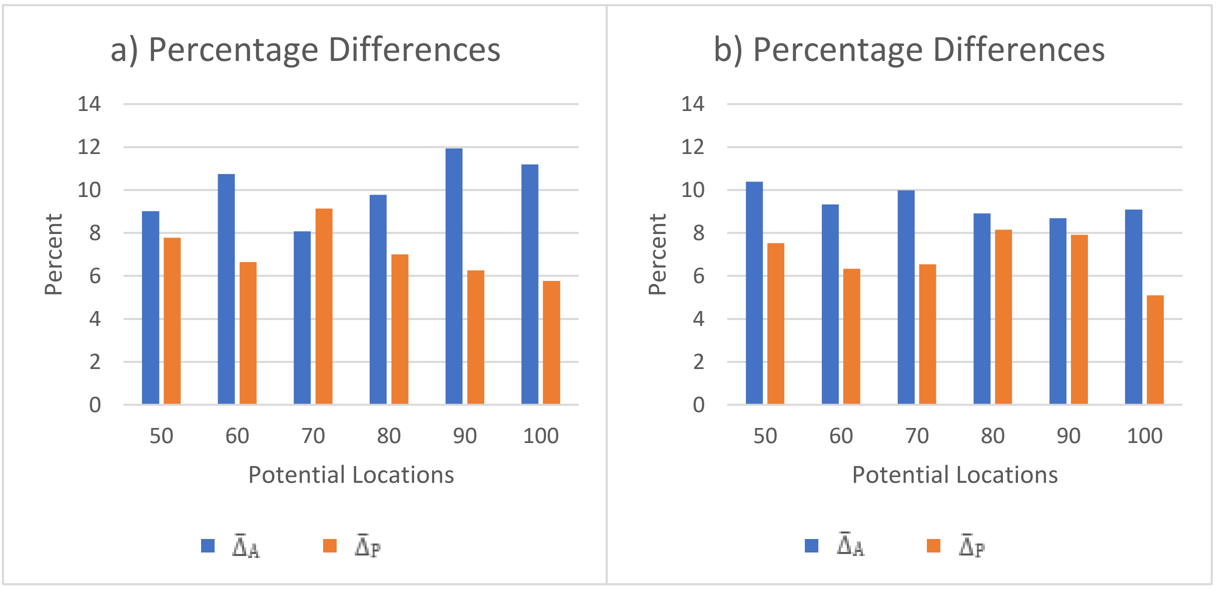


Figure 6-33 Large GA: Fitness Values Ex-Post

The same pattern can be observed in Figure 6-33: The ex-post fitness values are higher the more potential locations are available. Again, c) provides the best fitness values.

Figure 6-34 visualizes the percentage differences of ex-ante and ex-post. There seems to be no specific behavior when the number of potential locations increases. The percentage difference of ex-ante is almost always larger than the percentage difference of ex-post, which means that the best ex-post solution performs worse when plugged into the ex-ante objective function ($\bar{\lambda}_P$) than the best ex-ante solution when plugged into the ex-post objective function ($\bar{\psi}_A$).



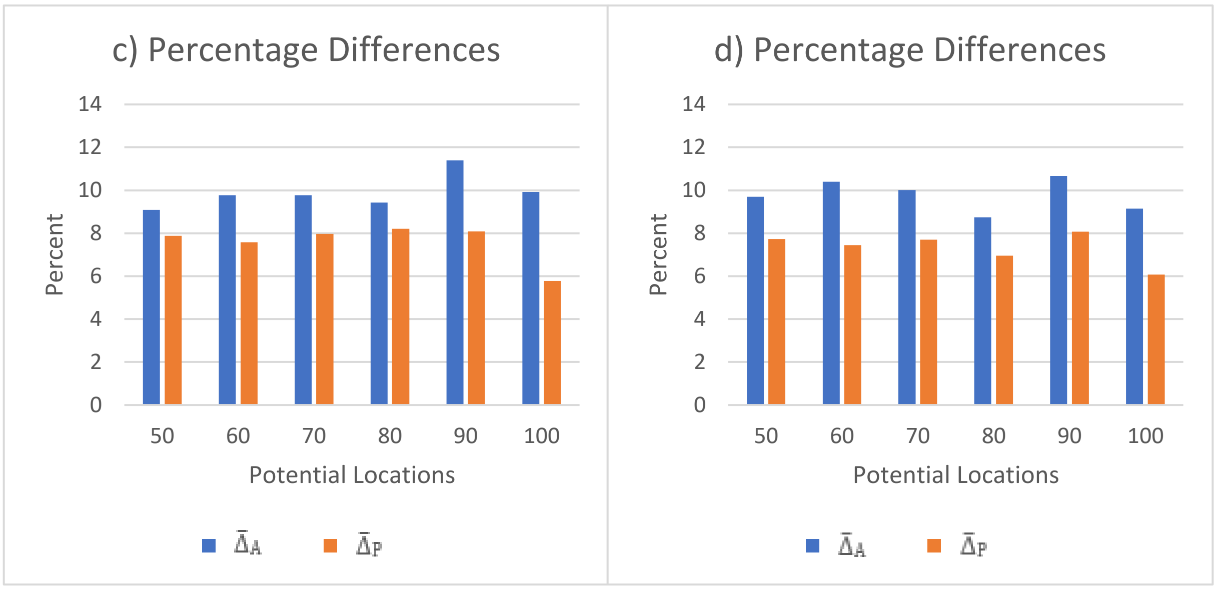


Figure 6-34 Large GA: Percentage Differences

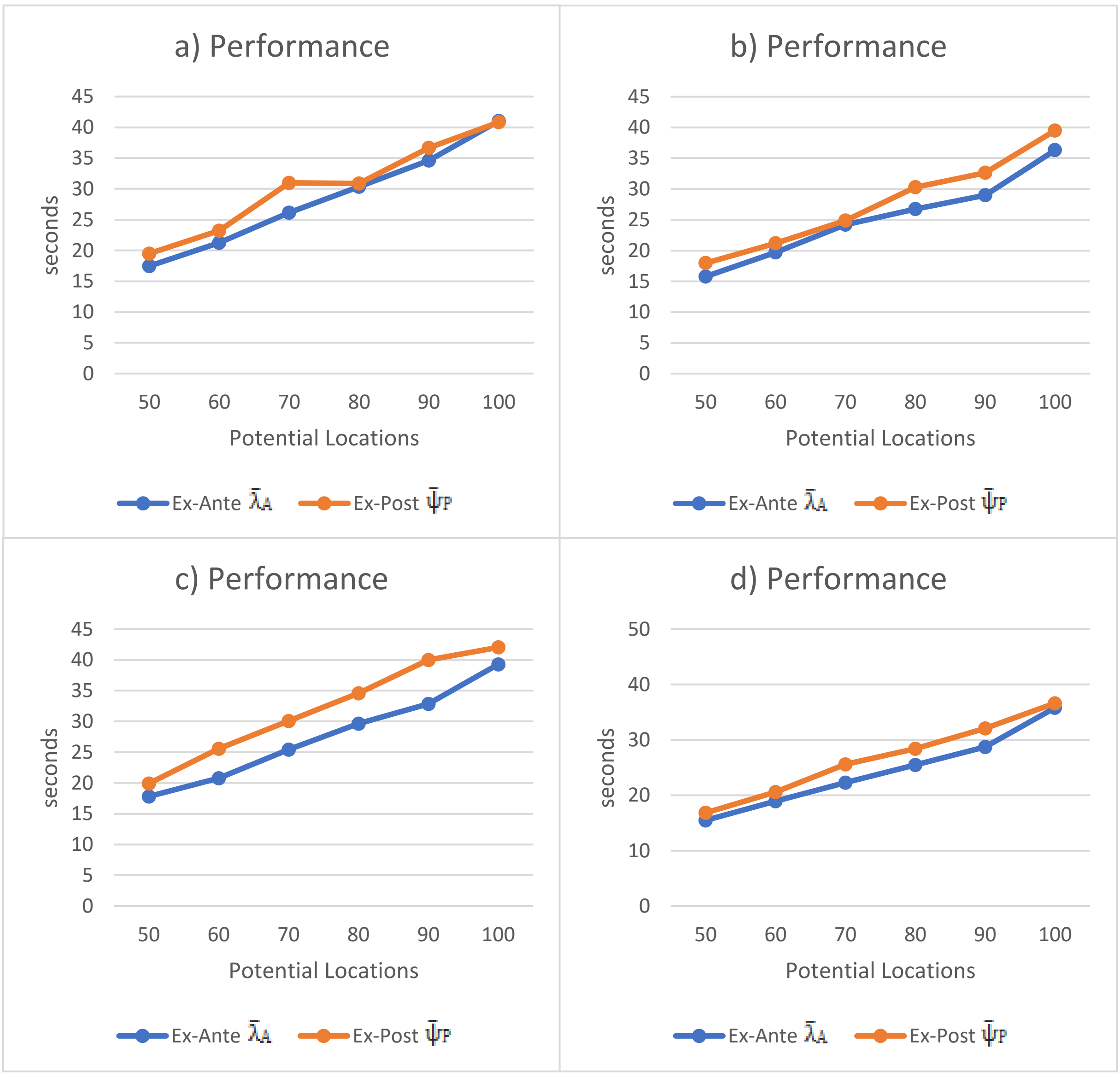


Figure 6-35 Large GA: Performance

Figure 6-35 shows the performance of the different settings. The tournament selection (visualized in b and d) appears to be faster than the roulette wheel selection (displayed in a and c). Ex-post generally performs worse than ex-ante because on average more generations are needed to obtain the best solution. Interestingly, the setting that delivers the highest fitness value is the slowest (c), whereas the setting that leads to the worst fitness value is the fastest (d).

In terms of the number of generations, it can be seen that the variances, which are displayed in Figure 6-36, of the ex-ante solutions are mostly (much) smaller than the variances of the ex-post solutions. In addition, the variances of the tournament selection (diagrams b and d) are generally smaller than those of the roulette wheel selection (diagrams a and c).

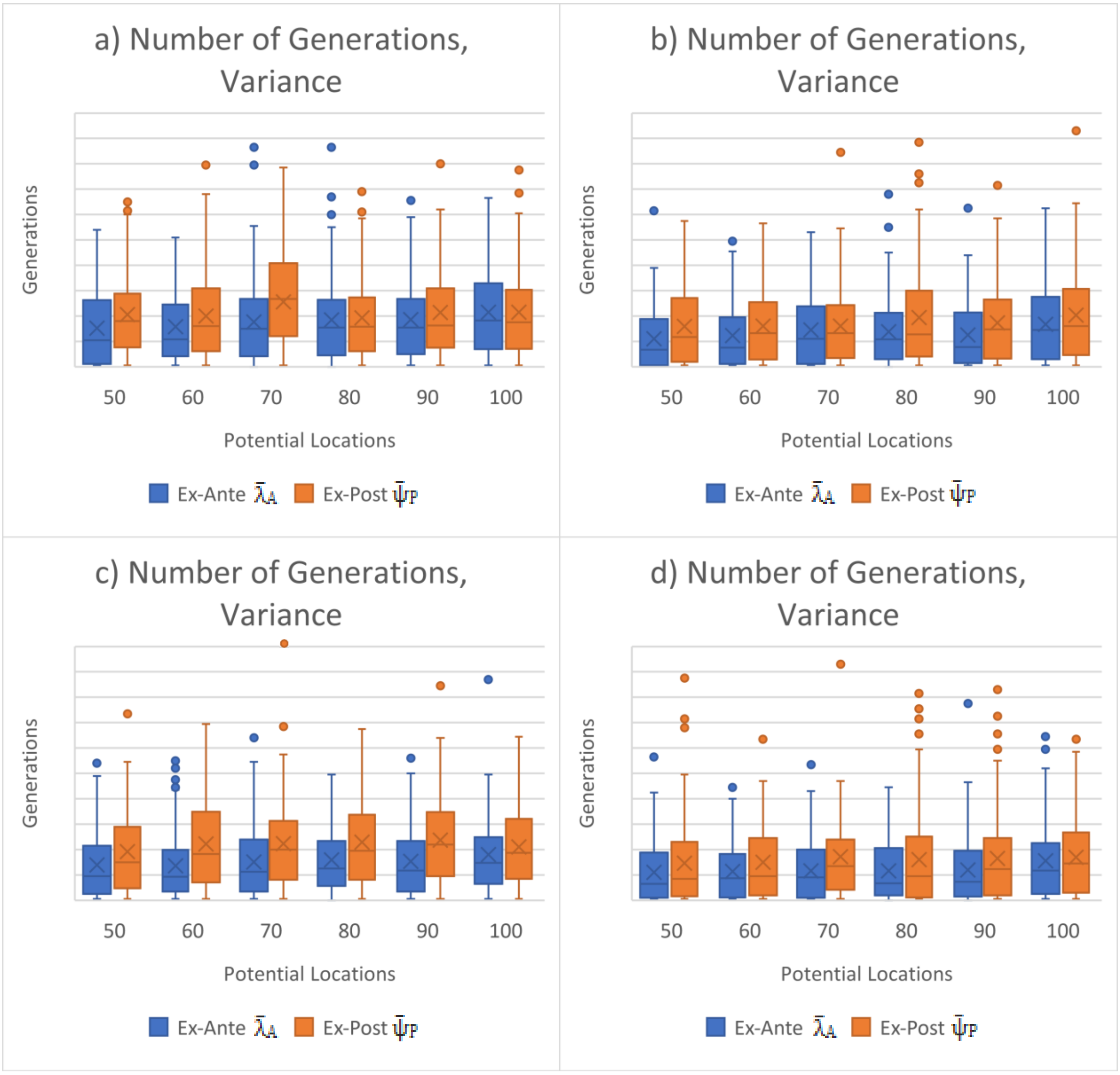


Figure 6-36 Large GA: Variance in Numbers of Generations

According to the data underlying these diagrams (given in Appendix C), 50% of the seeds deliver the solution after roughly 20 to 30 generations. In most cases, the ex-post approach needs more generations than the ex-ante approach to find the best solution. Furthermore, the number of generations has a positively skewed distribution for all four combinations and both evaluation approaches, which means that very high numbers of generations are rather rare. The outliers do not seem to follow any pattern, neither for the ex-ante nor for the ex-post approach, nor for the different combinations.

6.4.2. Comparison of Costs

In Table 6-25 the percentage differences in 1) show the same behavior for all combinations: The ex-ante costs are up to 16.46% worse than the ex-post costs in scenario 1, regardless of the number of potential locations available, and in scenario 2, ex-ante still has worse costs than ex-post, but here the differences decrease with a higher number of potential locations.

When the ex-ante utilitarian costs ($\bar{\gamma}_A$) are compared to the costs of the utilitarian objective function ($\bar{\gamma}_U$), $\bar{\gamma}_A$ is up to 31.16% worse than $\bar{\gamma}_U$ in scenario 1. These differences are smaller in scenario 2.

The percentage differences of the ex-post utilitarian costs and the costs of the utilitarian objective function in 3) are usually smaller than the percentage differences in 2). This means that ex-post delivers better solutions than ex-ante in terms of utilitarian costs.

a)			Potential Locations					
			50	60	70	80	90	100
$\bar{\Delta}_{1)}$	1) Ex-Ante ($\bar{\theta}_A$) vs. Ex-Post ($\bar{\theta}_P$)	Scenario 1	9.11	13.47	14.89	8.34	13.20	10.58
		Scenario 2	2.98	4.76	5.77	2.36	0.91	0.39
$\bar{\Delta}_{2)}$	2) Ex-Ante ($\bar{\gamma}_A$) vs. Utilitarian ($\bar{\gamma}_U$)	Scenario 1	22.05	24.90	28.33	25.31	29.36	23.52
		Scenario 2	15.64	17.74	18.04	13.84	13.34	14.76
$\bar{\Delta}_{3)}$	3) Ex-Post ($\bar{\gamma}_P$) vs. Utilitarian ($\bar{\gamma}_U$)	Scenario 1	10.10	9.73	10.45	13.80	13.18	11.96
		Scenario 2	10.76	12.67	10.06	10.12	11.50	13.46

b)			Potential Locations					
			50	60	70	80	90	100
$\bar{\Delta}_{1)}$	1) Ex-Ante ($\bar{\theta}_A$) vs. Ex-Post ($\bar{\theta}_P$)	Scenario 1	14.16	10.65	11.04	9.88	13.79	14.66
		Scenario 2	4.04	3.72	3.63	2.98	4.37	1.10
$\bar{\Delta}_{2)}$	2) Ex-Ante ($\bar{\gamma}_A$) vs. Utilitarian ($\bar{\gamma}_U$)	Scenario 1	19.56	24.93	23.47	19.72	29.99	25.94
		Scenario 2	17.53	14.22	15.03	14.04	17.06	15.91
$\bar{\Delta}_{3)}$	3) Ex-Post ($\bar{\gamma}_P$) vs. Utilitarian ($\bar{\gamma}_U$)	Scenario 1	3.96	10.15	11.37	9.74	14.46	10.44
		Scenario 2	12.66	10.08	10.49	10.58	12.79	12.71

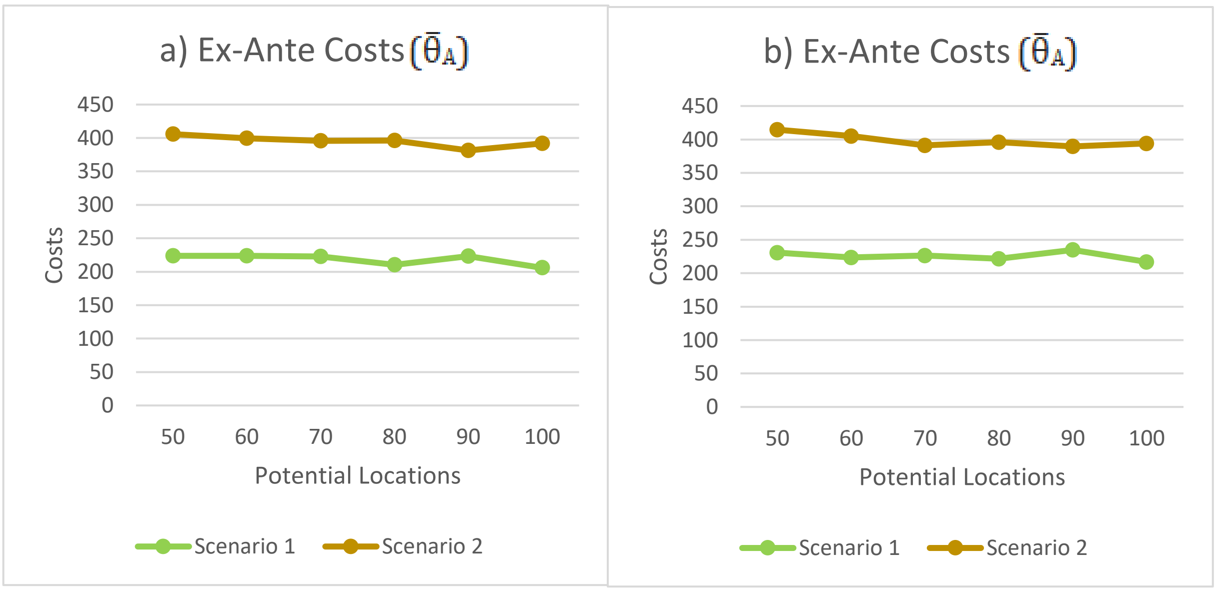
c)			Potential Locations					
			50	60	70	80	90	100
$\bar{\Delta}_{1)}$	1) Ex-Ante ($\bar{\theta}_A$)	Scenario 1	16.46	15.15	15.25	14.91	14.91	12.45
	vs. Ex-Post ($\bar{\theta}_P$)	Scenario 2	6.02	4.86	-0.08	3.86	4.43	2.44
$\bar{\Delta}_{2)}$	2) Ex-Ante ($\bar{\gamma}_A$)	Scenario 1	24.18	25.27	26.53	26.43	31.16	25.73
	vs. Utilitarian ($\bar{\gamma}_U$)	Scenario 2	17.95	15.71	12.49	17.21	17.67	15.94
$\bar{\Delta}_{3)}$	3) Ex-Post ($\bar{\gamma}_P$)	Scenario 1	6.81	8.45	9.11	8.98	13.37	12.75
	vs. Utilitarian ($\bar{\gamma}_U$)	Scenario 2	11.29	10.65	12.51	11.09	11.53	11.57

d)			Potential Locations					
			50	60	70	80	90	100
$\bar{\Delta}_{1)}$	1) Ex-Ante ($\bar{\theta}_A$)	Scenario 1	6.18	12.45	10.52	7.64	15.44	4.71
	vs. Ex-Post ($\bar{\theta}_P$)	Scenario 2	6.01	3.14	3.35	3.99	1.64	4.92
$\bar{\Delta}_{2)}$	2) Ex-Ante ($\bar{\gamma}_A$)	Scenario 1	17.14	22.58	21.82	23.90	26.74	19.77
	vs. Utilitarian ($\bar{\gamma}_U$)	Scenario 2	16.59	15.35	16.06	15.52	15.73	19.43
$\bar{\Delta}_{3)}$	3) Ex-Post ($\bar{\gamma}_P$)	Scenario 1	10.73	7.92	9.01	14.06	10.17	13.44
	vs. Utilitarian ($\bar{\gamma}_U$)	Scenario 2	10.62	11.96	13.08	10.89	12.15	12.77

Table 6-25 Large GA: Comparison of Costs (Ex-Ante, Ex-Post, and Utilitarian) in %

Figure 6-37 shows the ex-ante costs of all four combinations and both scenarios. These costs differ only slightly and there is little improvement when more potential locations are available. This is the case when the additional potential locations are placed in a way that they do not benefit the affected population nodes.

