Real Time City Evacuation Planning using Transfer Learning

A Thesis

Presented in Partial Fulfillment of the Requirements for the

Degree of Master of Science

with a

Major in Computer Science

in the

College of Graduate Studies

University of Idaho

by

Madhav Pandey

Major Professor: Robert B. Heckendorn, Ph.D.

Committee Members: Terence Soule, Ph.D.; Ahmed Abdel-Rahim, Ph.D.

Department Administrator: Terence Soule, Ph.D.

December 2018

Authorization to Submit Thesis

This thesis of Madhav Pandey, submitted for the degree of Master of Science with a major in Computer Science and titled "Real Time City Evacuation Planning using Transfer Learning," has been reviewed in final form. Permission, as indicated by the signatures and dates given below, is now granted to submit final copies to the College of Graduate Studies for approval.

Major Professor: _____Date_____
Robert B. Heckendorn, Ph.D.

Committee Members: _____Date_____
Terence Soule, Ph.D.

_____Date_____
Ahmed Abdel-Rahim, Ph.D.

Department Administrator: _____Date_____
Terence Soule, Ph.D.

Abstract

The aim of this research is to produce real time city evacuation planning during a disaster which can adapt to changing conditions. Our goal is to save as many lives as possible in the disaster. In this research, we mainly target three things. First, use of population-based evolutionary algorithms to solve city evacuation planning. Second, how past knowledge can be used in the city evacuation for generating adaptive solutions to the changing conditions of a disaster such as bridges collapse, change in safe areas or people not following instructions. This technique of using previous knowledge to solve a different but related problem is called transfer learning. Third, determine the influence of population diversity in transfer learning with evolutionary algorithms.

Evolutionary algorithms are iterative, they start with initial population (set of candidate solutions) and improve those solutions in each iteration. We have used Evolution Strategy (ES) as our evolutionary approach combined with probability based representation. The probability of road being chosen at the intersection is a gene in the problem representation. The list of all the road probabilities in the city form a candidate solution. Each candidate solution in a population gets evolved according to a fitness function which is the total safety value of all people based on where they are located. We perform repeated optimizations based on the current disaster conditions using previous populations as a starting population for reoptimization. We see how diverse solutions can enhance transfer learning.

From all the experiments performed, we found that evolutionary algorithms can solve the city evacuation problem very effectively under the changing conditions of a disaster. When transfer learning is used in evolutionary algorithms, they are helpful to produce better result than just starting with random initial solutions for reoptimization. Also, transfer learning with previous diverse population using Deterministic Crowding (DC) to increase population diversity gave better results than previous population using ES.

Acknowledgments

I would like to acknowledge the inspiration and guidance of Dr. Robert B. Heckendorn. This research would not have been possible without his regular advice and important guidelines throught the research. I would like to thank Dr. Terence Soule and Dr. Ahmed Abdel-Rahim for accepting my request to be in the committee and helping me with important suggestions. I would like to express special thanks to Keith J. Drew specially for some logic in code base and Homaja Pydi Kumar Marisetty for providing a tool to draw the topology of a city. Finally, I also recognize the help of faculty and staff of the Computer Science Department at the University of Idaho for supporting me directly or indirectly.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1 Problem Description

Natural or artificial disasters like tsunamis, hurricanes, volcanic eruptions, floods, fires, terrorist attacks, industrial accidents, etc., are responsible for thousands of deaths every year. Many lives could be saved by early warning and efficient evacuation planning. Toward that end we propose a real time evacuation planning system for cities. Our system treats a city as a graph with nodes and edges quantitatively tagged with their level of "safety". We then optimize the safety of all people in the city based on their location. Our system is designed to suggest to people as they arrive at each intersection which road they should take next in order to optimize overall safety. The system must be quick enough to re-optimize the whole city in order to account for the unfolding conditions of the disaster, for example, bridges collapse, safe areas becoming unsafe, roads becoming blocked by debris and people not following instructions.

We have chosen an Evolutionary Computation approach to solve the problem that meets the needs above of speed, effectiveness and robustness to changing conditions. In the process of creating this software we addressed the following research questions:

Q1 A real time solution to an evacuation problem requires adapting to changing conditions in a disaster. Can an iterative evolutionary algorithm solve the real time evacuation problem in an efficient and robust way?

Q2 Evolutionary algorithms maintain a population of potential solutions. Can having a population of solutions from previous optimization provide a computational advantage in re-optimizing a solution under the changing conditions of a disaster?

Transfer learning is a technique that stores knowledge while solving one problem and applying that knowledge to solve a different but related problem. In other words, this question is, can we apply transfer learning in evolutionary algorithms?

Q3 What is the influence of population diversity on the effectiveness of transfer learning?

1.2 Approach

Our approach is diagrammed in Figure 1.1. When a disaster occurs, we repeatedly apply evolutionary algorithms to replan evacuation routes to optimize safety of evacuees based on the current conditions. The optimized solutions are applied in real time to the evacuees. This means when a problem occurs such as bridge collapses, safe area changes or people are not following instructions, we can repeat the optimization using the previous optimized solution as a starting point for evolving new solutions to the changed conditions. Currently, re-optimization is not automatically triggered whenever a new problem occurs during disaster; we simply re-optimize when our previous optimization is completed. This technique of using previous result/knowledge to solve similar problems is transfer learning.

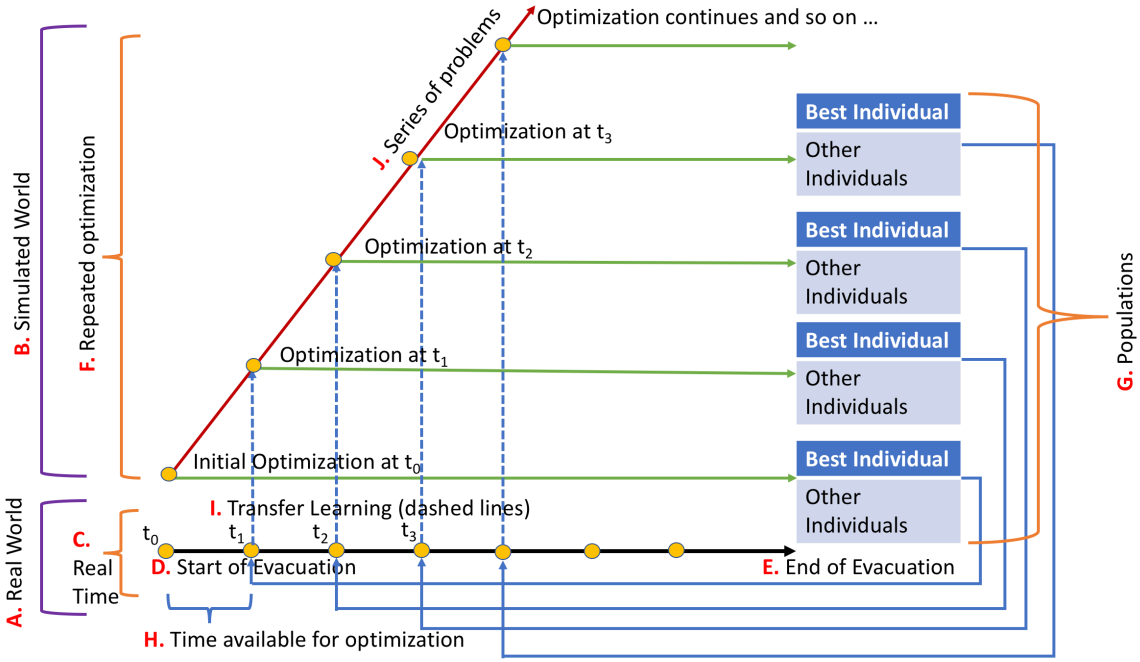


Figure 1.1: A diagram of our approach for city evacuation using evolutionary algorithms and transfer learning. The dashed line indicates when the information from the population in the previous optimization at time t_k is brought forward to solve the new problem defined by the real state of the evacuation and city at t_{k+1}.

In Figure 1.1 time runs along the X-axis. In the real world, as labeled on the Y-axes, time is "clock time". At times t_0, t_1, ..., t_n, the optimization process begins given the current conditions of the real disaster. These are things that could not have been known at the beginning of the disaster such as a bridge collapses or safe area changes. When a simulation is done at time t_k it begins all over again at time t_{k+1}. In the simulated world, each optimization relies on repeated simulated evacuations running until the end of the evacuation plan or everyone is optimally safe. The repeated simulations are indicated by the lines labeled "Simulated Worlds" along the Y-axes.

Here are the list of labels found in Figure 1.1.

A: Real World - The real world is the actual unfolding evacuation of the city during the real disaster.

B: Simulated World - This is the simulated world where our application simulates the traffic and tries to optimize safety.

C: Real Time - This is "wall clock" time, the time before, during, and after a disaster in real world. The length of this is the time we have to complete the evacuation.

D: Start of Evacuation - The beginning of the city evacuation.

E: End of Evacuation - This is the time when disaster ends or evacuation completes.

F: Repeated Optimization – A new optimization is begun as the old optimization completes. Optimizations continue until the evacuation is over.

G: Population - Set of candidate solutions generated after the completion of each evolutionary optimization. In the figure each optimization produces an adapted population of high performing solutions.

H: Time available for a single optimization - The time interval for performing an individual optimization. In particular, H indicates the "wall clock" time interval of the

first optimization, but also signifies the time of any optimization during the evacuation. At each time step, t_k, we start an optimization with the current city configuration and the population from the last optimization. The people in the city use the best solution found by the last optimization to continue their evacuation.

I: Transfer learning - The technique of using population from previous optimization and feeding to new optimization is an example of transfer learning. A major goal of this project is to determine whether a transfer learning approach lets evolution find solutions more rapidly, thereby reducing the time interval H.

J: Series of problems - A disaster evacuation unfolds unpredictably. Dynamic problems that may occur during evacuation create a series of similar but different optimization problems to be solved. We sample the city at the times t_k, and that produces a series of problems with similar but different parameters and less and less time to the end of the evacuation.

Evolutionary algorithms are excellent for producing practical solutions to very complex problems, which may or may not be globally optimal. Evolutionary Computation is a subfield of Stochastic Optimization. It is inspired by Darwinian principles of biological evolution. Evolutionary algorithms are population-based trial and error problem solvers. At first, they have set of candidate solutions called a **population**. These solutions get updated iteratively over generations. In each new **generation**, new solutions are generated with small changes from older solutions. Each solution is evaluated for an ordered measure of quality called **fitness**. If these newly produced solutions are better, in terms of fitness, than some older solutions, the new solutions will replace the older less desired solutions. With each new generation the population fitness tends to improve. For more detail on Evolutionary Computation see Section 2.2. As an evolutionary algorithm, we are using **Evolution Strategy (ES)**. We have chosen ES because it is fast, a good optimizer for complex multidimensional real-value optimization and uses strategic parameters such as mutability parameters. The

fitness function of a proposed solution is based on the safety of all evacuees at the end of a simulated evacuation.

We have used Evolutionary Computation to find answers to our research questions mentioned in Section 1.1. Our first question Q1 is can an evolutionary algorithm provide practical solutions to real time evacuation of cities? We implemented an ES version of the evacuation planning software as envisioned in Figure 1.1 and tested it on a set of test cases we have developed and different disaster scenarios based on the topology of the city of Boise, Idaho. Our second question Q2 is can the population-based exploration of the solution space yield a faster convergence to practical solutions for unexpected changes in unfolding disasters? If so, how can we enhance this effect? To explore this we tested several techniques of priming the population for reoptimization based on the population at the end of the last optimization. Various experiments are performed to see what, if any, part of optimizations at time t_k can be used to solve the disaster at t_{k+1}. This concept of reusing previous knowledge is **transfer learning**. We see if we can use transfer learning in an evolutionary algorithm's population-based search.

To further explore the effectiveness of population-based transfer learning we studied how diversity of solutions in the population affect the speed and quality of solutions presented by the evolutionary algorithm. For improving the diversity in the population, we use **Deterministic Crowding (DC)** with steady state approach, where steady state means two best solutions out of two parents and two children are added back in the population, and worst two are discarded making the population size constant. We retested the effectiveness of the algorithm on the test suite.

1.3 Overview

Chapter 2 presents the background of the problem and our approach. Chapter 2 also reviews the literature in evacuation planning systems developed so far for evacuating building, area, region, etc. Finally, we will formalize the problem of city evacuation defining the scope and

terminology of the problem. In Chapter 3, we discuss the detail of safety optimization and city evacuation simulation. Chapter 4 shows the experiments conducted in answering the research questions. Chapter 5 concludes with a summary of the research and the future work.

Chapter 2: Background

2.1 Evolution

Evolution is a process through which genetic characteristics (genotype) of organisms change over successive generations by the process of natural selection. Genetic changes made by evolution affect the physical characteristics (phenotype). Individuals with characteristics best adapted to their environment by avoiding predators, resisting diseases, finding foods or adjusting in the climatic changes are considered to be the best fit for survival and are more likely to reproduce and pass their genes to their offspring. In general, the fitness of a genotype is the quality that measures how successful the phenotype is in living and reproducing. We can see a lot of examples in biology that shows evolution has supported the adaptation of creatures to their environment. For example, finches in Galapagos Islands have developed different shaped beaks to use the different types of food found in their respective islands. One can think of evolution as an optimization algorithm trying to select the genotype with the best fitness. This idea is the basis of Evolutionary Computation.

Biological genomes may have a memory of useful genes used to solve past survival stresses. For example, suppose a mouse population once in the past survived a drought. Their genes may have mutated to resist the drought. Useful traces of those genes may reside in the mouse genome today although inactive and perhaps slightly mutated. If drought appears in the future, the gene may be quickly mobilized from its inactive state by mutation and selective pressure. We imagine the same thing could happen in an artificial population in which some individuals understand variations in successful solutions of the survival problem. In our case, we imagine it is possible that in the process of learning an evacuation, useful "tools" reside in the genotypes of the whole population that might help to solve a related problem. This would suggest that perhaps evolutionary populations might aid in transfer learning.

Diversity plays an important role in evolution. A diverse population provides fuel for variation. We will examine diversity's relation to transfer learning in the context of contingent genes in the population.

2.2 Evolutionary Computation

Evolutionary algorithms simulate biological evolution in a computer. There are different types of evolutionary algorithms like Evolutionary Programming, Genetic Algorithms, Evolution Strategies, Genetic Programming, etc. Evolutionary algorithms can be used as efficient problem solvers for optimization, constraint handling, machine learning and modeling tasks [Eiben and Schoenauer, 2002]. The main elements in all evolutionary algorithms are representation, population, fitness, parent selection, crossover, mutation, survival selection and termination condition.

Representation - This denotes the data representation for the genotype. It can be thought of as the **genome** that represents an individual. Here, **individual** is a candidate solution. This could be binary values, integers, real valued numbers, graphs, or whatever completely defines a solution that is sought after.

Population - Group of candidate solutions form a population. An individual in a population means one candidate solution of that population. In any type of evolutionary algorithm, a population consists of initial candidate solutions which will get updated with small genetic change in every generation.

Fitness - Fitness represents the quality of the candidate solution. Fitness defines the standard by which we evaluate the quality of a potential solution. In evolutionary algorithms this will be tied to selective pressure. The average fitness of the population of individuals tends to improve with time under selection thereby giving us better and better solutions.

Parent Selection - We select individuals from the population to be parents. Parents are used in mixing operators that blend and mutate previous solutions to get new solutions whose fitness we hope is correlated with the parent's fitness. Selection methods could be Tournament Selection, Fitness Proportional Selection, Rank Proportional Selection, etc. These parents go through mixing operators such as crossover and mutation to generate offspring. Offspring are new individuals (candidate solutions) whose genes are a combination of genes from their parents.

Crossover - A crossover operator combines the genes from the parents to form new offspring. So, an offspring has a mixture of genes from its parents. There are different methods to perform crossover such as one point crossover, two point crossover, uniform crossover, etc.

Mutation - A mutation operator changes one or more gene values in a chromosome. Mutation assumes locality in that a small change in the genotype will make a small change in the phenotype and then hopefully a small change in fitness.

Replacement Selection - This is the process of replacing individuals of lower fitness in population with newly generated higher fitness individuals. The replacement process helps to get better candidate solutions in every successive generation.

Termination - Termination is the stopping condition of the evolutionary algorithm. The termination condition is reached when a satisfactory solution is generated, or a time limit is exceeded.