Similar to the ex-ante costs, the ex-post costs in Figure 6-38 hardly differ from each other and do not improve when additional potential locations are added. The ex-post costs are generally slightly lower than the ex-ante costs.



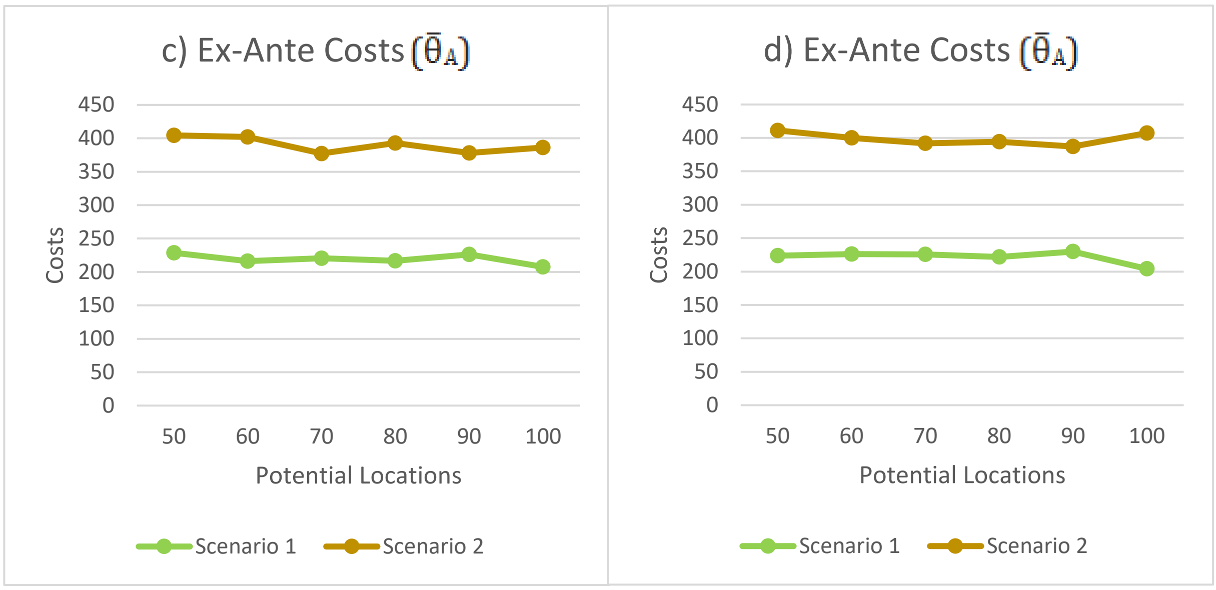


Figure 6-37 Large GA: Ex-Ante Costs

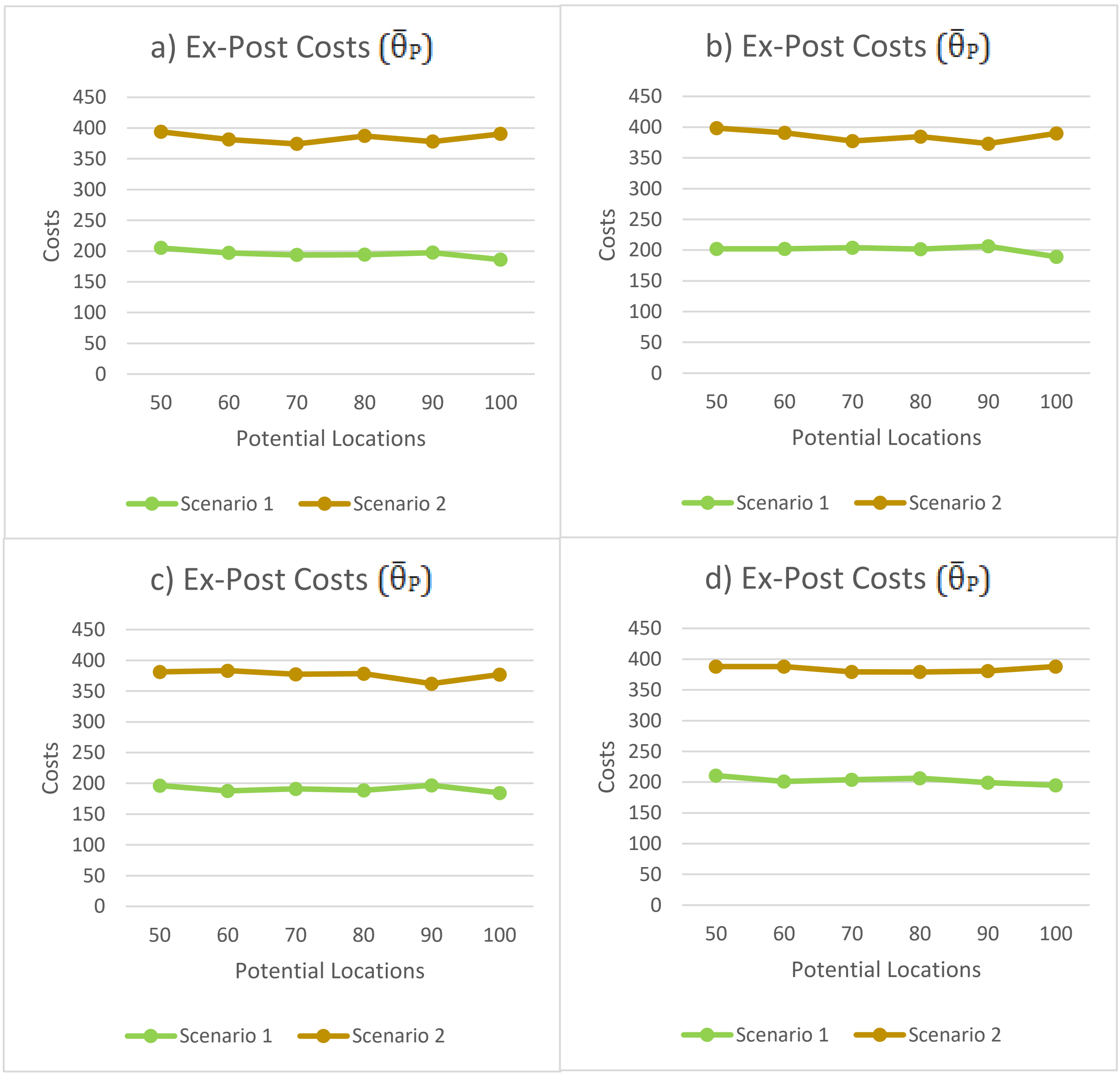


Figure 6-38 Large GA: Ex-Post Costs

Since the aim of the objective function is to maximize fitness values, and the performance differences of the four combinations a)-d) are negligible, the combination with the highest fitness values (two-point crossover and roulette wheel selection) is the best choice for this particular problem. Therefore, this combination was applied to the small problem size in subsection 6.3.2 “Genetic Algorithm”.

6.4.3. Comparison to Gutjahr (2020)

As mentioned earlier, Gutjahr (2020) also addresses inequity-averse optimization under uncertainty and compares experimental results of the ex-ante and ex-post approaches. It therefore seems interesting to discuss the observations of both works.

For ex-ante, Table 6-26 shows by how much the ex-ante fitness of the solution with the best ex-ante fitness ($\bar{\lambda}_A$) differs from the ex-ante fitness of the solution with the best ex-post fitness ($\bar{\lambda}_P$). These percentages are calculated using formula (2). The same applies to the ex-post approach, but this time formula (3) is used. In addition, the data provided in Gutjahr (2020) is subtracted by 1 and multiplied by 100 to allow for comparison.

		Potential Locations						Gutjahr
		50	60	70	80	90	100	(2020)
Ex-Ante	Min	-24.89	-72.28	-41.92	-37.58	-32.79	-41.72	0.00
	Max	43.24	45.32	44.37	47.49	46.29	54.06	28.42
	Average	9.08	9.76	9.77	9.42	11.40	9.92	9.04
Ex-Post	Min	-10.55	-19.89	-19.29	-30.46	-9.20	-36.19	0.72
	Max	34.80	38.17	33.61	33.45	31.91	29.05	16.73
	Average	7.87	7.59	7.97	8.21	8.09	5.78	8.89

Table 6-26 Large GA: Comparison of Percentage Differences

While the smallest percentage (Min) in Gutjahr’s paper is 0%, the smallest percentage of the large problem size (two-point crossover and roulette wheel selection) is even -72.28%. This can be easily explained as Gutjahr uses an approach (complete enumeration in combination with CPLEX) that delivers an optimal solution, whereas the large problem size is solved by a heuristic (genetic algorithm). For example, -72.28% means that λ_P delivers a much better solution than λ_A for this specific seed, which means that the GA got stuck in a local optimum when λ_A was calculated.
The largest percentage (Max) is 54.06%, which shows that λ_P is a lot worse than λ_A for this specific seed. Although the Min and Max percentages of the ex-post comparison are (much) smaller than those of the ex-ante comparison for each column, these values are nowhere near the values reported in Gutjahr. However, the average percentage of each column is quite close

to his work and the arithmetic mean of all average percentages (9.89% for ex-ante and 7.59% for ex-post) differs only slightly. Interestingly, the range (Max-Min) of ex-ante is much larger than the range of ex-post, implying that ex-post delivers less volatile results than ex-ante.

	Potential Locations						Average	Gutjahr (2020)
	50	60	70	80	90	100		
Ex-Ante	52	46	47	46	45	49	47.50	12
Ex-Post	42	43	43	47	49	45	44.83	12
Identical	6	11	10	7	6	6	7.67	0

Table 6-27 Large GA: Number of Occurrences of Smaller Percentage Differences

Table 6-27 shows how often the percentage differences of the ex-ante solutions (λ_A and λ_P) are smaller than the percentage differences of the ex-post solutions (ψ_P and ψ_A). When 50 potential locations are available, the number of seeds for which the ex-ante difference is smaller than the ex-post difference is considerable. For all other columns, the numbers are roughly the same. The averages show that the ex-ante difference is smaller than the ex-post difference in only 2.67% of the 100 seeds. This means that the ex-ante fitness of the solution with the best ex-ante fitness (λ_A) sees the ex-ante fitness of the solution with the best ex-post fitness (λ_P) slightly more favorable than the ex-post fitness of the solution with the best ex-post fitness (ψ_P) sees the ex-post fitness of the solution with the best ex-ante fitness (ψ_A).

To compare the average results and Gutjahr's (2020) results of the previous table, both columns are converted to percentages. For Gutjahr's results, the percentages are 50% and 50%, and for the average results the percentages (omitting identical results) are 51.44% and 48.56% for ex-ante and ex-post respectively. Thus, both evaluation approaches deviate from Gutjahr's results by only 1.44%.

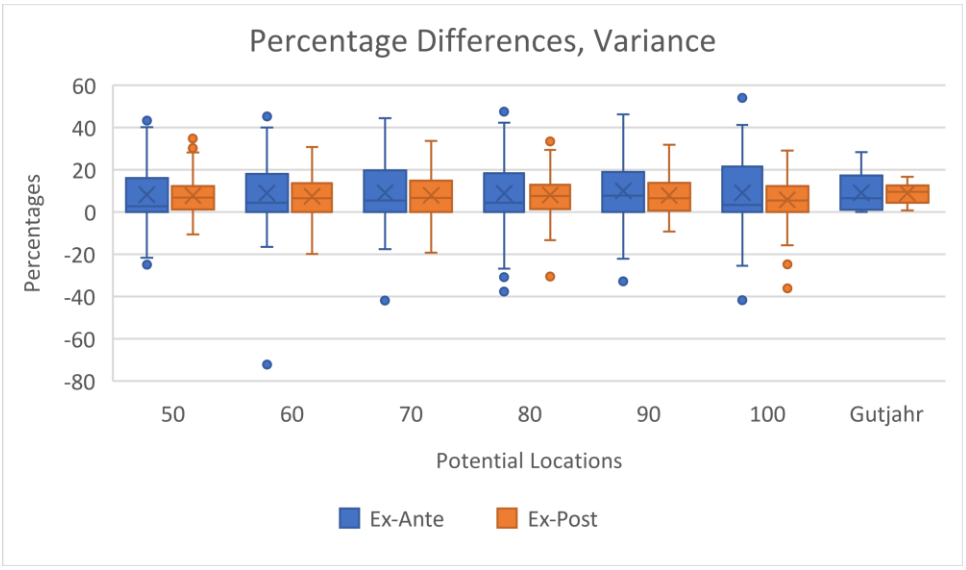


Figure 6-39 Large GA: Variance of Percentage Cost Differences

Figure 6-39 explains why ex-ante is favored although its arithmetic mean (9.89%) is worse than the arithmetic mean of the ex-post solutions (7.59%). While the range (Max-Min) of ex-ante is clearly larger than the range of ex-post, the ex-ante median is usually lower than the ex-post median, meaning that generally more than 50% of the ex-ante solutions have a lower percentage difference than the ex-post solutions. The data provided in Gutjahr (2020) shows the same pattern, although it is less volatile and does not have any negative percentages since it is solved to optimality.

6.5. Performance CE vs. GA

The aim of this section is to first compare the fitness values of the CE and the GA of the small problem size and then give approximate run times for different numbers of permanent shelters for both algorithms.

6.5.1. Visual Comparison

For two settings (increasing number of potential locations and increasing number of temporary locations), at first the fitness values of the arithmetic mean of 100 seeds are illustrated separately for ex-ante and ex-post. Then, the single seed that produces the largest difference between the CE and the GA fitness values is discussed.

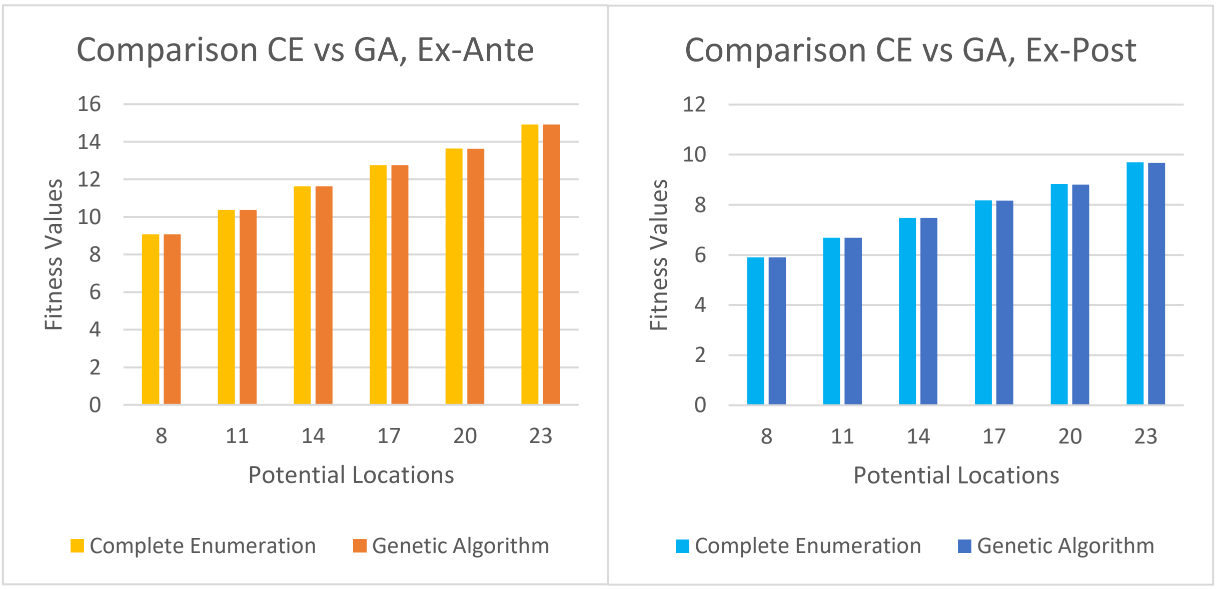


Figure 6-40 Comparison CE vs GA, Potential Locations, 100 Seeds

While the fitness values in Figure 6-40 (100 seeds) of both diagrams look identical and the values of the GA are indeed only 0.12% worse for ex-ante and at most 0.19% worse for ex-post, the fitness values of Figure 6-41 (single seed) are even 9.3% (for 20 potential locations) worse for ex-ante and 10.93% (for 23 potential locations) worse for ex-post. This outcome shows that the GA mostly found the best solution for various numbers of potential locations, which is not surprising considering the very short length of the chromosomes.

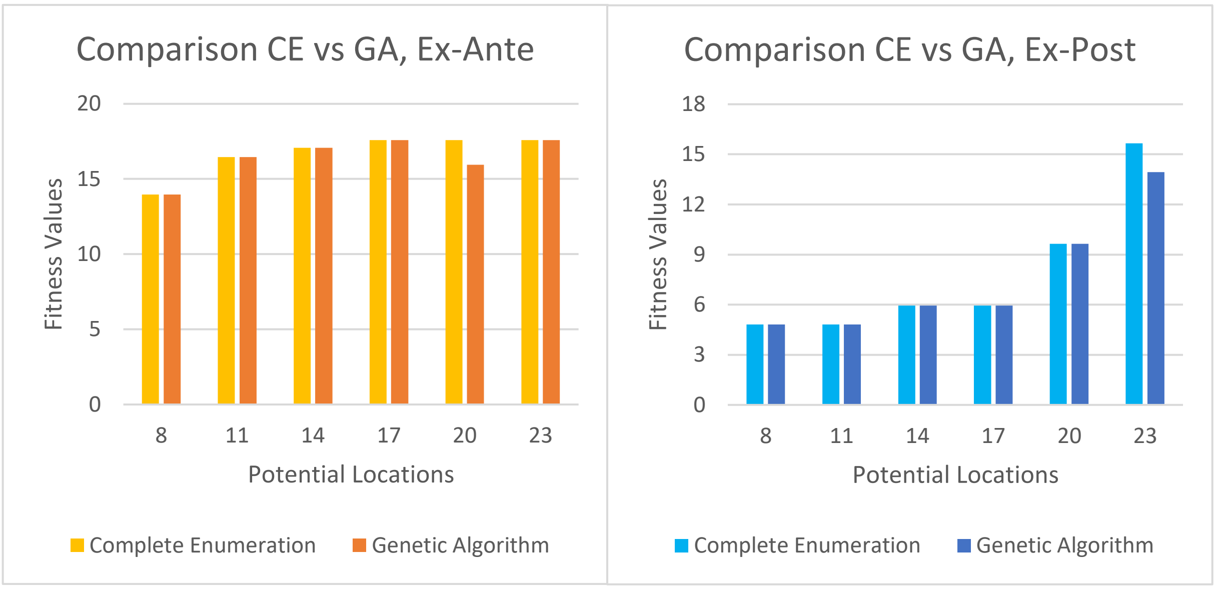


Figure 6-41 Comparison CE vs GA, Potential Locations, Single Seed

Similarly, for the temporary shelters in Figure 6-42 (100 seeds) and Figure 6-43 (single seed), the fitness values of the GA are only 0.03% (ex-ante) and 0.05% (ex-post) worse than the values of the CE when comparing the arithmetic means. For the single seed, the differences are also very small, 2.02% for ex-ante (for one temporary shelter) and 2.04% for ex-post (for one temporary shelter). The results show that the GA performs even better when the number of temporary shelters is changed compared to the number of potential locations.