2.3 Transfer learning

Transfer learning occurs when knowledge gained while solving one problem is reused to solve a different but similar problem. For example, knowledge gained while learning to recognize cats could apply when trying to recognize dogs. Details on transfer learning can

be found in [Pan and Yang, 2010]. Transfer learning technique has been mainly used in small scale applications like sensor-network-based localization, text classification or image classification problems [Pan and Yang, 2010]. But in the future, this technique may be used to solve challenging applications like video classification, social network analysis and logical inference.

2.4 Evacuation Literature

Emergency evacuation is very important in case of imminent threats, so a lot of work has been done in evacuation planning [Abdelgawad and Abdulhai, 2009]. Evacuation may vary from small area evacuation like clearing a building and evacuating an office area to large area evacuation like a city and regional evacuations. Different models have been developed for evacuation. Evacuation of building by [Chalmet et al., 1982] talks about how to evacuate large buildings with many occupants in minimum time. [Pelechano and Badler, 2006] uses trained agents as leaders for building evacuation. [Coutinho-Rodrigues et al., 2016] studied pedestrian evacuation in densely urbanized city centers common in old European cities. [Lu et al., 2005] talks about Capacity Constrained Routing Planner (CCRP) for evacuation. This presents a heuristic algorithm to produce a sub-optimal solution for an evacuation planning problem in a large size network. This focuses on shortest path (earliest destination arrival time) from any source node to any destination node. [Yin, 2009] states that CCRP by [Lu et al., 2005] is not scalable to large network and has proposed CCRP++ which is scalable to large city with large numbers of evacuees and huge size of transportation network. [Yin, 2009] mentions that they reuse search results in previous iterations and avoid repetitive global shortest path expansion in CCRP. That means, they expand from one source in each iteration and reuse this shortest path in the future iterations. Many other papers also present different extensions and modifications of CCRP. [Pel et al., 2010] has proposed a different approach which is based on traveler information and compliance behavior for modeling evacuations. Different multi-objective approaches to evacuation planning have been put

forward by [Saadatseresht et al., 2009], [Yuan and Han, 2010], [Stepanov and Smith, 2009]. [Saadatseresht et al., 2009] and [Yuan and Han, 2010] use an evolutionary approach while [Stepanov and Smith, 2009] uses an integer programming (IP) formulation for optimal route assignment.

Our lab has used Evolution Strategies with probability based approach and safety maximization fitness [Drew et al., 2017]. We have extended this work with high speed simulation in the fitness function and Deterministic Crowding to control diversity. Our approach is different from others in the way we use Deterministic Crowding (DC) algorithm to get diverse solutions; then we use those diverse solutions as a starting population for Evolution Strategy (ES) to get fast and robust solution even in dynamic challenges of a real disaster such as traffic congestion, sudden change in location of safe area, bridge/road collapses or people not following instructions. So, our approach is to transfer learning/knowledge from previous optimization of the city evacuation to solve new problems in the same city.

Chapter 3: Algorithms

In this Chapter, we describe the evolutionary algorithms, the traffic simulation algorithm and the data structures used in our program. The evolutionary algorithms are Evolution Strategy (ES) and Deterministic Crowding (DC) from [Eiben and Smith, 2015]. The main evolutionary cycle of any evolutionary computation algorithm is shown in Figure 3.1.

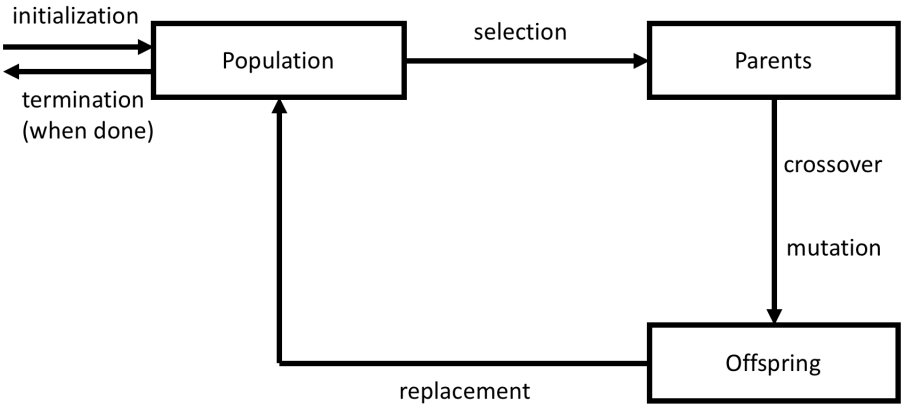


Figure 3.1: Evolutionary Cycle.

Before talking about the algorithms, we have to understand the topology, fitness, chromosome and SLang.

3.1 Topology

Topology describes the transportation network for the city. It consists of nodes, edges and agents. It includes capacity, speed and safety parameters. A node is a decision point in the network, usually an intersection. One node is connected to another node with an edge which in our case represents one direction of a road. Figure 3.2 shows a sample topology of a simplistic city to explain nodes and edges. Circles in a figure are nodes. The numbers in the circles are node IDs. A line connecting two nodes is an edge with arrow in it showing the

direction. Color of nodes in the figure represents the degree of safety from red for maximally dangerous to blue for minimally dangerous. Agents are vehicles which carry people along the road. One would expect with a good evacuation plan all agents would arrive at the bottom right corner at the end of a simulation for the city shown in Figure 3.2.

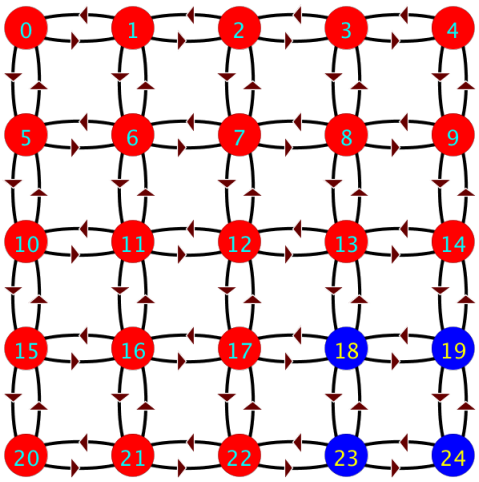


Figure 3.2: Topology of a city with nodes and edges. Agents are distributed on nodes.

3.1.1 City

City is the complete unit in the topology. City consists of nodes, edges and agents. City has information about the name of city, number of nodes contained in the city and maximum outgoing degree of node.

3.1.2 Node

Node is most likely an intersection of roads. Outgoing roads go out from the node and incoming roads come into the node. Node has its ID, safety, longitude, latitude, capacity and wait time. Node also has the information about which edges are going out of it. Each node has one self-edge, which goes out of that node and comes into the same node. This

self-edge is used to implement waiting at a node.

3.1.3 Edge

Edge is a road that connects two nodes with each other. The road is bidirectional if it is not a one way road. These are represented by two edges connecting two nodes in opposite directions. For example, we have node A and node B, then there are two edges: AB going from A to B and BA going from B to A. An edge has an ID, node from which it starts and the node to which it ends. An edge has a limited capacity of vehicles it can hold. Each edge has it's own free-flow time. Free flow is the time required to travel through the edge under ideal conditions meaning no other traffic or problems on the road. Finally, an edge knows how many agents are on the edge we'll call congestion. As the traffic increases in the edge, the travel time of that edge begins to increase. The travel time of the edge is calculated by using the congestion (or volume-delay, or link performance) function developed by Bureau of Public Roads (BPR) in 1964 [Bureau of Public Roads, 1964] formulated with:

$$time_e = \mathcal{F}_e \left(1 + \alpha \left(\frac{v_e}{C_e}\right)^{\beta}\right) \tag{3.1}$$

where,

$time_e$ = congestion flow time on edge e,

$\mathcal{F}_e$ = free flow travel time on edge e,

v_e = volume of traffic on edge e per unit of time (flow attempting to use edge e),

C_e = maximum capacity of edge e per unit of time,

α and β = traffic/delay parameters

[Manzo et al., 2013] suggests that the values for α and β vary with characteristics of network object. The value for α is 0.15 and β is 4.0 in the original BPR formula. As we follow the original BPR formula, we use these values in our experiments.

3.1.4 Agents and Individuals

An agent is a vehicle that carries people from one node to another through an edge, or it can represent an arbitrary clump of people who will evacuate together. In the first case it will ultimately allow us to consider both bus and bicycle as different capacity vehicles. In the second case, it speeds optimization. Agents have member counts which show how many individuals it carries. This is currently fixed for all agents; we use 10 in our experiments. Agent's safety is determined by where it is located. If the agent is at the node, it's safety value is equal to the safety value of that node, and if the agent is on the edge, it's safety value is equal to the safety value of that edge.