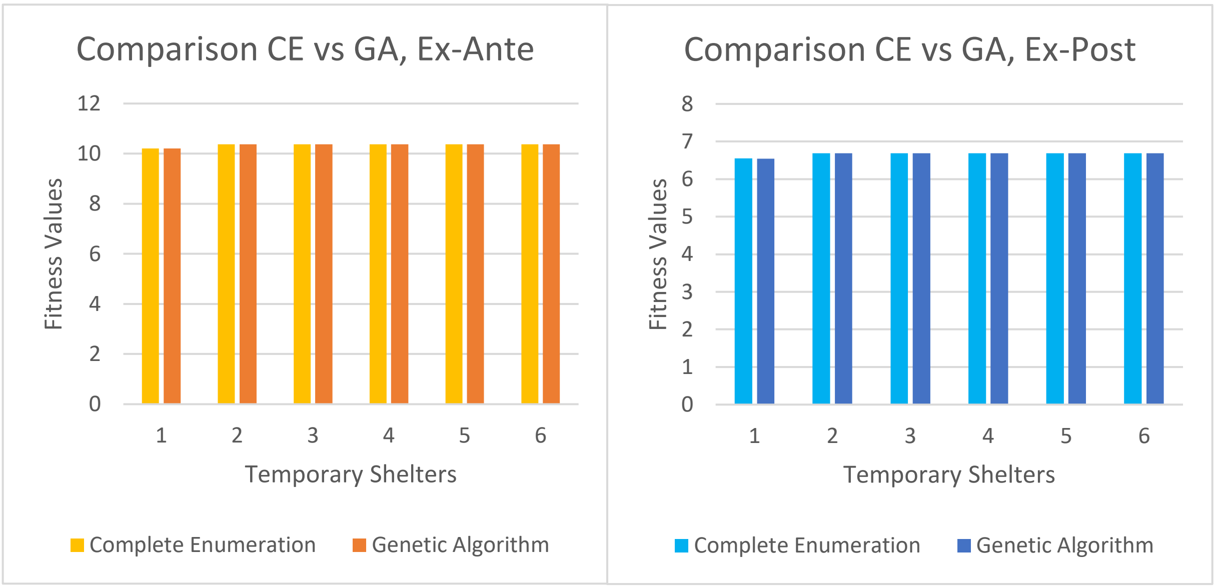


Figure 6-42 Comparison CE vs GA, Temporary Shelters, 100 Seeds

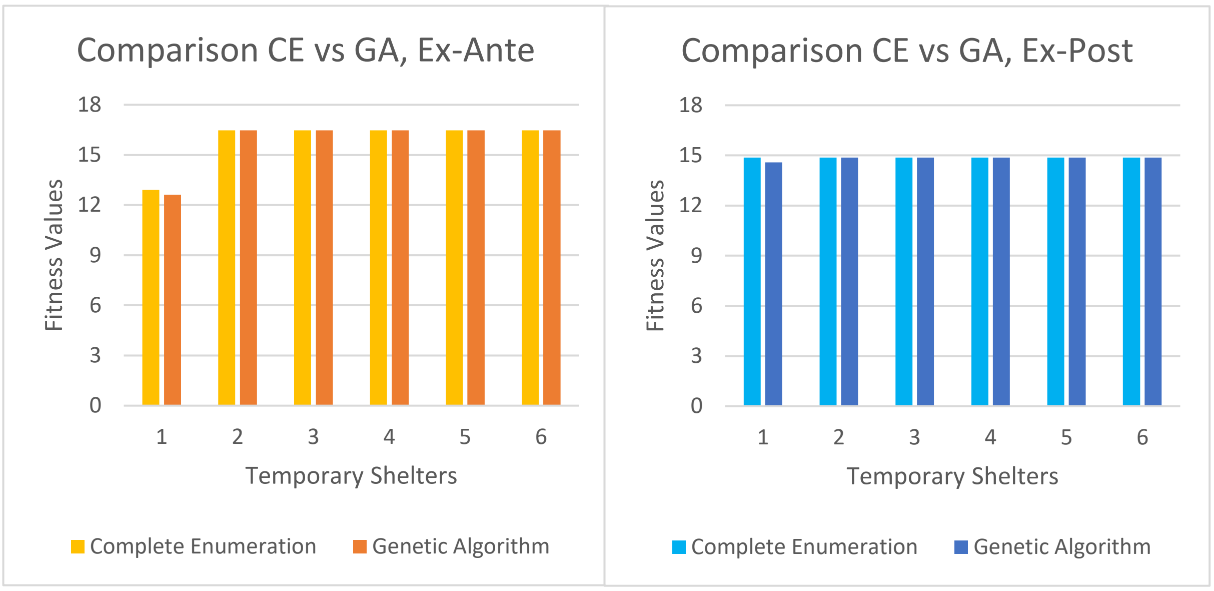


Figure 6-43 Comparison CE vs GA, Temporary Shelters, Single Seed

The differences of the number of population nodes and the number of permanent shelters are even smaller and therefore not visualized.

6.5.2. Time Measurement

The purpose of this subsection is to demonstrate the strength of the GA. Table 6-28 and Table 6-29 compare the performance of the CE and the GA for different numbers of permanent shelters. Based on the measured time of the large problem size where three permanent shelters are built and the number of possible combinations, all other values in both tables are calculated for the GA. This means that the time values of the GA at three permanent shelters are directly taken from the large problem size. For example, the GA takes 17.81 seconds to execute the code when 50 potential locations are available and three shelters are built. To obtain an estimated time for two shelters, 17.81 is divided by all possible combinations of three shelters (940) and then multiplied by all possible combinations of two permanent shelters (960), resulting in 18.19 seconds. Therefore, these values are not actual results but represent a reasonable estimate.

The time values of the CE are calculated similarly to the values of the GA, but instead of using the measured time of the large problem size, the measured time of the small problem size is used.

The ex-ante performances of the GA and the CE are given in Table 6-28. While the CE is significantly faster than the GA when only one permanent shelter is built, the differences already decrease when a second shelter is added: At 70 potential locations, the GA is faster than the CE for the first time.

Permanent Shelters	Algorithm	Time Unit	Potential Locations					
			50	60	70	80	90	100
1	GA	seconds	18.57	21.53	26.16	30.43	33.61	40.09
	CE	seconds	0.41	0.59	0.81	1.07	1.35	1.67
2	GA	seconds	18.19	21.16	25.78	30.05	33.23	39.68
	CE	seconds	10.05	17.54	28.06	42.12	60.23	82.90
3	GA	seconds	17.81	20.80	25.41	29.66	32.85	39.28
	CE	minutes	2.62	5.56	10.45	18.02	29.11	44.67
4	GA	seconds	17.43	20.43	25.03	29.28	32.48	38.87
	CE	hours	0.50	1.30	2.87	5.71	10.43	17.87
5	GA	seconds	17.05	20.07	24.65	28.89	32.10	38.47
	CE	days	0.19	0.59	1.56	3.57	7.39	14.15
6	GA	seconds	16.67	19.70	24.27	28.51	31.72	38.06
	CE	days	1.38	5.35	16.60	43.98	103.44	221.63

Table 6-28 Performance: Ex-Ante: CE vs. GA, per seed

It is noteworthy that the more shelters are built, the less time the GA needs to execute the code (see Figure 6-27 "GA: Permanent Shelters: Combinations"), while at the same time the runtime of the CE increases rapidly. For example, when only one permanent shelter is built and 100 potential locations are available, the GA needs 40.09 seconds and the CE 1.67, whereas the required time for six shelters decreases to 38.06 seconds for the GA and increases to 221.63 days for the CE. The time is given per seed, which means that in this case, calculating the mean of 100 seeds would take 60.72 years for the CE.

Permanent Shelters	Algorithm	Time Unit	Potential Locations					
			50	60	70	80	90	100
1	GA	seconds	20.77	26.46	30.97	35.49	40.92	42.89
	CE	seconds	0.37	0.54	0.74	0.97	1.23	1.52
2	GA	seconds	20.34	26.01	30.52	35.04	40.46	42.46
	CE	seconds	9.12	15.93	25.48	38.25	54.69	75.28
3	GA	seconds	19.92	25.56	30.07	34.59	40.00	42.03
	CE	minutes	2.38	5.04	9.49	16.36	26.43	40.57
4	GA	seconds	19.50	25.11	29.62	34.14	39.54	41.59
	CE	hours	0.46	1.18	2.61	5.18	9.47	16.23
5	GA	seconds	19.07	24.67	29.17	33.69	39.08	41.16
	CE	days	0.17	0.54	1.41	3.24	6.71	12.85
6	GA	seconds	18.65	24.22	28.73	33.25	38.62	40.73
	CE	days	1.26	4.86	15.07	39.94	93.93	201.25

Table 6-29 Performance: Ex-Post: CE vs. GA, per seed

For the CE, the ex-post evaluation (Table 6-29) takes less time to find a solution than the ex-ante evaluation, but the opposite is true for the GA because ex-post usually needs more generations to obtain the best solution than ex-ante. The time values given in the two tables are not very different for the GA, but for the CE, the differences increase with the size of the problem.

6.6. Case Study: Nepal

6.6.1. Settings

In this section, the GA is applied to the data of a real disaster that occurred in Nepal. Nepal was hit by a major earthquake on April 25, 2015. Over 22,000 people were injured and almost 9,000 died from collapsing buildings, landslides, and avalanches. Hundreds of aftershocks were followed by a second major earthquake on May 12 that destroyed over 600,000 houses and affected 8 million people (Reid, 2018).

The underlying data for this case study is taken from Fikar, Hirsch, and Nolz (2018), who simulate how word of mouth and damaged roads affect the distribution of relief goods in central Nepal. This real-world data consists of latitude and longitude of 27 major cities in central Nepal, and the number of households in each city. Figure 6-44 was created by pasting the latitude and longitude values into Microsoft 3D Maps for Excel, a data visualization tool (Nepal, 2020).

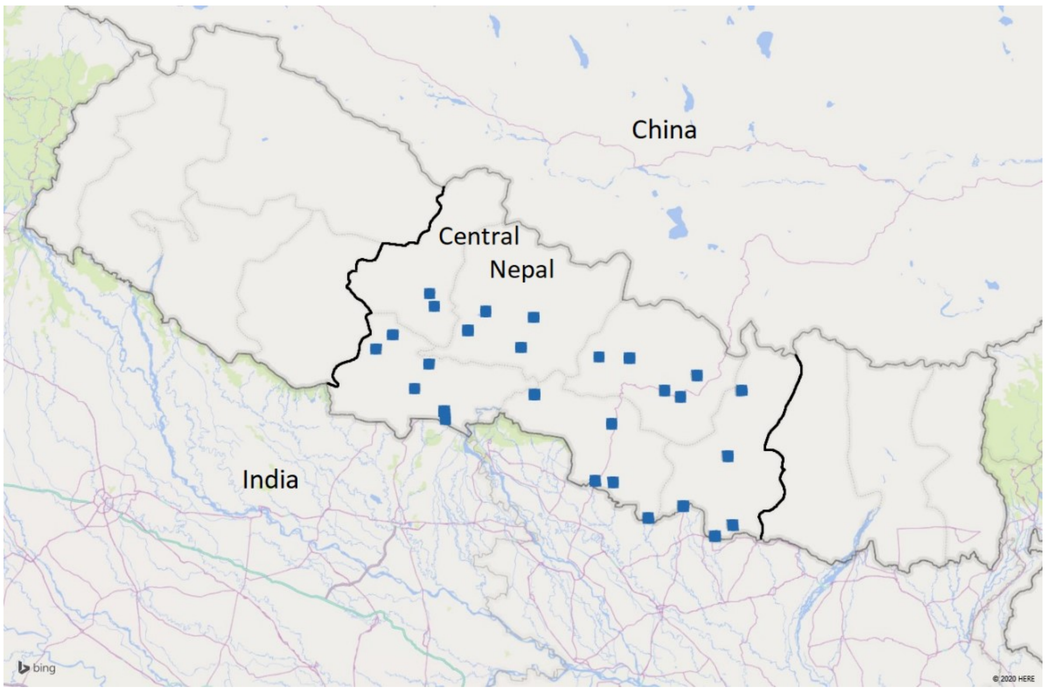


Figure 6-44 Nepal: Map of Central Nepal, 27 major cities, adapted from Nepal (2020)

Although the combination with two-point crossover and roulette wheel selection delivers the highest fitness values for the large problem size, the results of this section are calculated using two-point crossover and tournament selection because this combination delivers slightly higher fitness values. To demonstrate how a real disaster might be handled, only the results of a single seed (and not the arithmetic mean of 100 seeds) are discussed in this case study. Due to otherwise high computation times, only one temporary shelter is built, but the number of permanent shelters varies.

In the next subsection (6.6.2 "Applying Genetic Algorithm"), scenarios are generated based on the estimated PGA (peak ground acceleration) values and afterwards the GA is used to find a solution. Subsection (6.6.3 "Real Event") applies the GA to the data of the real earthquake. The results of these two subsections are then compared (6.6.4 "Comparison").

6.6.2. Applying Genetic Algorithm to Scenarios

6.6.2.1. Generating Scenarios

A seismic hazard map (Figure 6-45) which is based on estimated PGA values is taken from Rahman and Bai (2018) to create more realistic scenarios rather than simply placing them randomly on the previous map. PGA or peak ground acceleration is the maximum ground acceleration which is measured at a specific location during an earthquake (Sunaryo, 2017). The red star on the map indicates the epicenter of the April 25, 2015 earthquake, also called "Gorkha earthquake", named after the district where it occurred (Gautam and Rodrigues, 2018). The location of this earthquake is only included to show where it happened and does not affect the locations of the scenarios, which are selected based on seismic hazard potential.

Three scenarios (S 1, S 2, S 3) are placed on the map in a way that most of the orange and red (high seismic hazard potential) areas are included. The 27 population nodes (major cities) are taken from Figure 6-44. One difference to the previous subsections is that no new locations are added for potential locations. Each major city is now both, a population node and a potential location. Fikar et al. (2018) used this approach because shelters can be simply built within or near a city, and since this change does not affect the ex-ante or ex-post objective function, it is also used in this section.

After the locations of the scenarios have been chosen, an appropriate probability of occurrence is determined for each scenario. Since greenish colors have a lower PGA value than orange colors, it seems reasonable to base the probabilities on colors. This will assign a higher probability to scenarios with predominantly orange areas. The calculation is performed as follows: First, the color with the lowest PGA value of all scenarios is determined, which is greenish-yellow. This is now the base color of all scenarios. For scenario 3, this color is simply hidden under the other colors. The circle area of this color is equal to the entire circle of a scenario, as it is the base. Then the estimated circle area of the next higher color (including the areas of all even higher colors) is added and so on. This is done for all scenarios. The result of a scenario is divided by the sum of the results of all scenarios to get the percentage of that specific scenario. In this case study, the percentages of scenario 1, 2, and 3, are 25.285%, 31.525%, and 43.19% respectively.

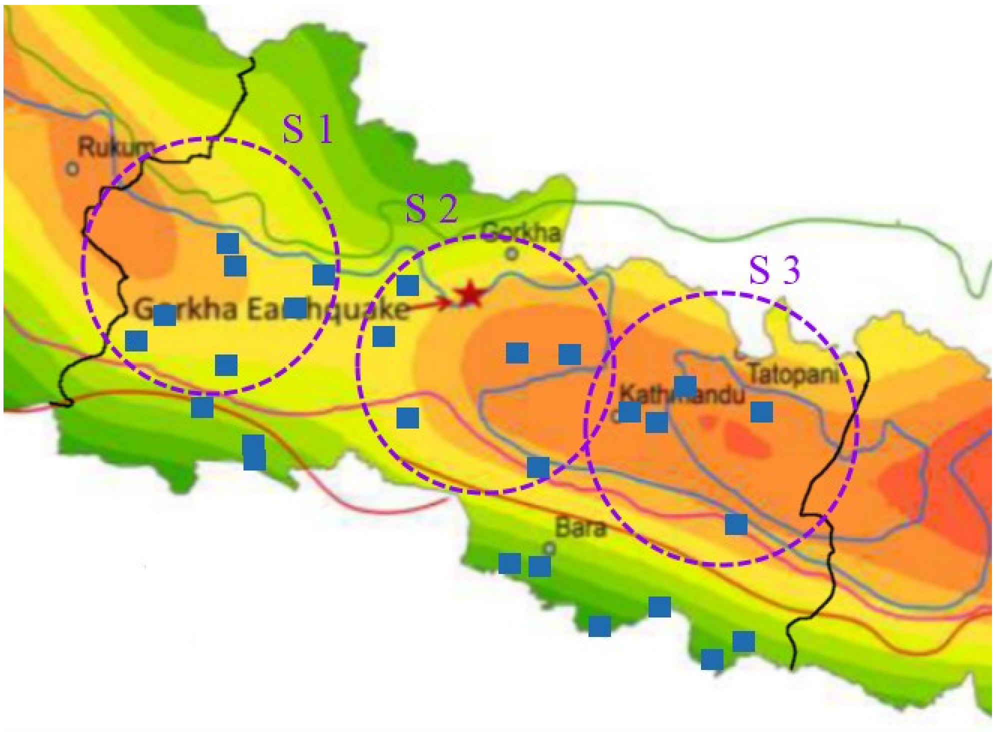


Figure 6-45 Nepal: Scenarios based on PGA Values, adapted from Rahman and Bai (2018)

As mentioned earlier, population nodes (cities) can be potential locations at the same time. Depending on the problem size, the number of potential locations could be huge without any constraints. Therefore, only the population nodes that lie within the circles (scenarios) can be potential locations. Which of these nodes are finally selected depends on a random number and the probability of occurrence of each scenario. Thus, a scenario with a high probability has a higher chance of having more potential locations than a scenario with a low probability.

6.6.2.2. Fitness Values and Performance

		Permanent Shelters		
		5	6	7
λ_A	Ex-Ante Fitness of Solution with Best Ex-Ante Fitness	943.08	1135.96	1293.64
λ_P	Ex-Ante Fitness of Solution with Best Ex-Post Fitness	551.31	656.43	906.11
Δ_A	Difference Ante in %	41.54	42.21	29.96
ψ_P	Ex-Post Fitness of Solution with Best Ex-Post Fitness	446.49	513.03	653.84
ψ_A	Ex-Post Fitness of Solution with Best Ex-Ante Fitness	368.50	480.97	562.85
Δ_P	Difference Post in %	17.47	6.25	13.92

Table 6-30 Nepal: Fitness Values

The fitness values given in Table 6-30 are visualized in Figure 6-46. For both approaches, the fitness increases when more permanent shelters are built, which is reasonable, as additional shelters can lead to a shorter travel distance for some of the affected population nodes.

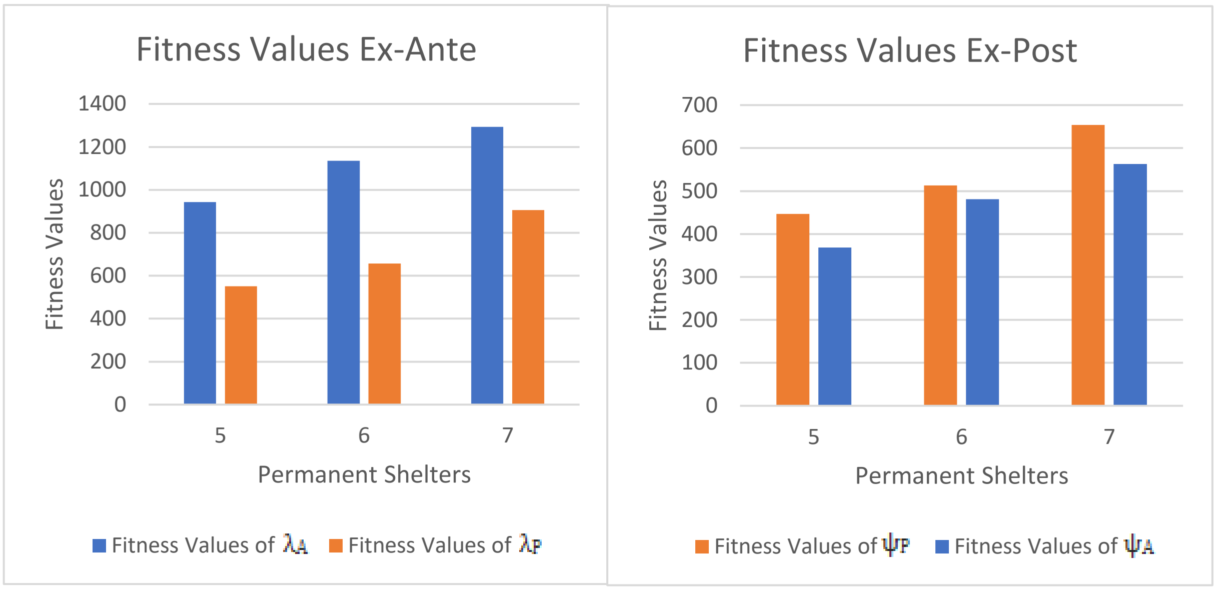


Figure 6-46 Nepal: Fitness Values

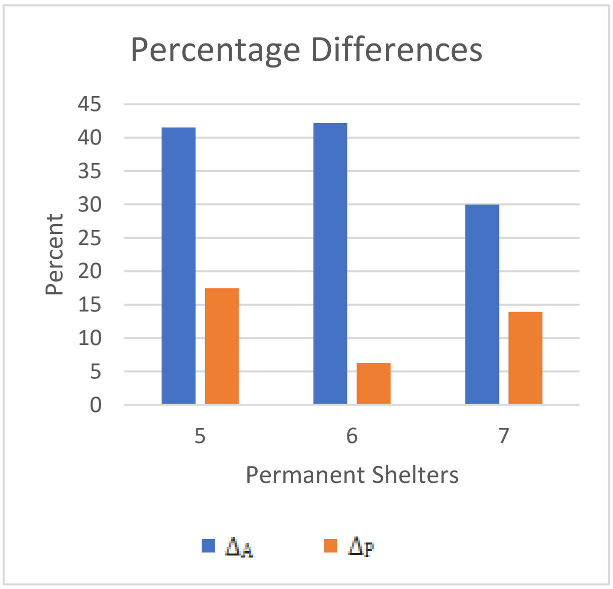


Figure 6-47 Nepal: Percentage Differences

The percentage differences in Figure 6-47 do not appear to have any particular pattern, however, Δ_A is always significantly larger than Δ_P. This means that the best ex-ante solution (ψ_A), when plugged into the ex-post objective function, delivers results that are much closer to the best ex-post solution than vice versa.

To observe how a different number of chromosomes affects the solution, the code is run again with 50 chromosomes instead of 20, with the result that both runs deliver identical fitness values. However, the number of generations differs in these two settings (Table 6-31). It takes up to 19 generations for ex-post to find the solution when 20 chromosomes are set, whereas only one generation is needed to when 50 chromosomes are set.

Chromosomes	20			50		
Permanent Shelters	5	6	7	5	6	7
Ex-Ante	1	1	1	1	1	0
Ex-Post	8	19	1	1	1	1

Table 6-31 Nepal: Number of Generations

Figure 6-48 shows the performance of 20 and 50 chromosomes. The difference of the two ex-post graphs is considerable. When 20 chromosomes are set, the number of generations needed to find the result varies greatly and the measured time at five and six shelters is therefore much higher than the time of ex-ante. 50 chromosomes lead to a much more stable graph, where the algorithms of both evaluation approaches take roughly the same time, though ex-ante performs now worse than before.

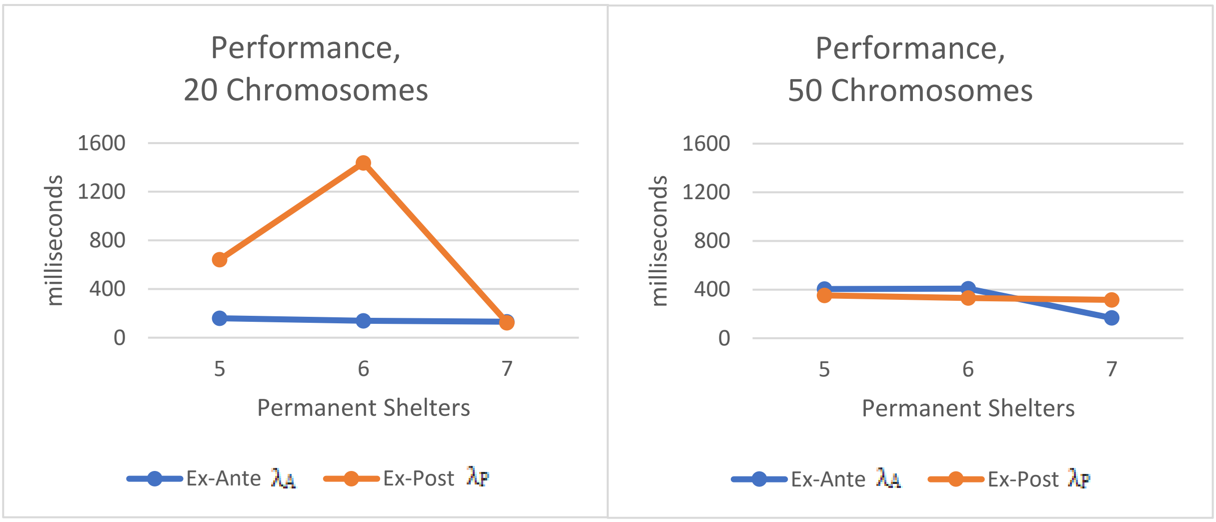


Figure 6-48 Nepal: Performance

6.6.2.3. Comparison of Costs

			Permanent Shelters 5	6	7
$\Delta_{1)}$	1) Ex-Ante (θ_A) vs. Ex-Post (θ_P)	Scenario 1	264.71	0.00	0.00
		Scenario 2	-78.10	-63.33	-41.97
		Scenario 3	-84.73	-74.37	-83.96
$\Delta_{2)}$	2) Ex-Ante (γ_A) vs. Utilitarian (γ_U)	Scenario 1	419.97	11.66	0.00
		Scenario 2	-44.41	62.21	0.00
		Scenario 3	547.52	269.26	1062.09
$\Delta_{3)}$	3) Ex-Post (γ_P) vs. Utilitarian (γ_U)	Scenario 1	0.00	0.00	2.72
		Scenario 2	395.58	784.54	471.45
		Scenario 3	[0.00]	[0.00]	[0.00]

Table 6-32 Nepal: Comparison of Costs (Ex-Ante, Ex-Post, and Utilitarian) in %

In Table 6-32, the costs of ex-ante in 1) are significantly lower than the costs of ex-post in all but one case. In scenario 3, ex-ante even saves up to 84.73% of the costs compared to ex-post. In 2), the ex-ante utilitarian costs are generally worse than the costs of the utilitarian objective function, and they are particularly bad in scenario 3, where they increase by up to 1062.09%.

The ex-post utilitarian costs in 3) are almost the same as the costs of the utilitarian objective function in scenario 1, much worse in scenario 2, and zero in scenario 3. It is possible that the costs are zero if a shelter is built at each affected population node. The reason why the numbers of scenario 3 are in brackets is that the costs of ex-post γ_P are 0 and in this case the resulting percentage of formula (8) $\Delta_{3)} = \frac{\gamma_P - \gamma_U}{\gamma_U}$ x 100 is always -100%, no matter what value γ_U has. These results cannot be compared with the results of 2) and 3) and are therefore in brackets.

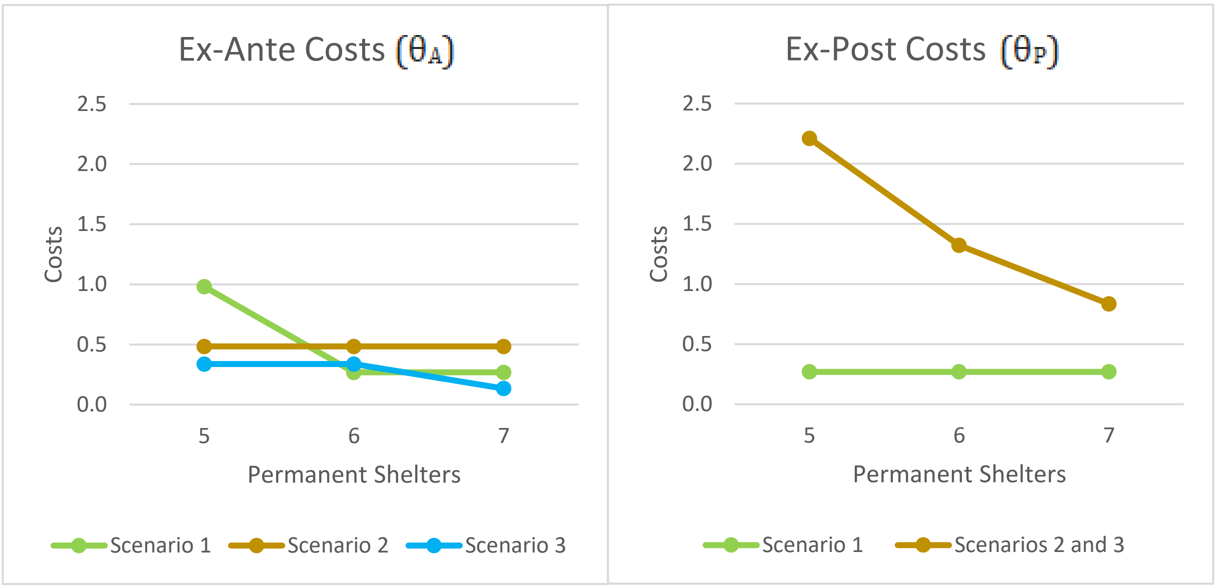


Figure 6-49 Nepal: Ex-Ante and Ex-Post Costs

Figure 6-49 displays the costs of all scenarios. The ex-ante cost of scenario 1 is at first higher than the ex-post cost but as the number of shelters increases, the costs decrease and remain the same. Scenarios 2 and 3 have identical ex-post costs that decrease as more shelters are added. However, even when seven permanent shelters are built, the costs of these two scenarios are still worse than the ex-ante costs.

6.6.3. Real Event

6.6.3.1. Affected Districts

After deciding on the best ex-ante and ex-post solution in the previous subsection, the best solution for the real disaster, a massive earthquake that has its epicenter in Gorkha district, will now be determined. In section 6.6.4 “Comparison of Scenarios and Real Event”, these solutions are compared.

The map in Figure 6-50 was created by a leading humanitarian organization called REACH Initiative, which provides “granular data, timely information and in-depth analysis from contexts of crisis, disaster and displacement” (REACH Initiative, n.d.). Their data is important for decision-makers when dealing with crisis-affected areas.

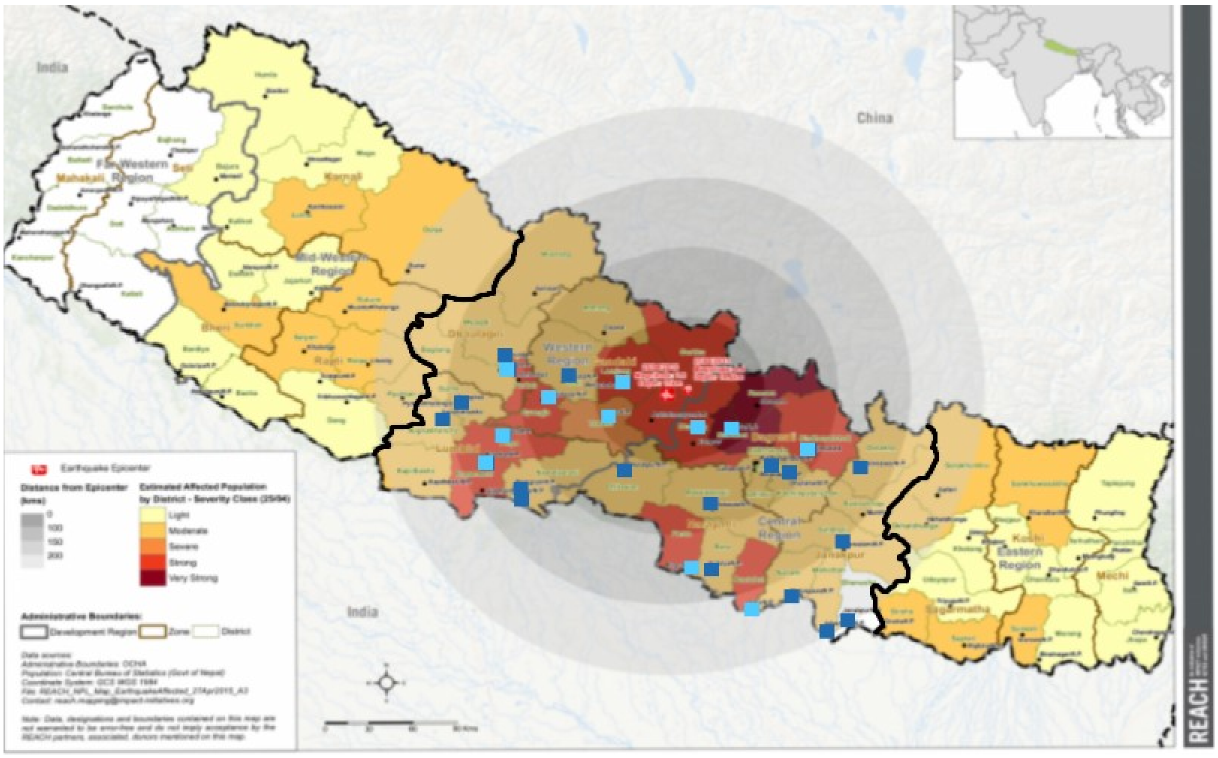


Figure 6-50 Nepal: Earthquake April 2015, adapted from REACH Initiative (2015)

It visualizes how badly the population of each district is affected by the earthquake. The colors of the districts indicate the severity: white – not affected, yellow – light, orange – moderate, dark orange – severe, red – strong, dark red – very strong. Almost all districts are impacted by this disaster, so this case study focuses on districts that are more heavily affected, which corresponds to the severe to very strong cases.

For a better overview, the color of the affected population nodes is changed to light blue on the map. Each population node (city) is a potential location at the same time.

6.6.3.2. Fitness Values and Performance

Table 6-33 and Figure 6-51 show that the fitness values of both evaluation approaches increase as more permanent shelters are added.

		Permanent Shelters		
		5	6	7
λ_A	Ex-Ante Fitness of Solution with Best Ex-Ante Fitness	272.27	307.59	346.01
λ_P	Ex-Ante Fitness of Solution with Best Ex-Post Fitness	272.27	307.59	407.82
Δ_A	Difference Ante in %	0.00	0.00	-17.86
ψ_P	Ex-Post Fitness of Solution with Best Ex-Post Fitness	272.27	307.59	407.82
ψ_A	Ex-Post Fitness of Solution with Best Ex-Ante Fitness	272.27	307.59	346.01
Δ_P	Difference Post in %	0.00	0.00	15.16

Table 6-33 Nepal: Real Event: Fitness Values

In the left diagram, the fitness value of ex-ante λ_A at 7 shelters is lower than the value of the best ex-post solution λ_P that is plugged into the ex-ante objective function. This means that the GA could not find the best ex-ante solution in this particular case.

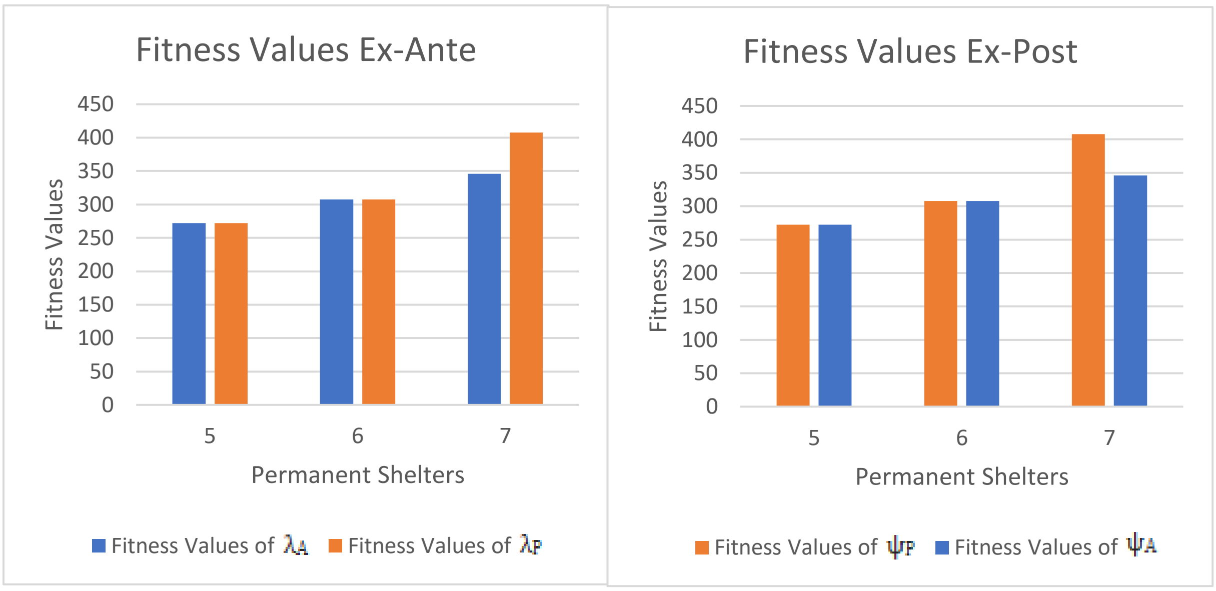


Figure 6-51 Nepal: Real Event: Fitness Values

The left diagram of Figure 6-52 visualizes the percentage differences of ex-ante and ex-post. When five or six shelters are built, both approaches are equally good. However, when seven shelters are built, ex-ante shows a negative percentage because the solution of ex-post λ_P is better than the solution of ex-ante λ_A.

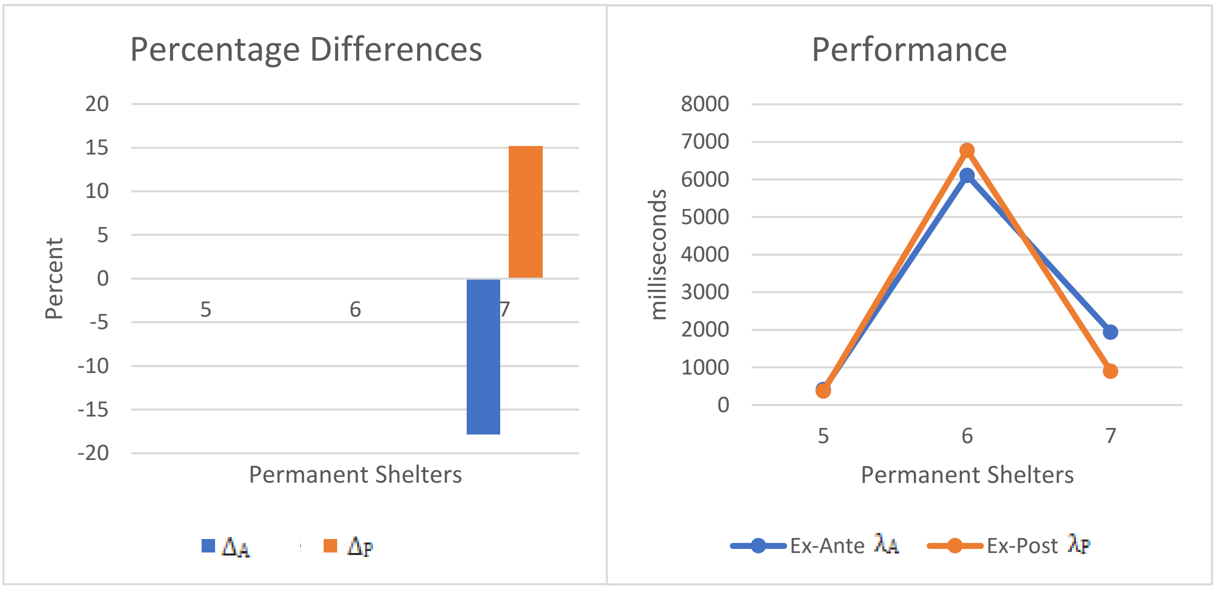


Figure 6-52 Nepal: Real Event: Percentage Differences and Performance

	Permanent Shelters		
	5	6	7
Ex-Ante	3	61	17
Ex-Post	3	70	8

Table 6-34 Nepal: Real Event: Number of Generations

Table 6-34 shows how many generations are needed for each number of shelters to obtain the fitness values given in Table 6-33. Both approaches need the most iterations when six shelters are built. The number of generations heavily influences the performance of the algorithms, which is visualized in Figure 6-52.

6.6.3.3. Comparison of Costs

Although both solutions, ex-ante and ex-post, deliver identical fitness values when six shelters are built, the costs of these solutions are different according to 1) in Table 6-35. This is the case when several different solutions have the same fitness value. Therefore, the combination of permanent and temporary shelters is different for ex-ante and ex-post, leads to the same fitness values, but has lower costs for ex-ante.

The ex-ante utilitarian costs γ_A are worse than the costs of the utilitarian objective function γ_U by up to 22.27%. The ex-post utilitarian costs γ_P in 3) are as good as the utilitarian function costs γ_U for five and seven shelters, but for six shelters the costs are even worse than the ex-ante utilitarian costs γ_A of 2).