3.2 Fitness

The fitness function calculates the safety value of all people depending upon where the agents are currently located. If they are at the node, their safety value is equal to the safety value of that node, and if they are on the edge, their safety value is equal to the safety value of that edge. We calculate the fitness value in the range of $[0, 1]$. 0 means no one survived in the disaster, and 1 means all people survived in the disaster.

For a total N agents, if s_i is the safety value of i^{th} agent, then fitness F at time t is given by the mean safety of agents as:

$$F_t = \frac{1}{N} \sum_{i=1}^{N} s_i \tag{3.2}$$

3.3 Chromosome

The chromosome is the solution representation. It must describe how agents move through the topology during the evacuation. We use a probability based approach. We represent the evacuation plan as a list of probabilities. For each node N and each road R leaving N there is a probability $p_{N,R} \in [0, 1]$ which is the probability that an agent at N will leave by R.

One road leaving the node returns immediately to the node after a small wait. This is used for waiting at a node and is called "self-edge". The list of all probabilities is the chromosome for the evolutionary algorithm. For each node N:

$$\sum_{r \in \text{out}(N)} p_{N,r} = 1 \tag{3.3}$$

Here is an actual listing of some probabilities:

```
0.278030 0.503819 0.218151
0.658674 0.236544 0.104782 0.000000
0.021569 0.332886 0.256236 0.295159 0.094150
0.134159 0.454595 0.403816 0.007430
0.000000 0.000000 1.000000
0.051990 0.091479 0.437318 0.147472 0.271741
0.490013 0.176637 0.088273 0.245077
0.049689 0.201919 0.377348 0.142510 0.228534
0.642855 0.138774 0.164844 0.053527
```

Here, the sample has 9 rows; that means it has 9 nodes (intersections) in the city. The values are the probabilities of edges (roads) being used in each node. The sum of all probabilities at a node is 1.0. The number of probabilities in a row corresponds to the number of edges coming from the node, which in turn is the number of roads leaving the node plus one self-edge.

3.4 SLang

SLang (Simulation Language) is a experiment description language that allows for the specification of problems and parameters for experiments [Drew et al., 2017].

The input file to the software contains SLang commands, which has information about the city like edges, nodes and agents. The file contains evolution parameters like population size, child population size, mutation probability, crossover probability, etc. SLang also contains the safety of the node and the edge. Agent distribution is also specified in SLang. Below is the sample SLang for a simple city.

```
// desc: Simple test with 2 safe nodes
// city: name, totalNodes, maxDegree
city testcity 9 5
// parameters: ES+, sigma, hours, pop/childpop, t_size, xover, mutate
```

```
parameters ES+ 0.1 0.5 100 100 3 0.2 0.5
// edge: src, dst, maxCapacity, freeflow, Alpha, Beta, safety
edge 0 1 1000.0 0.004 0.15 4.00 0.0
edge 0 7 1000.0 0.008 0.15 4.00 0.0
edge 1 0 1000.0 0.004 0.15 4.00 0.0
edge 1 2 1000.0 0.008 0.15 4.00 0.0
edge 1 7 1000.0 0.011 0.15 4.00 0.0
edge 2 1 1000.0 0.008 0.15 4.00 0.0
edge 2 3 1000.0 0.007 0.15 4.00 0.0
edge 2 5 1000.0 0.004 0.15 4.00 1.0
edge 2 8 1000.0 0.002 0.15 4.00 0.0
edge 3 2 1000.0 0.007 0.15 4.00 0.0
edge 3 4 1000.0 0.009 0.15 4.00 1.0
edge 3 5 1000.0 0.014 0.15 4.00 1.0
edge 4 3 1000.0 0.009 0.15 4.00 0.0
edge 4 5 1000.0 0.010 0.15 4.00 1.0
edge 5 2 1000.0 0.004 0.15 4.00 0.0
edge 5 3 1000.0 0.014 0.15 4.00 0.0
edge 5 4 1000.0 0.010 0.15 4.00 1.0
edge 5 6 1000.0 0.002 0.15 4.00 0.0
edge 6 5 1000.0 0.002 0.15 4.00 1.0
edge 6 7 1000.0 0.001 0.15 4.00 0.0
edge 6 8 1000.0 0.006 0.15 4.00 0.0
edge 7 0 1000.0 0.008 0.15 4.00 0.0
edge 7 1 1000.0 0.011 0.15 4.00 0.0
edge 7 6 1000.0 0.001 0.15 4.00 0.0
edge 7 8 1000.0 0.007 0.15 4.00 0.0
edge 8 2 1000.0 0.002 0.15 4.00 0.0
edge 8 6 1000.0 0.006 0.15 4.00 0.0
edge 8 7 1000.0 0.007 0.15 4.00 0.0
// node: id, capacity, wait time, safety, x coord, y coord
node 0 1 0.016 0.0 0.0 1.0
node 1 1 0.016 0.0 1.0 0.0
node 2 1 0.016 0.0 2.0 0.0
node 3 1 0.016 0.0 3.0 0.0
node 4 1 0.016 1.0 4.0 1.0
node 5 1 0.016 1.0 3.0 2.0
node 6 1 0.016 0.0 2.0 2.0
node 7 1 0.016 0.0 1.0 2.0
node 8 1 0.016 0.0 2.0 1.0
// agent: start node, number of agents, number of individuals per agent
agent 0 100 10
agent 1 100 10
show topology
// run: algorithm type, generations, runs
run 1 100 10
show parameters
```

In the SLang, the line starting with `city` has information about the city name and the number of nodes, followed by maximum out degree of node. The maximum out degree of node indicates the greatest number of edges that are going out from any node.

The line starting with `parameters` includes evolution strategy type "," or "+", sigma value, simulation hour, population size, child population size, tournament size, crossover probability and mutation probability respectively. Simulation hour is the time in hours; we have this much of time available for evacuating the city. Simulation hour 0.5 means we have

half an hour to evacuate the city.

The line starting with `edge` includes source node, destination node, maximum capacity, free flow time, alpha coefficient, beta coefficient and safety respectively.

Line starting with `node` includes node ID, node capacity, wait time, safety, longitude and latitude respectively.

Line starting with `agent` shows the distribution of agents along the nodes. `agent` is followed by node ID where the agents currently are at the start of simulation, number of agents and number of individuals per agent.

In the line starting with `run`, the first integer shows which algorithm to run: 0 means estimate proper simulation hour required to evacuate the city with fitness value greater than 0.98 (meaning more than 98% lives being saved), 1 means run ES (Evolution Strategy) algorithm and 2 means run DC (Deterministic Crowding) algorithm. Second integer value denotes the number of generations to run, and third integer shows how many runs to carry out. Here, runs mean how many repetitions to perform for the single optimization. These repetitions are used for calculating mean and standard deviation of fitness.

3.5 Evolution Strategy

Evolution Strategy (ES) was invented in 1990s by Rechenberg [Rechenberg, 1973] and Schwefel [Schwefel, 1995]. There are two types of ES represented by $(\mu+\lambda)$ and (μ, λ). μ individuals are present in the population and λ offspring are generated in each generation. In $(\mu + \lambda)$ type the best μ individuals are selected from the combined set of offspring and parents, and that becomes the population for the next generation. In (μ, λ) type the best μ individuals are selected from the set of offspring only. In all cases, "best" means higher fitness. When $\mu = \lambda$ then the new population is just the set of offspring. Note that the comma operator does not naturally preserve the best individual from generation to generation.

Algorithm 1 and Figure 3.3 describe the Evolution Strategy. The term σ in Algorithm 1 is called mutation step size. Our experiments mostly use $(\mu + \lambda)$ type of ES.