		Permanent Shelters 5	6	7
$\Delta_{1)}$	1) Ex-Ante (θ_A) vs. Ex-Post (θ_P)	0.00	-12.34	43.51
$\Delta_{2)}$	2) Ex-Ante (γ_A) vs. Utilitarian (γ_U)	0.00	22.27	10.05
$\Delta_{3)}$	3) Ex-Post (γ_P) vs. Utilitarian (γ_U)	0.00	61.60	0.00

Table 6-35 Nepal: Real Event: Comparison of Costs (Ex-Ante, Ex-Post, and Utilitarian) in %

In Figure 6-53, the costs of both graphs are the same for five shelters, fall more steeply for ex-ante than for ex-post when another shelter is added, and fall by almost 50% for ex-post when the costs of seven shelters are compared to those of five shelters.

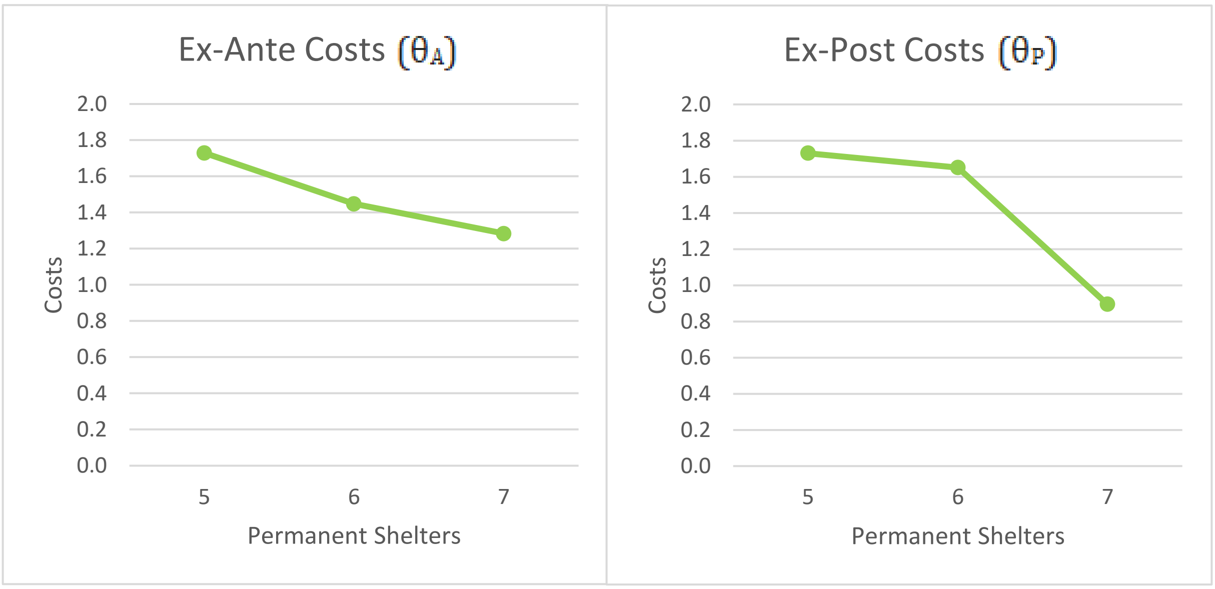


Figure 6-53 Nepal: Real Event: Ex-Ante and Ex-Post Costs

6.6.4. Comparison of Scenarios and Real Event

In this subsection, the solutions of subsections 6.6.2 "Applying Genetic Algorithm to Scenarios" (hereafter referred to as "scenario solution") and 6.6.3 "Real Event" are compared. For this purpose, the permanent and temporary shelters of the scenario solution are plugged into the code of the real event to start with the same settings (e.g. the same population nodes are affected). The code is executed separately for each scenario.

6.6.4.1. Fitness Values

The fitness values of the real event solution and the fitness values of the individual scenarios of the scenario solution are given in Table 6-36. Furthermore, the percentage differences of the fitness value of the real event and the fitness values of the individual scenarios are calculated for both evaluation approaches.

		Permanent Locations		
		5	6	7
λ_{REAL}	Real Event (ante)	272.27	307.59	346.01
λ_{S1}	Scenario 1 (ante)	140.77	140.77	140.77
λ_{S2}	Scenario 2 (ante)	104.30	140.77	140.77
λ_{S3}	Scenario 3 (ante)	92.48	140.77	140.77
$\Delta_{A,S1}$	Diff. Real & S1 (%)	48.30	54.23	59.32
$\Delta_{A,S2}$	Diff. Real & S2 (%)	61.69	54.23	59.32
$\Delta_{A,S3}$	Diff. Real & S3 (%)	66.03	54.23	59.32
ψ_{REAL}	Real Event (post)	272.27	307.59	407.82
ψ_{S1}	Scenario 1 (post)	112.74	118.59	119.50
ψ_{S2}	Scenario 2 (post)	112.74	118.59	140.77
ψ_{S3}	Scenario 3 (post)	87.31	118.59	114.34
$\Delta_{P,S1}$	Diff. Real & S1 (%)	58.59	61.45	70.70
$\Delta_{P,S2}$	Diff. Real & S2 (%)	58.59	61.45	65.48
$\Delta_{P,S3}$	Diff. Real & S3 (%)	67.93	61.45	71.96

Table 6-36 Nepal: Comparison: Fitness Values

These numbers are then visualized in Figure 6-54. In the left diagram, the fitness values of ex-ante are compared. Since different population nodes are affected in each scenario, it is no surprise that the fitness values of the scenarios are much worse than the fitness values of the real event. The fitness of scenarios 2 and 3 grows when another shelter is added but for scenario 1, the fitness remains the same.

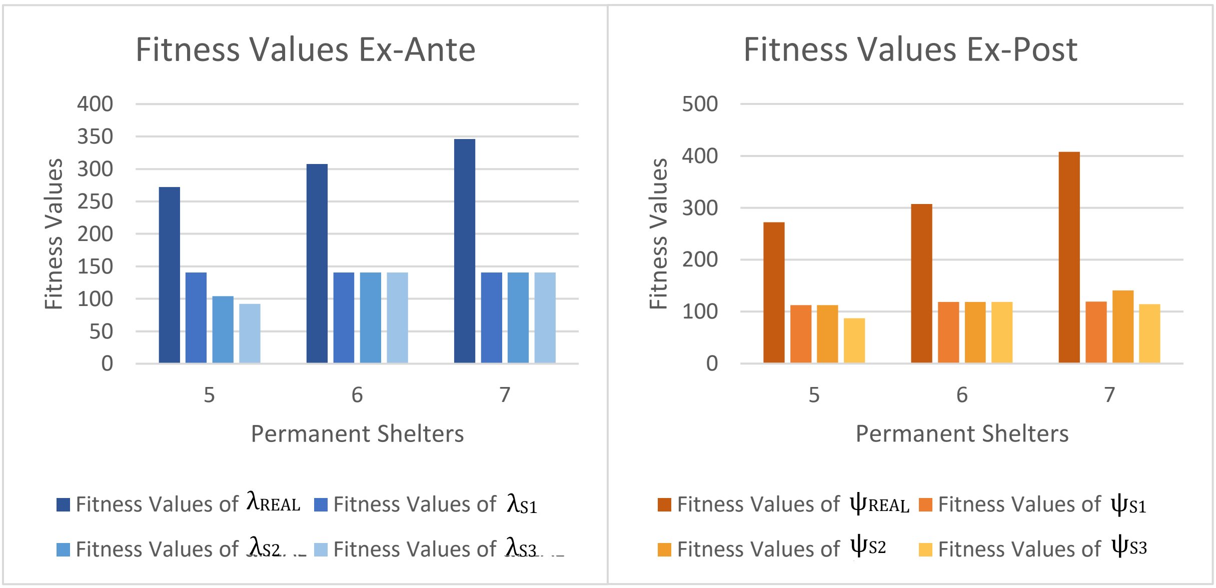


Figure 6-54 Nepal: Comparison: Fitness Values

Similarly, for ex-post, the fitness values of the scenario solutions are much worse than the fitness of the real event. The fitness of scenario 1 remains the same for all number of shelters. For scenario 2, the fitness increases when two shelters are added and for scenario 3, the fitness increases when one shelter is added.

Figure 6-55 shows the percentage differences of Table 6-36. To facilitate the comparison of ex-ante (blue colors) and ex-post (orange colors), the differences of the same scenario are displayed directly next to each other.

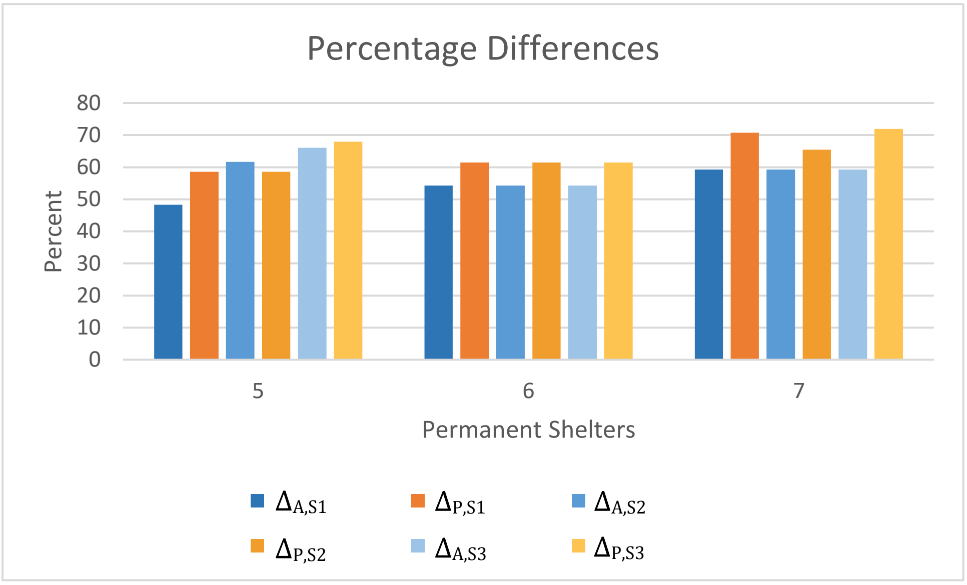


Figure 6-55 Nepal: Comparison: Percentage Differences

For five shelters, ex-post has a higher difference than ex-ante for scenarios 1 and 3, and a slightly lower difference for scenario 2. This part of the diagram seems to have an upwards trend. For six shelters, the differences remain the same for both approaches, but are a bit higher for ex-post. The ex-ante differences remain the same when seven shelters are built and vary when ex-post is applied. In general, ex-ante provides better and less fluctuating solutions than ex-post when compared to the real event solution.

The time performances of the real event solution and the scenario solution cannot be compared due to different settings, e.g. the shelters of the scenario solution are simply plugged into the real event algorithm, whereas for the real event solution, all population nodes are potential locations, which takes much more time to execute.

6.6.4.2. Comparison of Costs

In Table 6-37, the costs/distances of the ex-ante scenario solution are at least 104.14% worse than the costs of the ex-ante real event solution. The overall smallest percentage differences of 1) are obtained when six shelters are built, and the differences are worst for seven shelters. In 2), the percentages are usually much higher than in 1), implying that ex-post causes higher costs than ex-ante. Again, the differences are much smaller for six shelters than for seven shelters.

When the utilitarian costs of the ex-ante scenario solution are compared to the utilitarian objective function of the real event, the differences $\Delta_{3)}$ are a lot worse than the differences of the real event solution ($\Delta_{2)}$) given in Table 6-35. Scenario 1 delivers the solutions with the lowest costs.

4) is quite similar to 3): the given percentages are much worse than those of $\Delta_{3)}$ in Table 6-35. The ex-post utilitarian differences in 4) are usually worse than the ex-ante utilitarian differences in 3). Scenario 2 delivers the lowest costs for all number of shelters in 4).

			Permanent Shelters		
			5	6	7
$\Delta_{1)}$	1) Scenario Solution (ex-ante) vs. Real Event (ex-ante)	Scenario 1	104.14	105.82	188.56
		Scenario 2	188.89	118.59	186.96
		Scenario 3	197.93	118.59	227.62
$\Delta_{2)}$	2) Scenario Solution (ex-post) vs. Real Event (ex-post)	Scenario 1	209.09	153.63	392.66
		Scenario 2	167.18	124.96	289.04
		Scenario 3	276.53	194.45	486.30
$\Delta_{3)}$	3) Utilitarian Scenario Solution (ex-ante) vs. Utilitarian (Real Event)	Scenario 1	194.17	179.42	216.79
		Scenario 2	379.37	221.99	223.01
		Scenario 3	406.70	221.99	277.08
$\Delta_{4)}$	4) Utilitarian of Scenario Solution (ex-post) vs. Utilitarian (Real Event)	Scenario 1	321.90	288.81	306.69
		Scenario 2	275.13	234.05	210.88
		Scenario 3	431.04	357.46	392.59

Table 6-37 Nepal: Comparison: Comparison of Costs (Ex-Ante, Ex-Post, and Utilitarian) in %

Figure 6-56 displays the costs of both evaluation approaches. The scenario solution costs are a lot worse than the costs of the real event solution. For all scenarios, costs are lowest when six shelters are built. Scenario 1 has the best ex-ante costs and the worst ex-post costs of all scenarios.

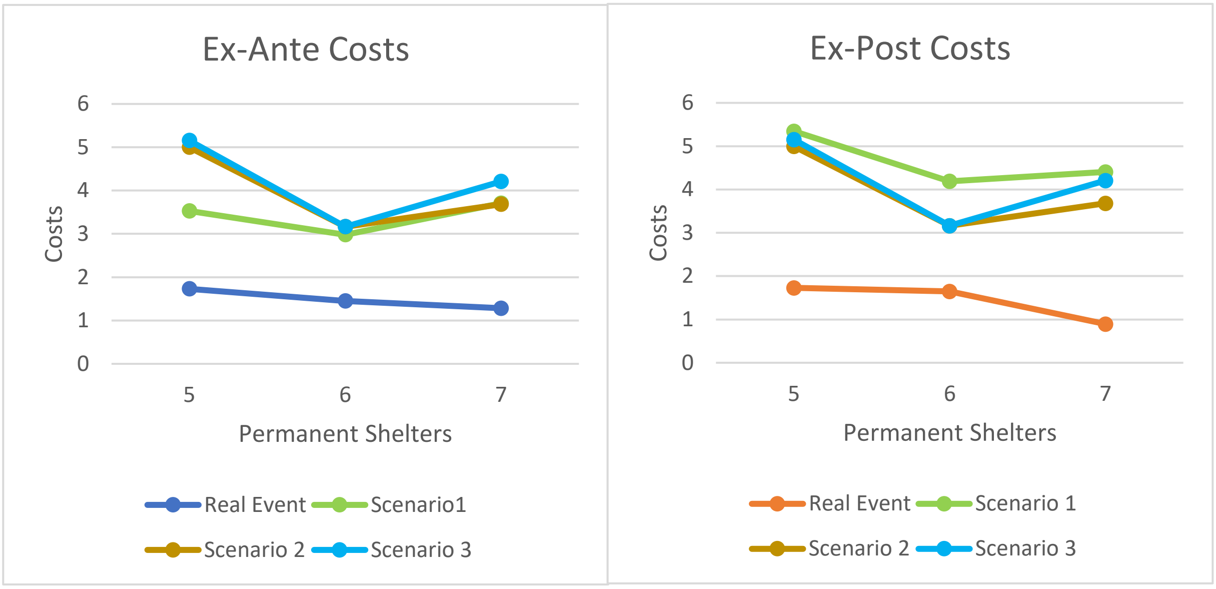


Figure 6-56 Nepal: Comparison: Ex-Ante and Ex-Post Costs

6.6.5. Choose Solution

Now that the real event has occurred and its solution has been compared to the scenario solution in the previous subsection, it is possible to decide which scenario should be chosen to maximize fairness. Depending on whether ex-ante or ex-post is applied, this solution may vary.

	Permanent Shelters		
	5	6	7
Ex-Ante	Scenario 1	Indifferent	Indifferent
Ex-Post	Scenario 1 or 2	Indifferent	Scenario 2

Table 6-38 Nepal: Solution: Decision-Making based on Fitness Values

Table 6-38 shows which scenario should be chosen to maximize the fitness value (see Table 6-36) for each number of permanent shelters. For example, when ex-ante is used and five permanent shelters are built, the temporary shelter of scenario 1 delivers the highest fitness value. Although the earthquake's epicenter is within scenario 2, this scenario is not necessarily the best choice because population nodes located outside of this scenario are also affected.

	Permanent Shelters		
	5	6	7
Ex-Ante	Scenario 1	Scenario 1	Scenario 1
Ex-Post	Scenario 2	Scenario 2	Scenario 2

Table 6-39 Nepal: Solution: Decision-Making based on Utilitarian Costs

If only the utilitarian costs (see Table 6-37) are considered, the scenarios should be chosen according to Table 6-39. Here it can be seen that for ex-ante always scenario 1 and for ex-post always scenario 2 offers the solution with the lowest costs.

6.7. Budget

6.7.1. Settings

In most cases, decision-makers do not only need to select the best locations for a given number of shelters, but also stay within a certain budget. To observe how this budget affects the solution, this section compares the outcome of two combinations. The genetic algorithm is applied to this problem since it is quite big, and the arithmetic mean of 100 seeds is calculated. As the roulette wheel selection combined with the two-point crossover delivers the best results for the large problem size in section 6.4, it is also used for this problem. In the following example, permanent shelters cost 1 monetary unit and temporary shelters cost 2 monetary units, since temporary shelters are needed immediately and must be built as quickly as possible.

The first combination consists of 4 permanent and 1 temporary shelter (4/1), and the second combination has 2 permanent and 2 temporary shelters (2/2). These combinations are chosen so that the number of monetary units (or budget) is the same for both combinations. For 4/1 this means 4 permanent shelters * 1 monetary unit + 1 temporary shelter * 2 monetary units equals 6 monetary units. The result of combination 2/2 is the same (2*1 + 2*2 = 6). Therefore, the budget of both combinations consists of 6 monetary units.

Population Nodes	Potential Locations	Scenarios	Chromosomes	Generations
100	20	2	6	50

Table 6-40 Budget: Settings for 4/1 and 2/2

The settings of the two combinations are given in Table 6-40. A very low number of chromosomes is chosen because a higher number usually delivers the best solution after zero (starting population) or one generation, which would not allow a meaningful comparison of the number of generations of ex-ante and ex-post.

6.7.2. Fitness Values and Performance

Table 6-41 and Figure 6-57 show that all fitness values ($\bar{\lambda}_A$, $\bar{\lambda}_P$, $\bar{\psi}_P$, and $\bar{\psi}_A$) decrease when combination 2/2 is used. This is a reasonable outcome because now only four shelters are built compared to the five shelters of combination 4/1, which can lead to higher costs.