Algorithm 1 $(\mu/\sigma \overset{+}{,} \lambda)$-Evolution Strategy

1: $//\mu = populationCount,\ \lambda = offspringCount$
2: $population \leftarrow$ initialize();
3: **for** $generation \leftarrow 1 \quad to \quad MAX$ **do**
4: **for** $i \leftarrow 0 \quad to \quad \mu/2$ **do**
5: $parent_1 \leftarrow$ tournamentSelection($population$)
6: $parent_2 \leftarrow$ tournamentSelection($population$)
7: $(child_1, child_2) \leftarrow crossover(parent_1, parent_2)$
8: $child_1 \leftarrow$ mutate($child_1$)
9: $child_2 \leftarrow$ mutate($child_2$)
10: *Add* $child_1,\ child_2$ *to* $childPopulation$
11: **if** $ES+$ *operator* **then**
12: $population \leftarrow$ bestPopulation(μ, sortWithFitness($population, childPopulation$))
13: **else if** $ES,$ *operator* **then**
14: $population \leftarrow$ bestPopulation(μ, sortWithFitness($childPopulation$))

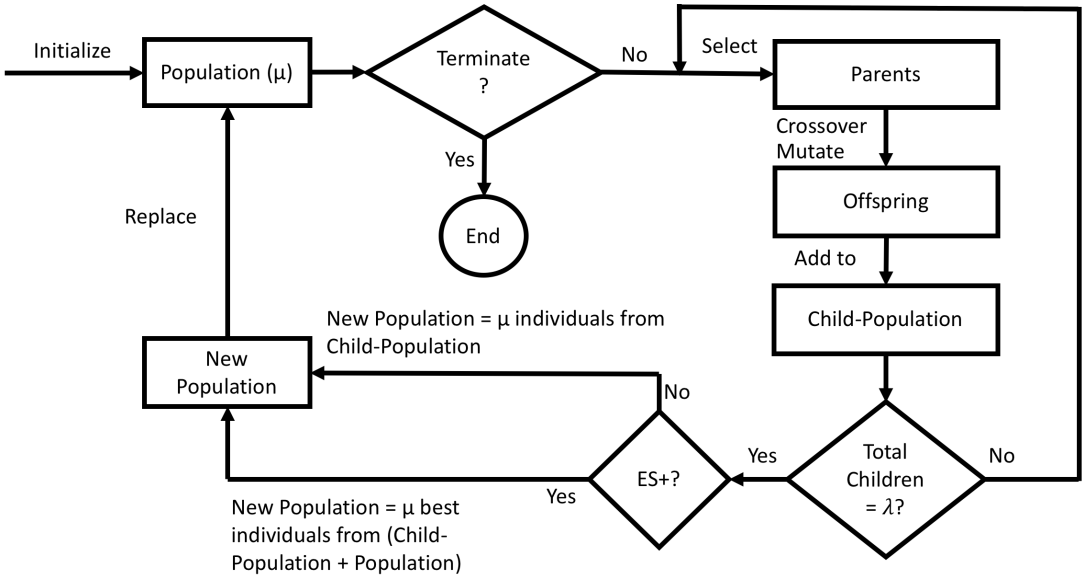


Figure 3.3: Evolution Strategy.

3.6 Deterministic Crowding

The crowding algorithm was originally suggested by De Jong's thesis [De Jong, 1975] as a way to preserve diversity where new individuals replace the similar individuals in population as mentioned in [Eiben and Smith, 2015]. Later, Mahfoud suggested an improvement on this algorithm and named it as Deterministic Crowding [Mahfoud, 1992]. This algorithm assumes that offspring are more similar to their parents than other individuals, so the offspring replace the parents.

Algorithm 2 and Figure 3.4 explain the Deterministic Crowding. The term μ in Algorithm 2 and Figure 3.4 is the number of individuals in the population.

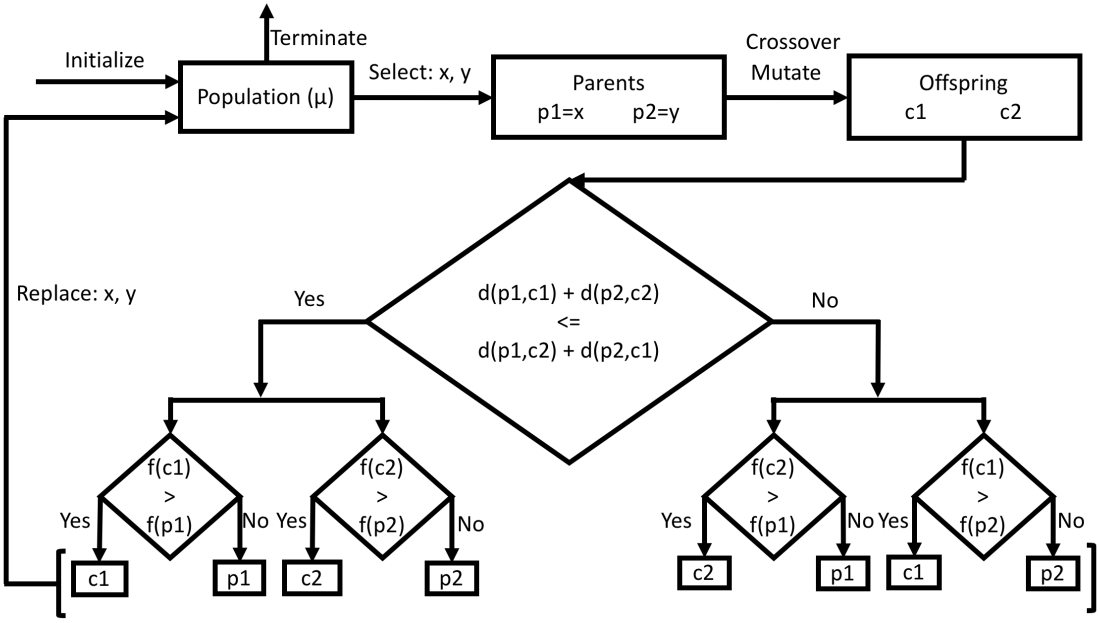


Figure 3.4: Deterministic Crowding.

Algorithm 2 Deterministic Crowding

1: $//\mu = individuals \quad in \quad population$
2: $population \leftarrow$ initialize();
3: **for** $generation \leftarrow 1 \quad to \quad MAX \quad$ **do**
4: $\quad population \leftarrow$ shuffle($population$)
5: $\quad$ **for** $i \leftarrow 0 \quad to \quad \mu/2 \quad$ **do**
6: $\quad\quad parent_1 \leftarrow population_{2i}$
7: $\quad\quad parent_2 \leftarrow population_{2i+1}$
8: $\quad\quad (child_1, child_2) \leftarrow$ crossover($parent_1, parent_2$)
9: $\quad\quad child_1 \leftarrow$ mutate($child_1$)
10: $\quad\quad child_2 \leftarrow$ mutate($child_2$)
11: $\quad\quad$ **if** [distance($parent_1, child_1$) + distance($parent_2, child_2$)] <= [distance($parent_1, child_2$) + distance($parent_2, child_1$)] **then**
12: $\quad\quad\quad$ **if** fitness($child_1$) > fitness($parent_1$) **then**
13: $\quad\quad\quad\quad population_{2i} \leftarrow child_1$
14: $\quad\quad\quad$ **if** fitness($child_2$) > fitness($parent_2$) **then**
15: $\quad\quad\quad\quad population_{2i+1} \leftarrow child_2$
16: $\quad\quad$ **else**
17: $\quad\quad\quad$ **if** fitness($child_2$) > fitness($parent_1$) **then**
18: $\quad\quad\quad\quad population_{2i} \leftarrow child_2$
19: $\quad\quad\quad$ **if** fitness($child_1$) > fitness($parent_2$) **then**
20: $\quad\quad\quad\quad population_{2i+1} \leftarrow child_1$

3.7 Traffic Simulation

A city wide traffic simulation is used to compute the fitness. The best chromosome will give probabilities for the intersections that will best get the agents out of the city in the allotted time. Note that the best for a specific city situation may not be the best at a later point in the evacuation. The first reason being that later optimizations are occurring after the agents have moved, but also that conditions in the disaster may have changed.

Algorithm 3 sketches the simulation algorithm. Initially agents are located at their source nodes and placed in a high speed priority queue called a Calendar Queue. The agents are prioritized in this queue by arrival time at the next node. The algorithm selects the next agent that requires action using the queue. The appropriate edge is determined for that agent to travel on using the probabilities taken from the chromosome. The time to travel through that edge is calculated based on factors such as capacity of edge and traffic. The travel time is added to the current time and used to compute next arrival time. At the start, the next arrival time for all agents is 0. If this next arrival time is less than or equal to the total allowed simulation time, $maxTime$, it is placed back to queue. Otherwise it is not placed back to calendar queue since it is beyond the time of consideration. It's safety is still the safety of the edge it is on however. This process continues until all agents' next arrival time exceeds allowed simulation time, making the calendar queue empty.