		Combinations	
		4/1	2/2
$\bar{\lambda}_A$	Ex-Ante Fitness of Solution with Best Ex-Ante Fitness	1.00	0.97
$\bar{\lambda}_P$	Ex-Ante Fitness of Solution with Best Ex-Post Fitness	0.97	0.94
$\bar{\Delta}_A$	Difference Ante in %	3.04	3.20
$\bar{\psi}_P$	Ex-Post Fitness of Solution with Best Ex-Post Fitness	0.92	0.89
$\bar{\psi}_A$	Ex-Post Fitness of Solution with Best Ex-Ante Fitness	0.88	0.86
$\bar{\Delta}_P$	Difference Post in %	4.04	3.12

Table 6-41 Budget: Fitness Values

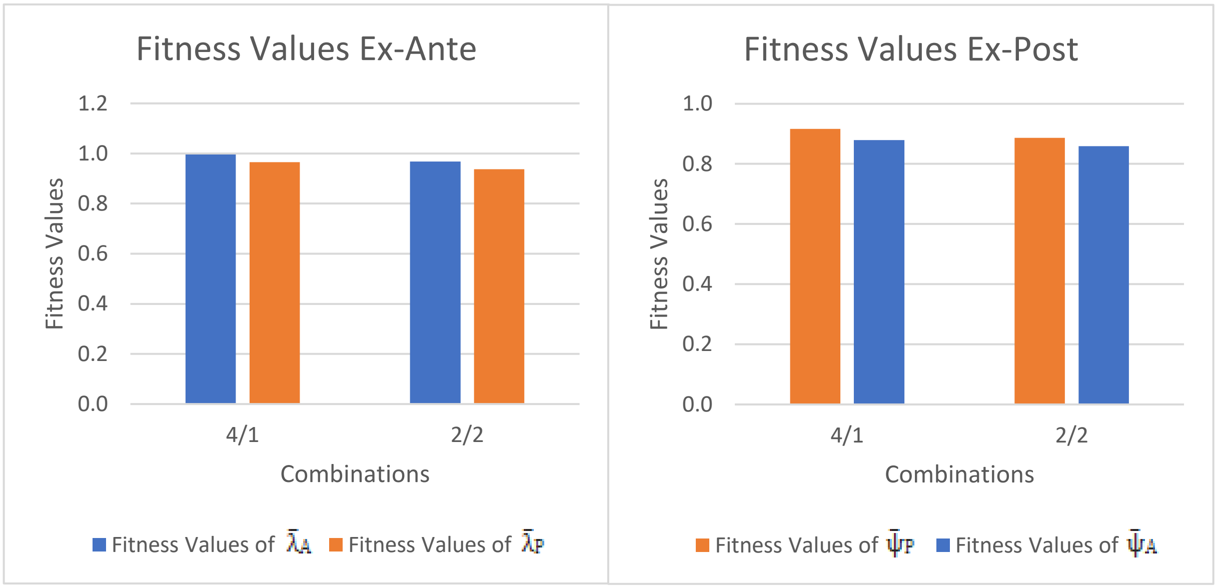


Figure 6-57 Budget: Fitness Values

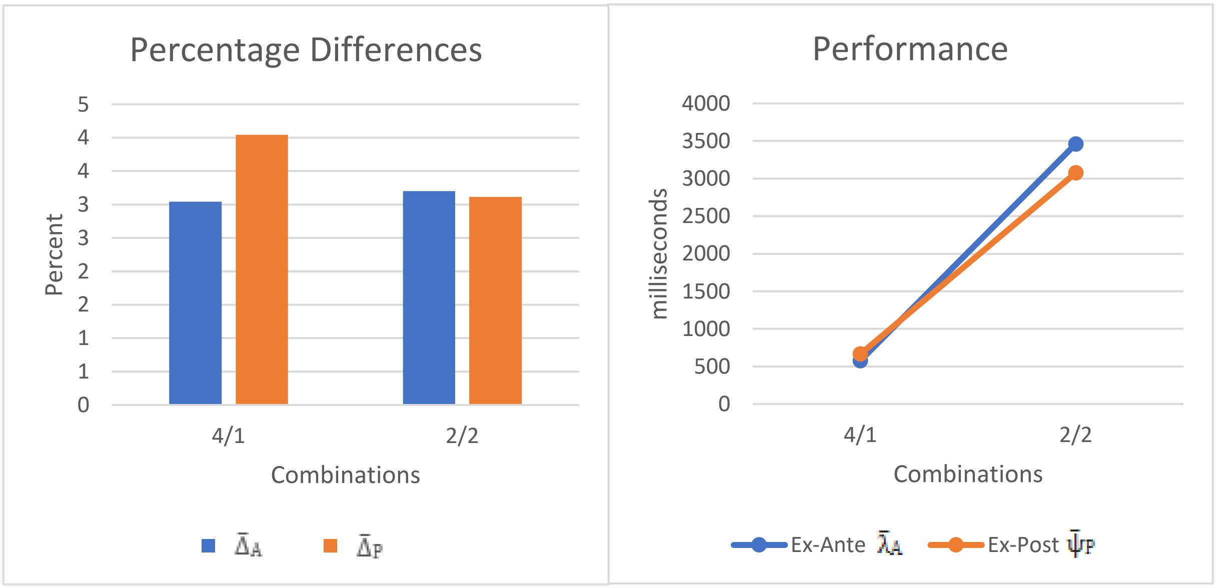


Figure 6-58 Budget: Percentage Differences and Performance

The percentage differences in Figure 6-58 do not seem to follow any pattern and are roughly the same. Executing the ex-ante code of combination 2/2 takes almost six times longer than executing the ex-ante code of combination 4/1.

In Figure 6-59, the box-and-whisker plot shows the variance in the number of generations needed to obtain the best solution for each seed. Comparing ex-ante to ex-post in combination 4/1, it appears that ex-post leads to a higher variance with fewer but higher outliers. Ex-ante of combination 2/2 has a negligible higher variance than ex-post. Interestingly, the median of all boxes is 1.

To give an overview on what the frequency of occurrences looks like, a histogram of ex-ante, combination 4/1, is added. The histograms of ex-post and combination 2/2 look very similar and are therefore omitted. About 74% of the seeds find the best ex-ante solution within 0 to 3 generations and only 1% of the seeds need 18 or more generations.

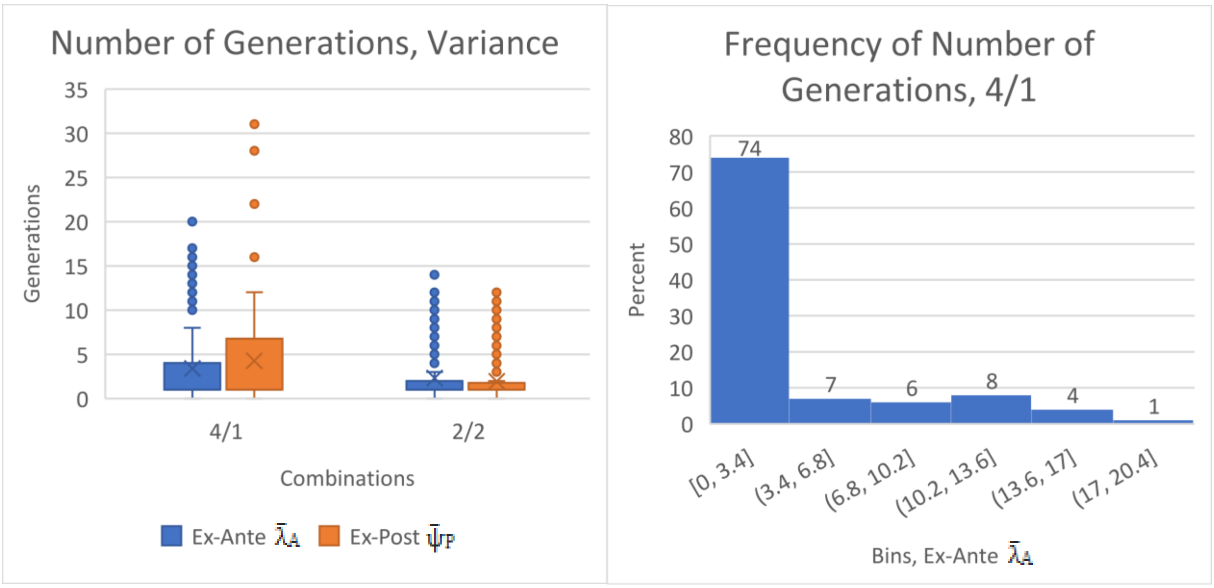


Figure 6-59 Budget: Variance in Needed Generations, Frequency of Occurrences

6.7.3. Comparison of Costs

According to the percentage differences given in Table 6-42, ex-ante $\bar{\theta}_A$ performs slightly worse than ex-post $\bar{\theta}_P$ regardless of the combination. Compared to the best utilitarian costs $\bar{\gamma}_U$, the utilitarian costs of the best ex-post solution $\bar{\gamma}_P$ are smaller than the utilitarian costs of the best ex-ante solution $\bar{\gamma}_A$. In 2) and 3), the differences of combination 2/2 are larger than the differences of combination 4/1.

			Combinations 4/1	Combinations 2/2
$\bar{\Delta}_{1)}$	1) Ex-Ante ($\bar{\theta}_A$) vs. Ex-Post ($\bar{\theta}_P$)	Scenario 1	1.50	0.82
		Scenario 2	0.38	0.49
$\bar{\Delta}_{2)}$	2) Ex-Ante ($\bar{\gamma}_A$) vs. Utilitarian ($\bar{\gamma}_U$)	Scenario 1	5.79	7.92
		Scenario 2	4.43	7.45
$\bar{\Delta}_{3)}$	3) Ex-Post ($\bar{\gamma}_P$) vs. Utilitarian ($\bar{\gamma}_U$)	Scenario 1	4.05	6.53
		Scenario 2	3.80	6.59

Table 6-42 Budget: Comparison of Costs (Ex-Ante, Ex-Post, and Utilitarian) in %

In Figure 6-60, the ex-ante costs of scenario 1 are slightly higher than the ex-post costs. The costs of both approaches and scenarios increase when combination 2/2 is built, which is reasonable because fewer shelters often lead to longer distances between shelters and affected population nodes.

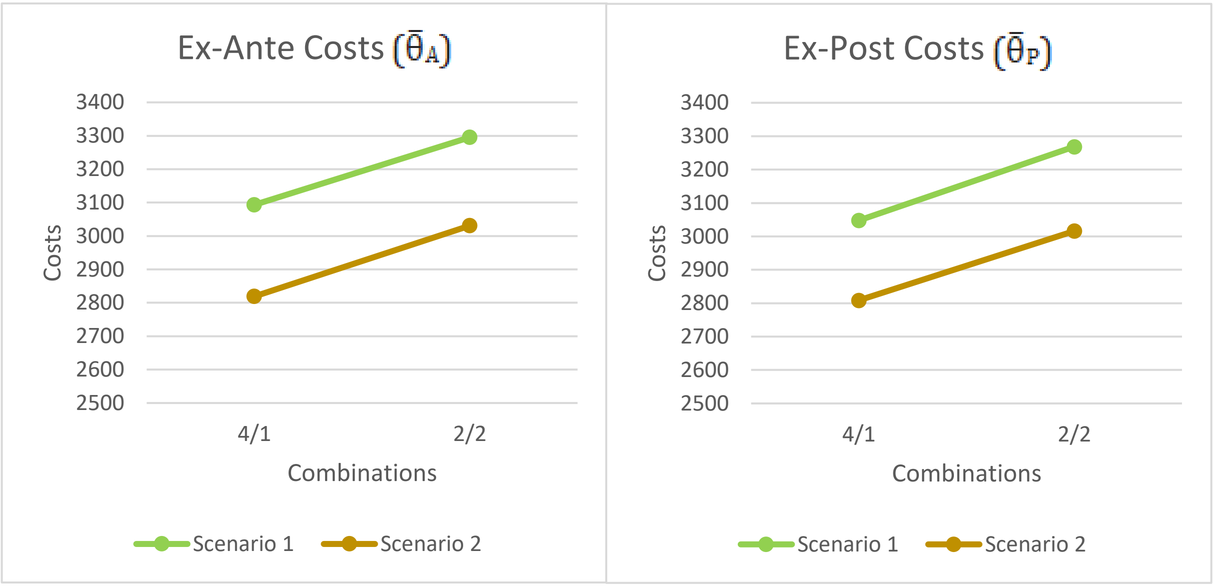


Figure 6-60 Budget: Ex-Ante and Ex-Post Costs

In summary, for the discussed settings and combinations, 4/1 achieved better fitness values than 2/2, as well as lower utilitarian costs.

6.7.4. Impact of Settings

The number of shelters (4/1 and 2/2) in the previous subsection may seem small but it is necessary due to high computation times. In this subsection it is shown how a larger number of potential locations, permanent shelters, temporary shelters, and chromosomes affects the results and runtime.

6.7.4.1. Large Number of Solutions

Table 6-43 shows the total solutions that are calculated for each shelter combination. As explained in subsection 6.3.2.3.1, the number of permanent combinations is equal to the number of chromosomes, which in this case is 20. The number of temporary combinations of each column is calculated using the binomial coefficient (formula (1)), where n is the remaining number of potential locations (= potential locations – permanent shelters) and k is the number of temporary shelters.

	Shelter Combinations				
	10/1	8/2	6/3	4/4	2/5
Permanent Combinations	20	20	20	20	20
Temporary Combinations	10	66	364	1820	8568
Total Solutions	200	1320	7280	36400	171360

Table 6-43 Impact of Settings: Number of Solutions, 20 Potential Locations, per Seed

For the first column (10/1 = 10 permanent and 1 temporary shelter) this means that each of the 20 permanent combinations has 10 temporary combinations, which leads to a total of 200 solutions that must be calculated per generation and per seed. Therefore, for shelter combination 2/5 and 100 seeds, 17.136.000 solutions are calculated per generation. As this calculation would take several days (see section 6.5 "Performance CE vs. GA"), the number of potential locations is reduced from 20 to 12 in Table 6-44.

6.7.4.2. Small number of Potential Locations

In Table 6-44 the same calculations are made as in Table 6-43 but this time with 12 potential locations. Shelter combination 2/5 now has only 5.040 solutions per generation and per seed. The fitness values of these settings are displayed in Figure 6-61.

	Shelter Combinations				
	10/1	8/2	6/3	4/4	2/5
Permanent Combinations	20	20	20	20	20
Temporary Combinations	2	6	20	70	252
Total Solutions	40	120	400	1400	5040

Table 6-44 Impact of Settings: Number of Solutions, 12 Potential Locations, per Seed

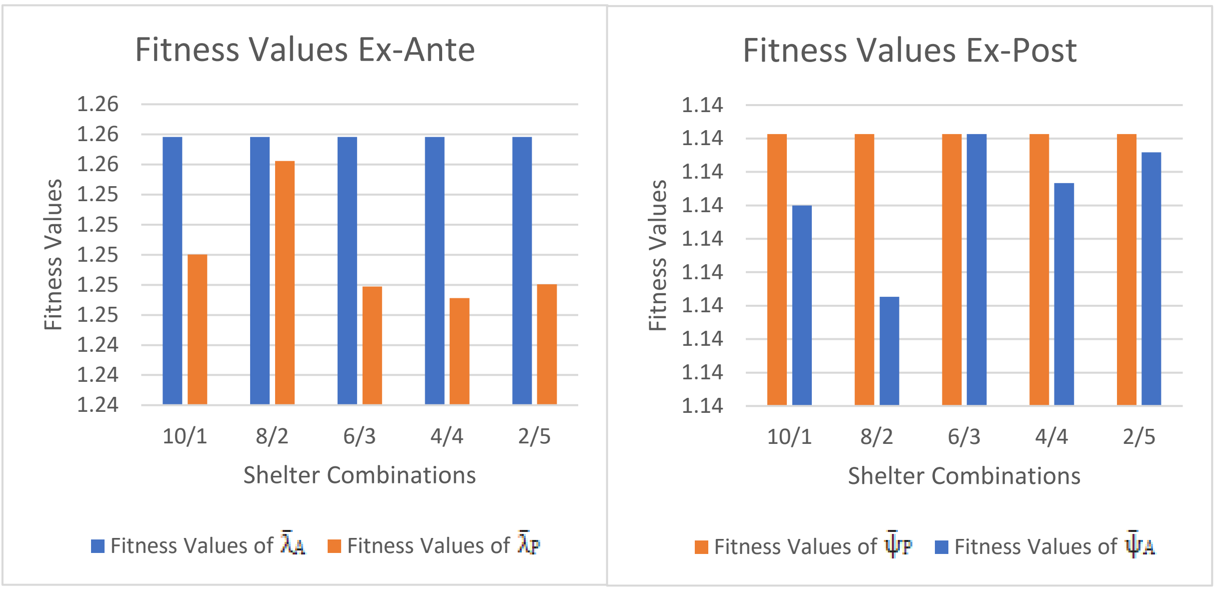


Figure 6-61 Impact: Fitness Values

The fitness values of ex-ante and ex-post ($\bar{\lambda}_A$, and $\bar{\psi}_P$) are identical for each shelter combination. This can easily happen when the number of potential locations is small compared to the number of shelters. For example, eleven shelters are built in combination 10/1, which means that only one of the twelve potential locations is not used. In 8/2, two potential locations are not used, and so on. In such cases it is very likely that the fitness values are the same and no meaningful comparison can be made.

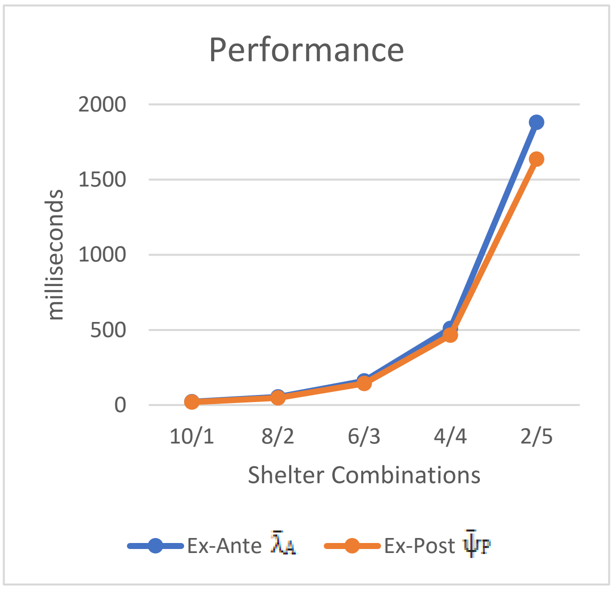


Figure 6-62 Impact: Performance

Figure 6-62 shows the performance of these combinations. The time that is required for each shelter combination is highly dependent on the number of temporary combinations given in Table 6-44. This graph shows exponential growth and it is therefore clear that an arbitrary large number of potential locations and temporary shelters cannot be chosen, as this would lead to enormous computation times.

6.7.4.3. Chosen Settings

In this subsection the settings that were eventually used for the “Budget” section are briefly discussed and compared to the previous subsection.

Figure 6-63 (from subsection 6.7.2) is strongly influenced by the data in Table 6-45. Here, to obtain more interesting results, the number of chromosomes is reduced from 20 to 6. For shelter combination 2/2, although there are now even 20 potential locations are available, the total number of combinations per generation and per seed is only 918. Combination 4/1 takes longer than almost all shelter combinations of Figure 6-62 because this combination requires on average several generations to get the result, whereas the combinations of Figure 6-62 usually do not even require a single generation. This depends strongly on the number of chromosomes.

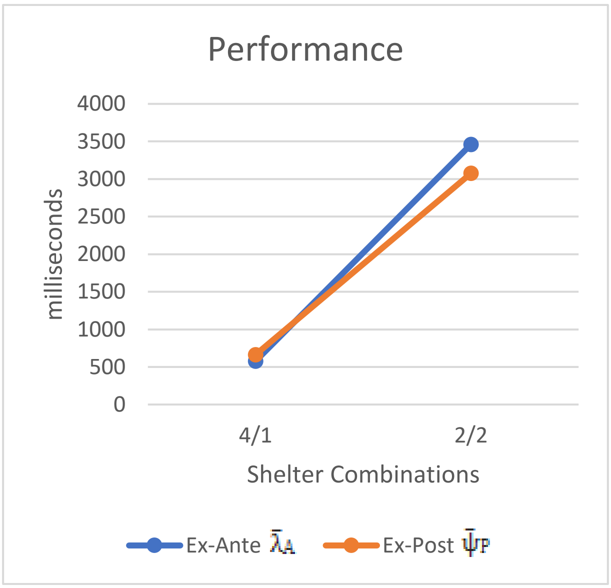


Figure 6-63 Chosen Settings: Performance

	Shelter Combinations 4/1	2/2
Permanent Combinations	6	6
Temporary Combinations	16	153
Total Solutions	96	918

Table 6-45 Chosen Settings: Number of Solutions

As explained earlier, the settings and the shelter combinations must be chosen carefully, otherwise the algorithm will either take a very long time to run or the results will not differ.

7. Conclusion

Research shows that fairness is important when decisions for disaster relief must be made. Whether a decision is perceived as fair depends on the perspective from which it is evaluated. In this work, a two-stage model is first solved by complete enumeration and then by a genetic algorithm, using the ex-ante and ex-post evaluation approach to determine the fairness of the solutions.

The experimental results of the large problem size indicate that, in general, the best ex-ante solution, plugged into the ex-post objective function, is usually much closer to the best ex-post solution than the best ex-post solution is to the best ex-ante solution when plugged into the ex-ante objective function. The results of the case study support this observation, as the best ex-ante solution, plugged into the ex-post objective function, is again closer to the best ex-post solution than vice versa.

In terms of costs, the ex-ante solutions generally perform worse than the ex-post solutions, with the exception of the case study, where the ex-ante solutions have (much) lower costs than the ex-post solutions. The same is true for the utilitarian costs. While ex-post provides solutions that are closer to the utilitarian solution for the large problem size, ex-ante generally delivers solutions with lower utilitarian costs than ex-post when the case study is concerned.

Although the ex-post approach takes slightly less time than the ex-ante approach to compute one generation, it usually requires more generations than ex-ante and therefore has poorer performance when the number of chromosomes is quite low. However, as observed in subsection 6.6.2.2, the performance of the ex-post approach tends to be better than the performance of the ex-ante approach when the number of chromosomes is larger.

References

Andreoni, J., Aydin, D., Barton, B., Bernheim, B. D., & Naecker, J. (2020). When Fair Isn't Fair: Understanding Choice Reversals Involving Social Preferences. *Journal of Political Economy*, *128*(5), 1673-1711.

Bentham, J. (1781). *An introduction to the principles of morals and legislation*. McMaster University Archive for the History of Economic Thought.

Bersano-Begey, T. F. (1997, July). Controlling exploration, diversity and escaping local optima in GP: adapting weights of training sets to model resource consumption. In *Late Breaking Papers at the 1997 Genetic Programming Conference* (pp. 7-10).