The simulation runs until the simulation time reaches the $maxTime$. Figure 3.5 diagrams how the simulation happens, and when the fitness (total safety of all agents) is calculated. The next arrival time for the agent in the figure is cumulative. The next arrival time for the agent starts with the value 0 and adds the edge travel time as the agent passes through that edge or the waiting time the agent waits at the node. Maximum value of the next arrival time is equal to the $maxTime$.

Algorithm 3 Traffic Simulation

```
1: while calendarQueue.notEmpty() do
2:     currentAgent ← calendarQueue.dequeue()
3:     currentNode ← currentAgent.getToNode()
4:     randomProbability ← randomValue(0, 1)
5:     for all edge in edgesOfCurrentNode do
6:         if randomProbability <= edge.getProbability() then
7:             nextEdge ← edge
8:             nextEdgeTotalTraffic ← nextEdge.getTraffic() + currentAgent.getMembers()
9:             if nextEdgeTotalTraffic > nextEdge.getMaxCapacity() then
10:                nextEdge ← NULL
11:            break;
12:    if currentAgent.getPreviousEdge() ≠ NULL then
13:        currentAgent.previousEdge.traffic ← currentAgent.getPreviousEdge().getTraffic()− currentAgent.getMembers()
14:    if nextEdge ≠ NULL then
15:        nextEdge.traffic ← nextEdgeTotalTraffic
16:        currentAgent.edge ← nextEdge
17:        currentAgent.toNode ← nextEdge.getToNode()
18:        currentAgent.nextArrivalTime ← currentAgent.getNextArrivalTime() + nextEdge.getTravelTime()
19:    else
20:        currentAgent.edge ← nextEdge
21:        currentAgent.toNode ← currentNode
22:        currentAgent.nextArrivalTime ← currentAgent.getNextArrivalTime() + currentNode.getWaitTime()
23:    if currentAgent.getNextArrivalTime() <= maxTime then
24:        calendarQueue.enqueue(currentAgent)
25: totalSafety ← calcTotalSafetyOfAllAgents()/totalAgents
```

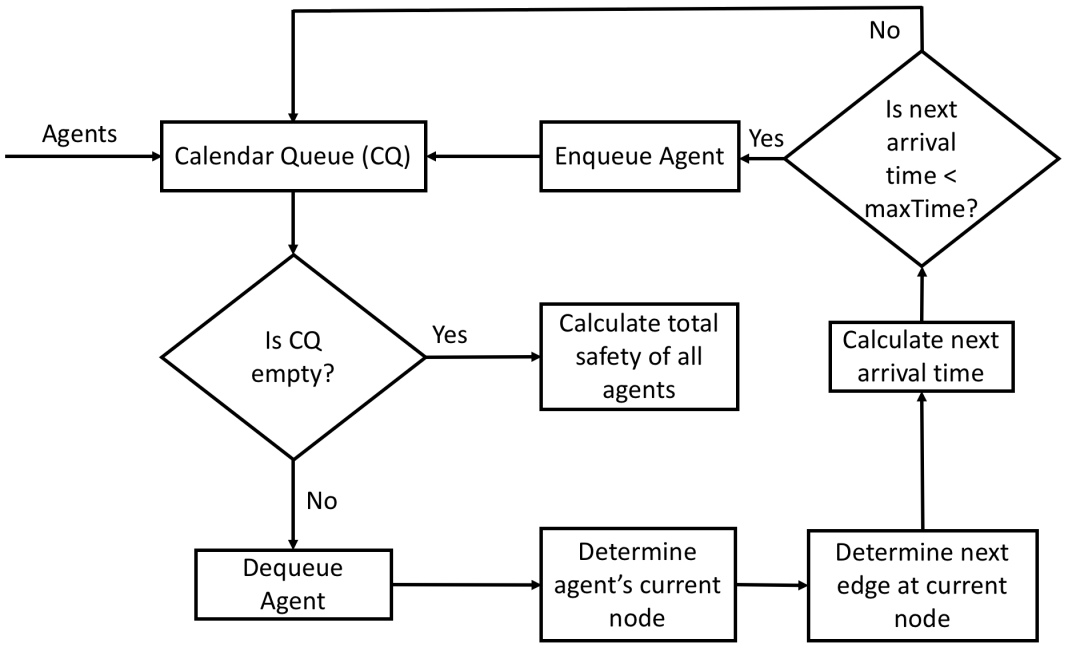


Figure 3.5: Traffic Simulation flow chart.

3.8 Calendar Queue

Calendar queue is a data structure analogous to desk calendar used for ordering future events by date. Calendar queue is the important part of our tool. Previously, we were using the C++ priority queue container. We replaced priority queue with a hand coded calendar queue. Using calendar queue, using arrays, careful memory management, clever sorting of probabilities to reduce testing and other techniques we were able to reduce execution time by about 90%. On a sample laptop we reduced the execution time from 20 minutes to 2 minutes. Fast results are crucial since it is directly related to people's lives. The calendar queue was implemented with the algorithm in [Brown, 1988]. The main operations in calendar queue are enqueue and dequeue. In our implementation, agents reside in calendar queue during traffic simulation as shown in Figure 3.5.

Chapter 4: Experiments and Results

As mentioned above, we use the word "topology" to refer to the graph representation of the transportation network including capacity, speed and safety parameters. We perform a series of experiments to answer the questions from Section 1.1. We use city topologies, algorithms and disaster scenario appropriate for each question. For brevity and clarity we will describe the cities topologies and test cases in Section 4.1. We will describe the details of the test adaptive algorithms in Section 4.2. Finally, we will address each question posed.

The computation time to run experiments depends on the machine configuration in which it is run. We recommend a high performance machine to perform the actual evacuation plan. To carry out all experiments, we used a machine with the configuration given in Table 4.1.

The parameters used in our algorithms to perform experiments, unless otherwise stated, are given in Table 4.2.

Table 4.1: Machine configuration used for performing experiments (configuration from year 2018).

Machine Configuration	
Machine	MacBookPro
Processor	2.7 GHz Intel Core i5
Memory	8 GB 1867 MHz DDR3
OS	macOS Version 10.13

Table 4.2: Parameters used in algorithms for the experiments.

Parameters	
ES-type	ES+
Single fixed sigma	0.1
Population size	100
Child Population size	100
Tournament size	3
Crossover type	uniform
Crossover probability	0.2%
Mutation probability	0.5%
Generations	100
Replicates	30

4.1 Topology of cities used in the experiments

For answering the questions we posed, we used a cross-sectional approach using a variety of cities topologies. The topologies of the cities are graphically shown in Figures 4.1 to 4.15. The thickness of edges in these figures shows the capacity (max number of vehicles per unit time) of edges. When it is thicker, the edge has higher capacity. Blue color nodes in the figures are safe nodes. The safety value ranges from 0 (maximally unsafe) to 1 (maximally safe).

City1: City with a simple 5×5 grid structure having same capacity for all roads as shown in Figure 4.1. This is a simple test case with no complexity. Four completely safe nodes 18, 19, 23 and 24 lie at the bottom right corner of the city. Everywhere else is completely unsafe. Agents are initially distributed at nodes 0, 1, 2, 3 and 4. We will run ES and DC on this city to evacuate all people from danger areas to safe areas and see if ES, DC really solve our evacuation problem. This will help to find answer to Question Q1. This is Test

Case 1a (see **TC1a** below).

Then, we will remove edge 1 ⇔ 2 in City1 which is Test Case 1b (**TC1b**), shown in Figure 4.2.

Also, we will remove 5 edges 1 ⇔ 2, 7 ⇔ 8, 11 ⇔ 12, 14 ⇔ 19, 22 ⇔ 23 in City1 which is Test Case 1b5 (**TC1b5**), shown in Figure 4.3 and re-optimize using previous solutions. While re-optimizing solutions, we will use previous solutions from both ES and DC and see if diversity generated by DC is really helpful for dynamic conditions during disaster. This will help us for finding answer to question Q3.

We will slightly change location of safe nodes in City1. We will make nodes 18 and 19 unsafe with safety 0 and make nodes 21 and 22 safe with safety 1. This is Test Case 1c (**TC1c**) as shown in Figure 4.4. This test case replicates the situation when safe nodes suddenly change during evacuation. Also, we will make huge changes in safe nodes. We will move safe nodes 18, 19, 23 and 24 to 15, 16, 20 and 21 as shown in 4.5. This is Test Case 1c4 (**TC1c4**). We will see if previous knowledge is still helpful in case of huge change, change of safe nodes from one corner of city to other corner.

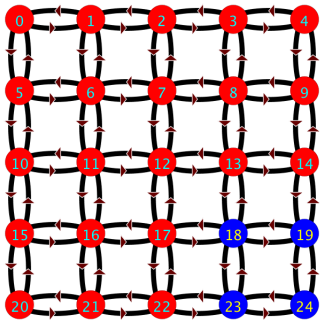


Figure 4.1: Diagram shows Test Case 1a (TC1a). City1 with simple topology having safe nodes 18, 19, 23, 24 (with safety 1.0) at bottom right corner of city and all roads having same capacity. Thickness of edge shows the capacity of edge. Initially agents are distributed at nodes 0, 1, 2, 3, 4.

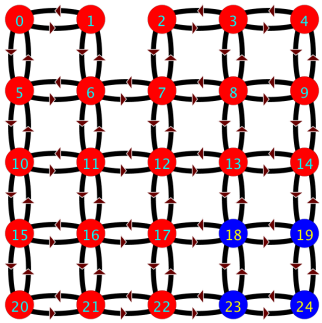


Figure 4.2: Diagram shows Test Case 1b (TC1b). City1 after removing edge 1 ⇔ 2 and having safe nodes 18, 19, 23, 24 (with safety 1.0) at bottom right corner of city and all roads having same capacity. Thickness of edge shows the capacity of edge. Initially agents are distributed at nodes 0, 1, 2, 3, 4.

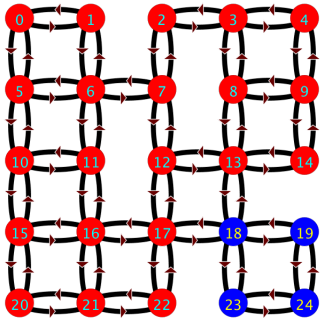


Figure 4.3: Diagram shows Test Case 1b5 (TC1b5). City1 after removing edges 1 ⇔ 2, 7 ⇔ 8, 11 ⇔ 12, 14 ⇔ 19, 22 ⇔ 23 and having safe nodes 18, 19, 23, 24 (with safety 1.0) at bottom right corner of city and all roads having same capacity. Thickness of edge shows the capacity of edge. Initially agents are distributed at nodes 0, 1, 2, 3, 4.

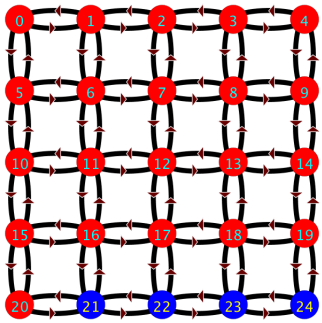


Figure 4.4: Diagram shows Test Case 1c (TC1c). City1 after changing safe nodes (with safety 1.0) from 18, 19, 23, 24 to 21, 22, 23, 24. All roads having same capacity and thickness of edge shows the capacity of edge. Initially agents are distributed at nodes 0, 1, 2, 3, 4.

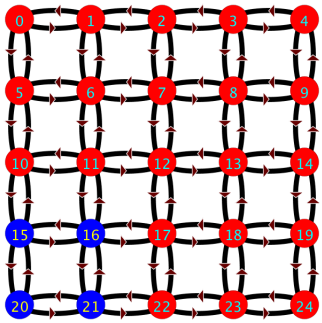


Figure 4.5: Diagram shows Test Case 1c4 (TC1c4). City1 after changing safe nodes (with safety 1.0) from 18, 19, 23, 24 to 15, 16, 20, 21. All roads having same capacity and thickness of edge shows the capacity of edge. Initially agents are distributed at nodes 0, 1, 2, 3, 4.

City2: City with bottle neck problem. Highway with higher capacity outskirts the city as shown in Figure 4.6. This is Test Case 2a (**TC2a**). Inner nodes have no connection with the outer highway. The evacuation is required using outer highway, and agents are supposed to split between north and south highways to move towards the safe place. Agents are initially distributed at nodes 0, 5, 10, 15 and 20. Whenever highway from north end is blocked, as shown in Figure 4.7, all agents should find a way from highway at south to move towards safe place. This is Test Case 2b (**TC2b**). This will answer question Q1 which asks

will ES and DC algorithms produce adaptive solution to evacuation problem.

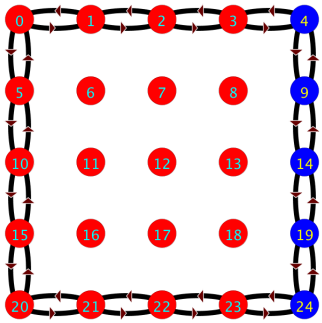


Figure 4.6: Diagram shows Test Case 2a (TC2a). Topology of city with bottle neck problem. Only north and south highways go towards safe nodes with safety 1.0. Thickness of edge shows the capacity of edge. Initially agents are distributed at nodes 0, 5, 10, 15, 20.

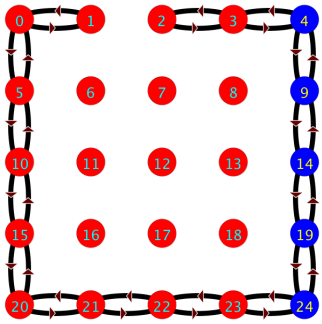


Figure 4.7: Diagram shows Test Case 2b (TC2b). Topology of City2 after removing edges $1 \Leftrightarrow 2$. Only south highway goes towards safe nodes with safety 1.0. Thickness of edge shows the capacity of edge. Initially agents are distributed at nodes 0, 5, 10, 15, 20.

City3: City with path constraint problem where two safe places are located at two corners of the city and vehicles should split equally and move towards those safe places. Figure 4.8 shows the topology for this problem. Agents are initially distributed in nodes 0, 1, 2, 3 and 4. This is Test Case 3a (**TC3a**). Some agents should find path towards the bottom left safe place, and some agents should find path towards the bottom right safe place. Then, we will change the starting nodes of agents to nodes 2, 5, 9, 10 and 14 and see if the

probabilities set from previous optimization are useful while re-optimizing for this condition.

Changing starting nodes for agents simulates the condition when agents could not follow proper direction from traffic manager and happen to reach unexpected nodes. This is Test Case 3d (**TC3d**). This will help us find answer for question Q2. Also, we will see if more diverse solutions from previous optimization are more useful than less diverse solutions. This will help to find answer to question Q3.

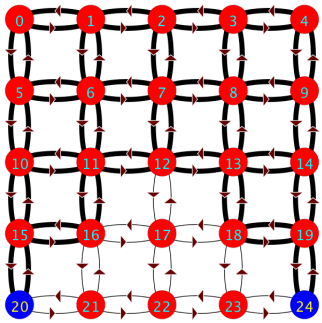


Figure 4.8: Diagram shows Test Case 3a (TC3a). Topology of city with constraint problem. Safe nodes with safety 1.0 lie at bottom left and right corners of city. Thickness of edge shows the capacity of edge. Initially agents are distributed at nodes 0, 1, 2, 3, 4.

City4: City with fast and slow paths. Agents can reach safe place quickly using one path, but there is another alternative path which takes longer time to reach safe place. Agents are supposed to move to safe place using shortest path, but when there is blockage in shortest path, they should move along the longer path towards the safe place. Figure 4.9 shows the topology of the city. Agents are initially distributed at nodes 0, 5, 10, 15 and 20. This is Test Case 4a (**TC4a**). We will see what happens when bridge 1 $\Leftrightarrow$ 2 is collapsed and shortest path is closed. In this case, agents should find the way from longer path towards the safe node. This is Test Case 4b (**TC4b**) as shown in Figure 4.10. This will help in finding answer for question Q1.

Also, we will see if diverse solutions from previous optimization for the city are useful in case of bridge collapse. This will help us in finding answer to question Q3.