Borunda, A. (2019, February 06). The last five years were the hottest ever recorded. Retrieved October 20, 2020, from https://www.nationalgeographic.com/environment/2019/02/2018-fourth-warmest-year-ever-noaa-nasa-reports/

Darwin, C. R. 1859. *On the origin of species by means of natural selection, or the preservation of favoured races in the struggle for life*. London: John Murray.

Elloumi, S., Labbé, M., & Pochet, Y. (2004). A new formulation and resolution method for the p-center problem. *INFORMS Journal on Computing*, *16*(1), 84-94.

Fikar, C., Hirsch, P., & Nolz, P. C. (2018). Agent-based simulation optimization for dynamic disaster relief distribution. *Central European Journal of Operations Research*, *26*(2), 423-442.

Gautam, D., & Rodrigues, H. F. P. (Eds.). (2018). *Impacts and Insights of the Gorkha Earthquake*. Elsevier.

Grefenstette, J. J. (1986). Optimization of control parameters for genetic algorithms. *IEEE Transactions on systems, man, and cybernetics*, *16*(1), 122-128.

Guha-Sapir, D., Vos, F., Below, R., & Ponserre, S. (2011). Annual disaster statistical review 2010: the numbers and trends.

Guha-Sapir, D., Vos, F., Below, R., & Ponserre, S. (2012). Annual disaster statistical review 2011: the numbers and trends.

Gutjahr, W. J. (2020). Inequity-Averse Stochastic Decision Processes. *European Journal of Operational Research*, 288 (2021), 258-270.

Hedar, A. R., Ong, B. T., & Fukushima, M. (2007). Genetic algorithms with automatic accelerated termination. *Department of Applied Mathematics and Physics, Kyoto University, Tech. Rep*, *2*.

Holland, J. H. (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press.

Hussain, K., Salleh, M. N. M., Cheng, S., & Shi, Y. (2019). Metaheuristic research: a comprehensive survey. *Artificial Intelligence Review*, *52*(4), 2191-2233.

International Federation of Red Cross and Red Crescent Societies. (2016, October 13). World Disasters Report 2016 - Resilience: Saving lives today, investing for tomorrow. Retrieved October 20, 2020, from https://www.ifrc.org/en/news-and-media/press-releases/general/world-disasters-report-2016---resilience-saving-lives-today-investing-for-tomorrow/

Jebari, K., & Madiafi, M. (2013). Selection methods for genetic algorithms. *International Journal of Emerging Sciences*, *3*(4), 333-344.

Leung, Y. W., & Wang, Y. (2001). An orthogonal genetic algorithm with quantization for global numerical optimization. *IEEE Transactions on Evolutionary computation*, *5*(1), 41-53.

Mazepus, H., & van Leeuwen, F. (2020). Fairness matters when responding to disasters: An experimental study of government legitimacy. *Governance*, *33*(3), 621-637.

Mehta, T. (n.d.). Draw a circle. Retrieved November 06, 2020, from http://www.tushar-mehta.com/publish_train/xl_vba_cases/0610%20draw%20circle.shtml

Miller, B. L., & Goldberg, D. E. (1995). Genetic algorithms, tournament selection, and the effects of noise. *Complex systems*, *9*(3), 193-212.

Mitchell, M. (1998). *An introduction to genetic algorithms*. MIT press.

Mostajabdaveh, M., Gutjahr, W. J., & Salman, F. S. (2020). Inequity-averse shelter location for disaster preparedness. *IISE Transactions*, *1-41*.

Nepal (26. May 2020). Bing Maps. Microsoft Corporation. Created with Microsoft 3D Maps for Excel.

Pongcharoen, P., Khadwilard, A., & Hicks, C. (2008). A Genetic Algorithm with a New Repair Process for Solving Multi-stage, Multi-machine, Multi-product Scheduling Problems. *Industrial Engineering & Management Systems*, *7*(3), 204-213.

Rahman, M. M., & Bai, L. (2018). Probabilistic seismic hazard assessment of Nepal using multiple seismic source models. *Earth and Planetary Physics*, *2*(4), 327-341.

Rawls, J. (1957). I. Justice as Fairness. The Journal of Philosophy, Vol. 54, No. 22, American Philosophical Association Eastern Division: Symposium Papers to be Presented at the Fifty-Fourth Annual Meeting, Harvard University, December 27-29, 1957 (Oct. 24, 1957), pp. 653-662

REACH Initiative (n.d.). Who We Are. Retrieved October 16, 2020, from https://www.reach-initiative.org/who-we-are/

REACH Initiative (2015). Nepal - April 2015 Earthquake: Estimated Affected Areas as of 25th April 2015 - Nepal. Retrieved October 16, 2020, from https://reliefweb.int/map/nepal/nepal-april-2015-earthquake-estimated-affected-areas-25th-april-2015

Reeves, C. (2003). Genetic algorithms. In *Handbook of metaheuristics* (pp. 55-82). Springer, Boston, MA.

Reid, K. (2018). 2015 Nepal earthquake: Facts, GAQs, and how to help. Retrieved from https://www.worldvision.org/disaster-relief-news-stories/2015-nepal-earthquake-facts

Rodriguez, J., Vos, F., Below, R., & Guha-Sapir, D. (2009). Annual disaster statistical review 2008: The numbers and trends.

Rubin, J. M. (2011). Retrieved November 07, 2020, from https://engineering.mit.edu/engage/ask-an-engineer/can-a-computer-generate-a-truly-random-number/

Sanderson, D., & Sharma, A. (2016). World Disasters Report 2016. *Resilience: saving lives today, investing for tomorrow. International Federation of Red Cross and Red Crescent Societies*.

Sunaryo. (2017, November). Study of Peak Ground Acceleration (PGA) by means of microzonation data: Case Study on Batubesi Dam of Nuha, East Luwu, South Sulawesi, Indonesia. In *AIP Conference Proceedings* (Vol. 1908, No. 1, p. 030013). AIP Publishing LLC.

The C++ Resources Network. (n.d.). Retrieved November 07, 2020, from www.cplusplus.com/reference/random/

Vidal, T., Crainic, T. G., Gendreau, M., Lahrichi, N., & Rei, W. (2011). A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, *60*(3), 611-624.

Walls, C. (2016, November 18). Tasks and scheduling. Retrieved November 14, 2020, from https://www.embedded.com/tasks-and-scheduling/

Weisstein, Eric W. (n.d.). "Binomial Coefficient." From *MathWorld*--A Wolfram Web Resource. https://mathworld.wolfram.com/BinomialCoefficient.html

XLware. (n.d.). Download. Retrieved November 06, 2020, from https://www.libxl.com/download.html

Appendix A – Complete Enumeration

```

void calcCombinations(const vector<int>& indicesPotLoc, vector<int> storedCombinations, int start, int end, int numStoredCombinations, int numPermanentLocations, vector<vector<int>>& combinations) {
    if (numStoredCombinations == numPermanentLocations) {
        combinations.push_back(storedCombinations);
        return;
    }
    for (; start <= end && end - start + 1 >= numPermanentLocations - numStoredCombinations; ++start) {
        storedCombinations[numStoredCombinations] = indicesPotLoc[start];
        calcCombinations(indicesPotLoc, storedCombinations, start + 1, end, numStoredCombinations + 1, numPermanentLocations, combinations);
    }
}

void possibleCombinations(const vector<int>& indicesPotLoc, int numPotentialLocations, int numPermanentLocations, vector<vector<int>>& combinations) {
    vector<int> storedCombinations(numPermanentLocations);
    calcCombinations(indicesPotLoc, storedCombinations, 0, numPotentialLocations - 1, 0, numPermanentLocations, combinations);
}

void nearestDistVector(vector<vector<vector<vector<float>>>>& nearestDistances, const vector<vector<bool>>& scenarios, const vector<vector<vector<int>>>& possibleCombinations, const vector<vector<float>>& distanceMatrix) {
    int permComb = nearestDistances.size();
    int numScen = nearestDistances[0].size();
    int tempComb = nearestDistances[0][0].size();
    int numPopNodes = nearestDistances[0][0][0].size();
    int affectedLoc = possibleCombinations[0][0].size();
    for (int permanent = 0; permanent < permComb; ++permanent)
        for (int scen = 0; scen < numScen; ++scen)
            for (int temporary = 0; temporary < tempComb; ++temporary)
                for (int popNode = 0; popNode < numPopNodes; ++popNode)
                    if (scenarios[scen][popNode]) {
                        float minDist = FLT_MAX;
                        for (int location = 0; location < affectedLoc; ++location)
                            if (minDist > distanceMatrix[possibleCombinations[permanent][temporary][location]][popNode]) {
                                minDist = distanceMatrix[possibleCombinations[permanent][temporary][location]][popNode];
                                nearestDistances[permanent][scen][temporary][popNode] = distanceMatrix[possibleCombinations[permanent][temporary][location]][popNode];
                            }
                    }
}

```

```cpp
void getMaximumValues(const vector<vector<vector<vector<float>>>>& nearestDistances, vector<vector<vector<vector<float>>>>& maximumDistances) {
    int permComb = nearestDistances.size();
    int numScen = nearestDistances[0].size();
    int tempComb = nearestDistances[0][0].size();
    int numPopNodes = nearestDistances[0][0][0].size();
    for (int permanent = 0; permanent < permComb; ++permanent)
        for (int scen = 0; scen < numScen; ++scen)
            for (int temporary = 0; temporary < tempComb; ++temporary) {
                float maxDist = 0;
                for (int popNode = 0; popNode < numPopNodes; ++popNode)
                    if (nearestDistances[permanent][scen][temporary][popNode] > maxDist) {
                        maxDist = nearestDistances[permanent][scen][temporary][popNode];
                        maximumDistances[permanent][scen][temporary][0] = maxDist;
                    }
            }
}

void getMinimumDistances(const vector<vector<vector<vector<float>>>>& maximumDistances, vector<vector<map<int, float>>>& minimumDistances, mt19937 eng) {
    int permComb = maximumDistances.size();
    int numScen = maximumDistances[0].size();
    int tempComb = maximumDistances[0][0].size();
    for (int permanent = 0; permanent < permComb; ++permanent)
        for (int scen = 0; scen < numScen; ++scen) {
            float minDist = FLT_MAX;
            vector<int> identicalDistancesIndices;
            for (int temporary = 0; temporary < tempComb; ++temporary)
                if ((maximumDistances[permanent][scen][temporary][0] >(minDist - 0.001)) && (maximumDistances[permanent][scen][temporary][0] < (minDist + 0.001)))
                    identicalDistancesIndices.push_back(temporary);
                else if (maximumDistances[permanent][scen][temporary][0] < minDist) {
                    identicalDistancesIndices.clear();
                    identicalDistancesIndices.push_back(temporary);
                    minDist = maximumDistances[permanent][scen][temporary][0];
                }
            uniform_int_distribution<> distrIdenticalDistances(0, identicalDistancesIndices.size() - 1);
            int randNum = distrIdenticalDistances(eng);
            minimumDistances[permanent][scen][identicalDistancesIndices[randNum]] = minDist;
        }
}

void calculatePermCombValues(const vector<vector<map<int, float>>>& minimumDistances, const vector<float>& probabilities, vector<float>& permCombValues) {
    int permComb = minimumDistances.size();
    int numScen = minimumDistances[0].size();
    for (int permanent = 0; permanent < permComb; ++permanent) {
```

```cpp
    float fairnessValue = 0;
    for (int scen = 0; scen < numScen; ++scen)
      fairnessValue += (((minimumDistances[permanent][scen].begin())->second)*probabilities[scen]);
    permCombValues[permanent] = fairnessValue;
  }
}

void findSmallestValueIndex(float& smallestDistance, const vector<float>& permCombValues, int& index, mt19937 eng) {
  int permComb = permCombValues.size();
  vector<int> identicalDistancesIndices;
  for (int permanent = 0; permanent < permComb; ++permanent)
    if ((smallestDistance >(permCombValues[permanent] - 0.001)) && (smallestDistance < (permCombValues[permanent] + 0.001)))
      identicalDistancesIndices.push_back(permanent);
    else if (smallestDistance > permCombValues[permanent]) {
      identicalDistancesIndices.clear();
      identicalDistancesIndices.push_back(permanent);
      smallestDistance = permCombValues[permanent];
    }
  uniform_int_distribution<> distrIdenticalDistances(0, identicalDistancesIndices.size() - 1);
  index = identicalDistancesIndices[distrIdenticalDistances(eng)];
}

void printExPostSolution(int index, const vector<vector<map<int, float>>>& minimumDistances, const vector<vector<int>>& permanentCombinations, const vector<vector<vector<int>>>& temporaryCombinations, const vector<vector<vector<vector<float>>>>& nearestDistances) {
  cout << "\nEx-Post: \nbuild permanent location/s: \n";
  int permComb = permanentCombinations[0].size();
  int numScen = nearestDistances[0].size();
  int tempComb = temporaryCombinations[0][0].size();
  int numPopNodes = nearestDistances[0][0][0].size();
  float costs;
  for (int permanent = 0; permanent < permComb; ++permanent)
    cout << permanentCombinations[index][permanent] << " ";
  for (int scen = 0; scen < numScen; ++scen) {
    costs = 0;
    cout << "\nfor scenario " << scen << " build temporary location/s: \n";
    for (int temp = 0; temp < tempComb; ++temp)
      cout << temporaryCombinations[index][(minimumDistances[index][scen].begin())->first][temp] << " ";
    for (int popNode = 0; popNode < numPopNodes; ++popNode)
      costs += nearestDistances[index][scen][(minimumDistances[index][scen].begin())->first][popNode];
    cout << "\nwith costs: " << costs << endl;
  }
}

void getMinimumVectors(const vector<vector<map<int, float>>>& minimumDistances, const vector<vector<vector<vector<float>>>>& nearestDistances, vector<vector<map<int, vector<float>>>>& minimumVectors) {
  int permComb = nearestDistances.size();
```

```cpp
    int numScen = nearestDistances[0].size();
    for (int permanent = 0; permanent < permComb; ++permanent)
        for (int scen = 0; scen < numScen; ++scen) {
            minimumDistances[permanent][scen].begin()->first;
            minimumVectors[permanent][scen][minimumDistances[permanent][scen].begin()->first] = nearestDistances[permanent][scen][minimumDistances[permanent][scen].begin()->first];
        }
}

void calculateAnteScenarioValues(const vector<vector<map<int, vector<float>>>>& minimumVectors, const vector<float>& probabilities, vector<vector<float>>& ante_sumOfScenarios) {
    int numPermComb = minimumVectors.size();
    int numScen = minimumVectors[0].size();
    int numPopNodes = ante_sumOfScenarios[0].size();
    for (int permanent = 0; permanent < numPermComb; ++permanent)
        for (int node = 0; node < numPopNodes; ++node) {
            float fairness = 0;
            for (int scen = 0; scen < numScen; ++scen)
                fairness += (((minimumVectors[permanent][scen].begin())->second[node])*probabilities[scen]);
            ante_sumOfScenarios[permanent][node] = fairness;
        }
}

void getExAnteVector(const vector<vector<float>>& ante_sumOfScenarios, vector<float>& maxFairnessValues) {
    int permComb = ante_sumOfScenarios.size();
    int numPopNodes = ante_sumOfScenarios[0].size();
    for (int permanent = 0; permanent < permComb; ++permanent) {
        float maxDist = 0;
        for (int node = 0; node < numPopNodes; ++node)
            if (ante_sumOfScenarios[permanent][node]> maxDist) {
                maxFairnessValues[permanent] = ante_sumOfScenarios[permanent][node];
                maxDist = ante_sumOfScenarios[permanent][node];
            }
    }
}



void getMinimumVectorAnte(const vector<float>& maxFairnessValues, float& minDist, int& indexAnteMin, mt19937 eng) {
    int permComb = maxFairnessValues.size();
    vector<int> identicalDistancesIndices;
    for (int permanent = 0; permanent < permComb; ++permanent)
        if ((minDist < (maxFairnessValues[permanent] + 0.001)) && (minDist > (maxFairnessValues[permanent] - 0.001)))
            identicalDistancesIndices.push_back(permanent);
        else if (minDist > maxFairnessValues[permanent]) {
            identicalDistancesIndices.clear();
            identicalDistancesIndices.push_back(permanent);
            minDist = maxFairnessValues[permanent];
        }
    uniform_int_distribution<> distrIdenticalDistances(0, identicalDistancesIndices.size() - 1);
    indexAnteMin = identicalDistancesIndices[distrIdenticalDistances(eng)];
}

void printExAnteSolution(const vector<vector<int>>& permanentCombinations, const vector<vector<vector<int>>>& temporaryCombinations, const int indexAnteMin, const vector<vector<map<int, vector<float>>>>& minimumVectors, const vector<vector<vector<vector<float>>>>& nearestDistances) {
    int permComb = permanentCombinations[0].size();
    int numScen = nearestDistances[0].size();
    int tempComb = temporaryCombinations[0][0].size();
    int numPopNodes = nearestDistances[0][0][0].size();
    float costs;
    cout << "\nEx-Ante: \nbuild permanent location/s: \n";
    for (int permanent = 0; permanent < permComb; ++permanent)
        cout << permanentCombinations[indexAnteMin][permanent] << " ";
    for (int scen = 0; scen < numScen; ++scen) {
        costs = 0;
        cout << "\nfor scenario " << scen << " build temporary location/s: \n";
        for (int temp = 0; temp < tempComb; ++temp)
            cout << temporaryCombinations[indexAnteMin][(minimumVectors[indexAnteMin][scen].begin())->first][temp] << " ";
        for (int popNode = 0; popNode < numPopNodes; ++popNode)
            costs += nearestDistances[indexAnteMin][scen][(minimumVectors[indexAnteMin][scen].begin())->first][popNode];
        cout << "\nwith costs: " << costs << endl;
    }
}

void utilitarianObjective(const vector<vector<vector<vector<float>>>>& nearestDistances, const vector<int>& weight, const vector<float>& probabilities) {
    int permComb = nearestDistances.size();
    int numScen = nearestDistances[0].size();
    int tempComb = nearestDistances[0][0].size();
    int numPopNodes = nearestDistances[0][0][0].size();
    vector<vector<vector<float>>> storedTempCombUtility(permComb, vector<vector<float>>(numScen, vector<float>(tempComb, 0)));
    for (int perm = 0; perm < permComb; ++perm)
        for (int scen = 0; scen < numScen; ++scen)
            for (int temp = 0; temp < tempComb; ++temp)
                for (int pop = 0; pop < numPopNodes; ++pop)
                    storedTempCombUtility[perm][scen][temp] += (nearestDistances[perm][scen][temp][pop] * weight[pop]);
    vector<vector<float>> storedScenarioUtility(permComb, vector<float>(numScen, 0));
    for (int perm = 0; perm < permComb; ++perm)
        for (int scen = 0; scen < numScen; ++scen) {
            float minDist = FLT_MAX;
            for (int temp = 0; temp < tempComb; ++temp)
                if (storedTempCombUtility[perm][scen][temp] < minDist)
                    minDist = storedTempCombUtility[perm][scen][temp];
            storedScenarioUtility[perm][scen] = minDist;
        }
    vector<float> storedPermUtility(permComb, 0);
    for (int perm = 0; perm < permComb; ++perm)
        for (int scen = 0; scen < numScen; ++scen)
            storedPermUtility[perm] += (storedScenarioUtility[perm][scen] *
              probabilities[scen]);
    float minDist = FLT_MAX;
    int indexUtility = 0;
    for (int perm = 0; perm < permComb; ++perm)
        if (storedPermUtility[perm] < minDist) {
            minDist = storedPermUtility[perm];
            indexUtility = perm;
        }
    cout << "\nUtilitarian Function ";
    for (int scen = 0; scen < numScen; ++scen)
        cout << "\nScenario " << scen << " costs: " << storedScenarioUtility
          [indexUtility][scen];
}
```

Appendix B – Genetic Algorithm

```

void createInitialPopulation(vector<vector<int>>& population, const int numPermanentLocations, const int numPotentialLocations, mt19937& eng) {
    int randomNumber;
    int numChromosomes = population.size();
    for (int chromosome = 0; chromosome < numChromosomes; ++chromosome) {
        vector<int> availableLocations(numPotentialLocations);
        for (int indexPotLoc = 0; indexPotLoc < numPotentialLocations; ++indexPotLoc)
            availableLocations[indexPotLoc] = indexPotLoc;
        for (int permanentLocation = 0; permanentLocation < numPermanentLocations; ++permanentLocation) {
            uniform_int_distribution<> distrPotLoc(0, availableLocations.size() - 1);
            randomNumber = distrPotLoc(eng);
            population[chromosome][permanentLocation] = availableLocations[randomNumber];
            availableLocations.erase(availableLocations.begin() + randomNumber);
        }
    }
}


void selectionProcessRouletteWheel(const int numChromosomes, mt19937& eng, vector<int>& matingPoolIndices, const vector<pair<float, int>>& fitnessValues, int bestChromosomeIndex, int worstChromosomeIndex) {
    matingPoolIndices[0] = worstChromosomeIndex;
    matingPoolIndices[1] = bestChromosomeIndex;
    uniform_int_distribution<> distrLowFitness(0, numChromosomes *0.3);
    for (int chromIndex = 2; chromIndex < 4; ++chromIndex) {
        int randNum = distrLowFitness(eng);
        matingPoolIndices[chromIndex] = randNum;
    }
    uniform_int_distribution<> distrLowOrMedium(0, numChromosomes *0.6);
    for (int chromIndex = 4; chromIndex < 6; ++chromIndex) {
        int randNum = distrLowOrMedium(eng);
        matingPoolIndices[chromIndex] = randNum;
    }
    float sumFitnessValues = 0;
    for (int chromosome = 0; chromosome < numChromosomes; ++chromosome)
        sumFitnessValues += fitnessValues[chromosome].first;
    vector<float> cumulatedPercentages(numChromosomes + 1, 0);
    for (int chromosome = 0; chromosome < numChromosomes; ++chromosome)
        cumulatedPercentages[chromosome + 1] = fitnessValues[chromosome].first / sumFitnessValues;
    for (int chromosome = 1; chromosome < numChromosomes; ++chromosome)
        cumulatedPercentages[chromosome] += cumulatedPercentages[chromosome - 1];
    uniform_real_distribution<> distrPercentage(0, 1);
    float randomNumber;
    int chosenChromosome;
```

```cpp
    for (int chromosome = 6; chromosome < numChromosomes; ++chromosome) {
        randomNumber = distrPercentage(eng);
        chosenChromosome = 0;
        for (int chromosome = 1; chromosome < numChromosomes; ++chromosome)
            if (randomNumber > cumulatedPercentages[chromosome])
                chosenChromosome = chromosome;
            else break;
        matingPoolIndices[chromosome] = chosenChromosome;
    }
}

void selectionProcessTournament(const int numChromosomes, mt19937& eng, vector<int>& matingPoolIndices, const vector<pair<float, int>>& fitnessValues, int bestChromosomeIndex, int worstChromosomeIndex) {
    int tournamentSize = 4;
    matingPoolIndices[0] = worstChromosomeIndex;
    matingPoolIndices[1] = bestChromosomeIndex;
    uniform_int_distribution<> distrLowFitness(0, numChromosomes *0.3);
    for (int chromIndex = 2; chromIndex < 4; ++chromIndex) {
        int randNum = distrLowFitness(eng);
        matingPoolIndices[chromIndex] = randNum;
    }
    uniform_int_distribution<> distrLowOrMedium(0, numChromosomes *0.6);
    for (int chromIndex = 4; chromIndex < 6; ++chromIndex) {
        int randNum = distrLowOrMedium(eng);
        matingPoolIndices[chromIndex] = randNum;
    }
    vector<int> activeTournament(tournamentSize, 0);
    for (int chromosome = 6; chromosome < numChromosomes; ++chromosome) {
        vector<int> availableChromosomes(numChromosomes);
        for (int indexChrom = 0; indexChrom < numChromosomes; ++indexChrom)
            availableChromosomes[indexChrom] = indexChrom;
        for (int tournParticipant = 0; tournParticipant < tournamentSize; ++tournParticipant) {
            uniform_int_distribution<> distrAvailableChrom(0, availableChromosomes.size() - 1);
            int randNum = distrAvailableChrom(eng);
            activeTournament[tournParticipant] = availableChromosomes[randNum];
            availableChromosomes.erase(availableChromosomes.begin() + randNum);
        }
        float maxFitness = 0;
        int indexMaxFitness = 0;
        for (int indexParticipant = 0; indexParticipant < tournamentSize; ++indexParticipant)
            if (fitnessValues[activeTournament[indexParticipant]].first > maxFitness) {
                maxFitness = fitnessValues[activeTournament[indexParticipant]].first;
                indexMaxFitness = indexParticipant;
            }
        matingPoolIndices[chromosome] = fitnessValues[activeTournament[indexMaxFitness]].second;
    }
}
```

```cpp

void crossoverPosition1point(const int numPermanentLocations, const int numChromosomes, mt19937& eng, vector<int>& crossPointPositions) {
    uniform_int_distribution<> distrPosCrossover((numPermanentLocations - 1)*0.2, (numPermanentLocations - 1)*0.8);
    for (int chromPair = 0; chromPair < numChromosomes / 2; ++chromPair)
        crossPointPositions[chromPair] = distrPosCrossover(eng);
}

void crossoverPosition2point(const int numPermanentLocations, const int numChromosomes, mt19937& eng, vector<vector<int>>& crossPointPositions) {
    uniform_int_distribution<> distrPosCrossover1((numPermanentLocations - 1)*0.1, (numPermanentLocations - 1)*0.5);
    int position1;
    int position2;
    for (int chromPair = 0; chromPair < numChromosomes / 2; ++chromPair) {
        position1 = distrPosCrossover1(eng);
        crossPointPositions[chromPair][0] = position1;
        uniform_int_distribution<> distrPosCrossover2(position1 + 1, fmax((numPermanentLocations - 1)*0.9, position1 + 1));
        position2 = distrPosCrossover2(eng);
        crossPointPositions[chromPair][1] = position2;
    }
}


void performCrossover1point(const int numPermanentLocations, const vector<int>& crossPointPositions, vector<vector<int>>& population, const vector<int>& matingPoolIndices) {
    int numChromosomes = population.size();
    vector<vector<int>> newPopulation(numChromosomes, vector<int>(numPermanentLocations, 0));
    for (int gene = 0; gene < numPermanentLocations; ++gene) {
        newPopulation[numChromosomes - 2][gene] = population[matingPoolIndices[0]][gene];
        newPopulation[numChromosomes - 1][gene] = population[matingPoolIndices[1]][gene];;
    }
    for (int chromosome = 0; chromosome < numChromosomes / 2 - 1; ++chromosome) {
        for (int gene = 0; gene < crossPointPositions[chromosome]; ++gene) {
            newPopulation[chromosome * 2][gene] = population[matingPoolIndices[chromosome * 2 + 2]][gene];
            newPopulation[chromosome * 2 + 1][gene] = population[matingPoolIndices[chromosome * 2 + 1 + 2]][gene];
        }
        for (int gene = 0; gene < numPermanentLocations - crossPointPositions[chromosome]; ++gene) {
            newPopulation[chromosome * 2][gene + crossPointPositions[chromosome]] = population[matingPoolIndices[chromosome * 2 + 1 + 2]][gene + crossPointPositions[chromosome]];
            newPopulation[chromosome * 2 + 1][gene + crossPointPositions[chromosome]] = population[matingPoolIndices[chromosome * 2 + 2]][gene + crossPointPositions[chromosome]];
        }
    }
    population.clear();
    population = newPopulation;
}


void performCrossover2point(const int numPermanentLocations, const vector<vector<int>>& crossPointPositions, vector<vector<int>>& population, const vector<int>& matingPoolIndices) {
    int numChromosomes = population.size();
    vector<vector<int>> newPopulation(numChromosomes, vector<int>(numPermanentLocations, 0));
    for (int gene = 0; gene < numPermanentLocations; ++gene) {
        newPopulation[numChromosomes - 2][gene] = population[matingPoolIndices[0]][gene];
        newPopulation[numChromosomes - 1][gene] = population[matingPoolIndices[1]][gene];;
    }
    for (int chromosome = 0; chromosome < numChromosomes / 2 - 1; ++chromosome) {
        for (int gene = 0; gene < crossPointPositions[chromosome][0]; ++gene) {
            newPopulation[chromosome * 2][gene] = population[matingPoolIndices[chromosome * 2 + 2]][gene];
            newPopulation[chromosome * 2 + 1][gene] = population[matingPoolIndices[chromosome * 2 + 1 + 2]][gene];
        }
        for (int gene = 0; gene < crossPointPositions[chromosome][1] - crossPointPositions[chromosome][0]; ++gene) {
            newPopulation[chromosome * 2][gene + crossPointPositions[chromosome][0]] = population[matingPoolIndices[chromosome * 2 + 1 + 2]][gene + crossPointPositions[chromosome][0]];
            newPopulation[chromosome * 2 + 1][gene + crossPointPositions[chromosome][0]] = population[matingPoolIndices[chromosome * 2 + 2]][gene + crossPointPositions[chromosome][0]];
        }
        for (int gene = 0; gene < numPermanentLocations - crossPointPositions[chromosome][1]; ++gene) {
            newPopulation[chromosome * 2][gene + crossPointPositions[chromosome][1]] = population[matingPoolIndices[chromosome * 2 + 2]][gene + crossPointPositions[chromosome][1]];
            newPopulation[chromosome * 2 + 1][gene + crossPointPositions[chromosome][1]] = population[matingPoolIndices[chromosome * 2 + 1 + 2]][gene + crossPointPositions[chromosome][1]];
        }
    }
    population.clear();
    population = newPopulation;
}


void repairMechanism(const int numPermanentLocations, vector<vector<int>>& population, const int numPotentialLocations, mt19937& eng) {
    int numChromosomes = population.size();
    for (int chromosome = 0; chromosome < numChromosomes; ++chromosome) {
        while (true) {
            bool shouldbreak = 0;
```

```
      for (int compare1 = 0; compare1 < numPermanentLocations - 1; +
       +compare1) {
        for (int compare2 = compare1 + 1; compare2 <
         numPermanentLocations; ++compare2) {
          if (population[chromosome][compare1] == population
          [chromosome][compare2]) {
            shouldbreak = 1;
            break;
          }
        }
        if (shouldbreak) break;
      }
      if (shouldbreak) {
        vector<int> duplicates(numPotentialLocations, 0);
        for (int gene = 0; gene < numPermanentLocations; ++gene)
          ++duplicates[population[chromosome][gene]];
        for (int potLoc = 0; potLoc < numPotentialLocations; ++potLoc)
         {
          if (duplicates[potLoc] > 1) {
            vector<int> storeDuplPos;
            for (int gene = 0; gene < numPermanentLocations; +
          +gene)
              if (potLoc == population[chromosome][gene])
                storeDuplPos.push_back(gene);
            vector<int> remainingLocs;
            for (int potLoc = 0; potLoc < numPotentialLocations; +
          +potLoc)
              if (duplicates[potLoc] == 0)
                remainingLocs.push_back(potLoc);
            uniform_int_distribution<> distrRemovePos(0,
          storeDuplPos.size() - 1);
            int randomNumberRemove = distrRemovePos(eng);
            uniform_int_distribution<> distrAddPos(0,
          remainingLocs.size() - 1);
            int randomNumberAdd = distrAddPos(eng);
            population[chromosome][storeDuplPos
          [randomNumberRemove]] = remainingLocs[randomNumberAdd];
          }
        }
      }
      else break;
    }
  }
}


void mutation(const int numChromosomes, const int numPermanentLocations,
 mt19937& eng, const float mutationPercentage, const int
 numPotentialLocations, vector<vector<int>>& population) {
  uniform_int_distribution<> distrMutation(0, 99);
  int randomNumber;
  for (int chromosome = 0; chromosome < numChromosomes - 1; ++chromosome) {
    for (int gene = 0; gene < numPermanentLocations; ++gene) {
      randomNumber = distrMutation(eng);
      if (randomNumber < mutationPercentage) {
        vector<int> remainingLoc;
```

```
                for (int indexPotLoc = 0; indexPotLoc < numPotentialLocations; ++indexPotLoc)
                    if (find(population[chromosome].begin(), population[chromosome].end(), indexPotLoc) == population[chromosome].end())
                        remainingLoc.push_back(indexPotLoc);
                uniform_int_distribution<> distrPosition(0, remainingLoc.size() - 1);
                int randomNumber = distrPosition(eng);
                population[chromosome][gene] = remainingLoc[randomNumber];
            }
        }
    }
}
```

Appendix C – Large Problem Size

a) Ex-Ante	Potential Locations					
	50	60	70	80	90	100
Arithmetic Mean	30.39	31.64	35.35	36.84	37.26	42.99
0%	1.00	1.00	0	0	1.00	1.00
25%	2.75	8.75	8.75	11.00	10.00	14.00
50%	21.00	21.50	30.00	31.00	31.00	36.50
75%	51.50	49.00	52.50	52.25	52.50	65.25
100%	108.00	102.00	173.00	173.00	132.00	133.00
Standard Deviation	30.66	29.17	33.36	32.98	32.02	33.70
Variance	940.30	850.96	1112.92	1087.93	1025.59	1135.97
Skewness	0.88	0.90	1.58	1.35	0.98	0.71
Range	107	101	173	173	131	132
Interquartile Range	48.75	40.25	43.75	41.25	42.5	51.25

a) Ex-Post	Potential Locations					
	50	60	70	80	90	100
Arithmetic Mean	41.21	40.08	51.13	38.52	42.79	43.06
0%	1.00	1.00	1.00	1.00	1.00	1.00
25%	15.75	12.75	24.75	12.75	15.00	14.75
50%	36.00	32.00	53.50	31.50	32.50	35.00
75%	57.25	61.25	79.25	54.25	61.25	60.25
100%	130.00	159.00	157.00	138.00	160.00	155.00
Standard Deviation	31.25	35.06	35.00	32.47	36.13	32.72
Variance	976.45	1229.41	1225.06	1054.49	1305.08	1070.74
Skewness	0.86	1.06	0.41	1.04	1.05	0.85
Range	129.00	158.00	156.00	137.00	159.00	154.00
Interquartile Range	41.50	48.50	54.50	41.50	46.25	45.50

b) Ex-Ante	Potential Locations					
	50	60	70	80	90	100
Arithmetic Mean	22.14	24.67	35.35	36.84	37.26	42.99
0%	1.00	1.00	0	0	1.00	1.00
25%	1.00	2.75	8.75	11.00	10.00	14.00
50%	13.50	15.00	30.00	31.00	31.00	36.50
75%	37.25	39.00	52.50	52.25	52.50	65.25
100%	123.00	99.00	173.00	173.00	132.00	133.00
Standard Deviation	24.76	25.73	33.36	32.98	32.02	33.70
Variance	612.87	661.90	1112.92	1087.93	1025.59	1135.97
Skewness	1.38	1.06	1.58	1.35	0.98	0.71
Range	122.00	98.00	173.00	173.00	131.00	132.00
Interquartile Range	36.25	36.25	43.75	41.25	42.50	51.25

b) Ex-Post	Potential Locations					
	50	60	70	80	90	100
Arithmetic Mean	31.88	32.11	32.10	38.82	34.68	40.67
0%	1.00	1.00	1.00	1.00	1.00	1.00
25%	4.00	7.25	7.00	8.00	7.50	9.75
50%	23.50	26.50	26.50	25.50	29.50	32.00
75%	52.75	51.00	47.50	60.00	53.00	60.50
100%	115.00	113.00	169.00	177.00	143.00	186.00
Standard Deviation	30.86	28.58	30.48	37.42	31.95	37.51
Variance	952.05	816.89	929.22	1400.41	1020.66	1407.01
Skewness	0.90	0.73	1.43	1.28	1.01	1.39
Range	114.00	112.00	168.00	176.00	142.00	185.00
Interquartile Range	48.75	43.75	40.50	52.00	45.50	50.75

c) Ex-Ante	Potential Locations					
	50	60	70	80	90	100
Arithmetic Mean	27.96	27.11	30.17	31.88	30.68	36.02
0%	1.00	1.00	1.00	0	1.00	1.00
25%	5.00	7.00	7.00	11.75	7.00	13.00
50%	19.00	18.50	22.50	25.00	23.50	29.50
75%	43.00	39.25	48.00	46.25	46.25	49.25
100%	108.00	110.00	128.00	99.00	112.00	174.00
Standard Deviation	26.92	25.67	27.84	25.75	28.08	31.08
Variance	724.60	659.05	775.29	662.96	788.66	965.92
Skewness	1.02	1.31	1.09	0.79	0.98	1.48
Range	107.00	109.00	127.00	99.00	111.00	173.00
Interquartile Range	38.00	32.25	41.00	34.50	39.25	36.25

c) Ex-Post	Potential Locations					
	50	60	70	80	90	100
Arithmetic Mean	38.37	44.42	44.56	45.85	47.42	42.03
0%	1.00	1.00	1.00	1.00	1.00	1.00
25%	10.50	14.75	16.75	16.75	21.00	17.00
50%	30.00	36.50	40.00	39.00	44.00	37.50
75%	58.00	69.25	61.50	64.50	68.50	62.75
100%	147.00	139.00	204.00	135.00	170.00	129.00
Standard Deviation	31.94	36.13	35.75	35.74	34.83	30.51
Variance	1020.32	1305.46	1277.95	1277.36	1213.20	931.16
Skewness	0.82	0.77	1.38	0.65	1.06	0.58
Range	146.00	138.00	203.00	134.00	169.00	128.00
Interquartile Range	47.50	54.50	44.75	47.75	47.50	45.75

d) Ex-Ante	Potential Locations					
	50	60	70	80	90	100
Arithmetic Mean	21.95	22.85	23.33	23.24	23.92	31.20
0%	1.00	1.00	1.00	0	0	1.00
25%	2.00	2.75	2.00	4.00	3.00	5.00
50%	13.00	17.50	18.00	13.50	14.50	23.50
75%	37.25	35.50	40.00	39.75	37.00	45.00
100%	113.00	89.00	107.00	89.00	155.00	129.00
Standard Deviation	24.16	22.90	23.77	24.62	26.96	30.56
Variance	583.54	524.37	564.83	605.90	726.72	933.76
Skewness	1.30	0.98	1.08	1.09	1.79	1.17
Range	112.00	88.00	106.00	89.00	155.00	128.00
Interquartile Range	35.25	32.75	38.00	35.75	34.00	40.00

d) Ex-Post	Potential Locations					
	50	60	70	80	90	100
Arithmetic Mean	29.36	29.87	34.30	32.13	33.05	34.01
0%	1.00	1.00	1.00	1.00	1.00	1.00
25%	3.75	4.00	8.75	2.75	4.00	6.00
50%	17.00	19.00	27.00	19.00	24.50	29.00
75%	46.00	49.00	48.00	48.75	49.00	52.50
100%	175.00	127.00	186.00	163.00	166.00	127.00
Standard Deviation	33.66	29.54	32.78	36.74	34.50	31.01
Variance	1132.98	872.60	1074.70	1349.59	1189.91	961.40
Skewness	1.81	0.92	1.63	1.55	1.55	0.96
Range	174.00	126.00	185.00	162.00	165.00	126.00
Interquartile Range	42.25	45.00	39.25	46.00	45.00	46.50

Abstract

In disaster management, decisions have to be fair in order to be accepted by the majority of the population. In this work, shelters are built before and after a disaster occurred and the locations of these shelters are chosen so that fairness is maximized. The fairness of a possible solution is evaluated by two different approaches, ex-ante and ex-post, that are visualized using a two-stage decision tree. Small problem instances are generated, first solved by complete enumeration, and then compared with the results of a genetic algorithm, while large problem instances are only solved by the genetic algorithm due to otherwise excessive computation times. In addition, the solutions of the ex-ante/ex-post objective function are plugged into the ex-post/ex-ante objective function and the percentage differences are compared. Afterwards, the ex-ante and ex-post solutions are plugged into the utilitarian objective function and compared with the utilitarian solution.

Zusammenfassung

Im Katastrophenmanagement müssen Entscheidungen fair sein, um von der Mehrheit der Bevölkerung akzeptiert zu werden. In dieser Arbeit werden Schutzräume vor und nach einer Katastrophe gebaut und die Standorte dieser Schutzräume werden so gewählt, dass die Fairness maximiert wird. Die Fairness einer möglichen Lösung wird durch zwei verschiedene Ansätze bewertet, Ex-ante und Ex-post, die anhand eines zweistufigen Entscheidungsbaums visualisiert werden. Kleine Probleminstanzen werden generiert, zuerst durch vollständige Enumeration gelöst und dann mit den Ergebnissen eines genetischen Algorithmus verglichen, während große Probleminstanzen aufgrund ansonsten übermäßiger Rechenzeiten nur durch den genetischen Algorithmus gelöst werden. Zusätzlich werden die Lösungen der Ex-ante/Ex-post Zielfunktion in die Ex-post/Ex-ante Zielfunktion eingesetzt und die prozentualen Unterschiede verglichen. Anschließend werden die Ex-ante und Ex-post Lösungen in die utilitaristische Zielfunktion eingesetzt und mit der utilitaristischen Lösung verglichen.