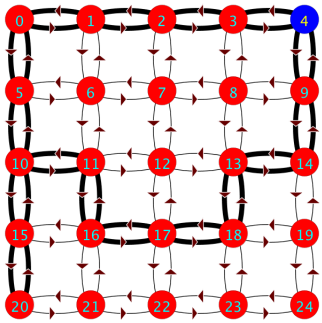


Figure 4.9: Diagram shows Test Case 4a (TC4a). Topology of city with fast and slow paths to go towards safe node 4 with safety 1.0. Thickness of edge shows the capacity of edge. Initially agents are distributed at nodes 0, 5, 10, 15, 20.

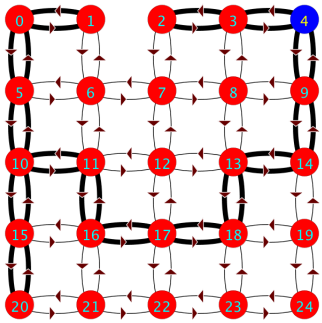


Figure 4.10: Diagram shows Test Case 4b (TC4b). Topology of City4 after removing bridge 1 $\Leftrightarrow$ 2. Thickness of edge shows the capacity of edge. Initially agents are distributed at nodes 0, 5, 10, 15, 20.

City5: City with path problem. The high capacity road which leads vehicles to a safe place travels almost the whole city like a maze as shown in Figure 4.11. The capacity of roads other than highways have a very low capacity, almost negligible. Agents are initially placed at node 0, and they have to reach safe nodes 23, 24 following snaky path. This is Test Case 5a (**TC5a**). We will run test in this city to see if ES, DC finds solutions though roads structure is complex. This will help us in finding answer to question Q1.

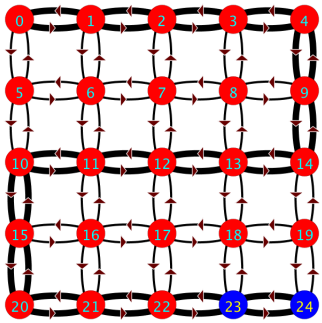


Figure 4.11: Diagram shows Test Case 5a (TC5a). Topology of city with path problem. Maze like road to reach safe nodes 23, 24 with safety 1.0. Thickness of edge shows the capacity of edge. Initially agents are distributed at node 0.

City6: City with single highway where the highway has two safe nodes 3 and 4 at one end and all agents at the other end. Figure 4.12 shows the topology of this city. Agents are initially placed at node 0. This is Test Case 6a (**TC6a**). In this test case, agents should distribute between both nodes. This test is for question Q1.

Figure 4.12: Diagram shows Test Case 6a (TC6a). Topology of city with two safe nodes 3, 4 (with safety 1.0) at one end of highway and agents are at other end of highway. Thickness of edge shows the capacity of edge. Initially agents are distributed at node 0.

City7: City with single highway where the highway has two safe nodes 0 and 4 at two ends and all agents at the center node. Figure 4.13 shows the topology of this city. Agents are initially placed at node 2. This is Test Case 7a (**TC7a**). In this test case, agents should

split in both directions to use both nodes. This test is for question Q1.

Figure 4.13: Diagram shows Test Case 7a (TC7a). Topology of city with two safe nodes 0, 4 (with safety 1.0) at two opposite ends of highway. Thickness of edge shows the capacity of edge. Initially agents are distributed at node 2.

City8: City with Y shaped path as shown in Figure 4.14. Two partial safe nodes 0 and 2 have safety 0.5 and full safe node 4 has safety 1.0. Agents are initially placed at node 1. This is Test Case 8a (**TC8a**). In this test case, agents should finally discover path to the full safe node 4. This test is for question Q1.

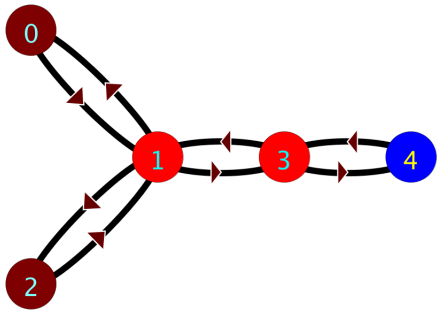


Figure 4.14: Diagram shows Test Case 8a (TC8a). Topology of city with Y shaped path with partial safe nodes 0, 2 with safety 0.5 and full safe node 4 with safety 1.0. Thickness of edge shows the capacity of edge. Initially agents are distributed at node 1.

City9: This city is Boise, Idaho. We use this test case to run all algorithms in a real city to see that those algorithms work in larger networks. Boise has 283 nodes and complex road

structure characteristic of real cities. This is Test Case 9a (**TC9a**). Also, we will change the topology of original Boise city and introduce some problems of bridges collapse and safe areas change. We will make safe nodes 10, 194, 224, 259 and 278 as unsafe with safety 0. And, we will remove edges 1 ⇔ 3, 5 ⇔ 117, 20 ⇔ 21, 29 ⇔ 202, 114 ⇔ 270, 140 ⇔ 141, 196 ⇔ 197, 233 ⇔ 240, 152 ⇔ 161 and 266 ⇔ 280. This new Test Case is 9x (**TC9x**).

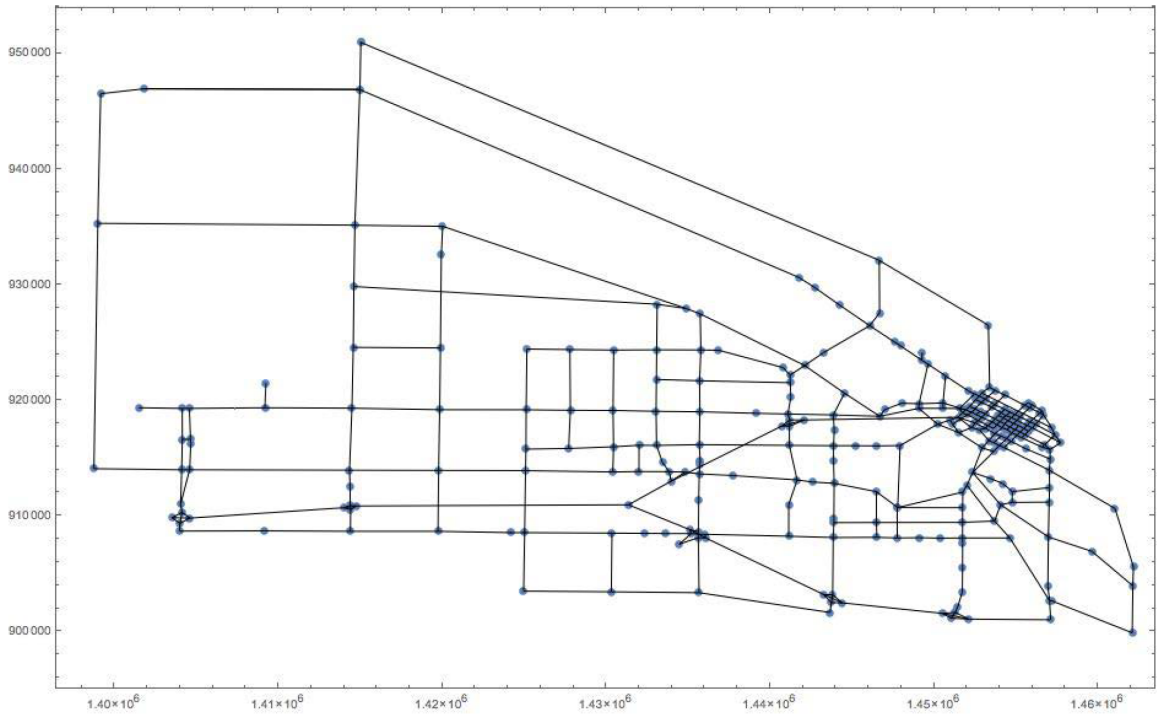


Figure 4.15: Topology of Boise city, adopted from [Drew et al., 2017].

Meaning of suffix a, b, c, d and x in test cases are illustrated in Table 4.3. Table 4.4 shows the summary of test cases used in the experiments. This table also shows which test case is related to which research question.

Table 4.3: Meaning of suffix in Test Cases.

Suffix	Description	Example
a	Test Cases with normal city.	TC1a, TC2a, TC3a
b	Test Cases with bridge/edge collapsed in city, b5 means 5 edges broken.	TC1b, TC1b5, TC2b
c	Test Cases with safe nodes changed in normal city, c4 means 4 safe nodes changed.	TC1c, TC1c4
d	Test Cases with original distribution of agents changed in normal city.	TC3d
x	Test Cases with combined changes in normal city like bridges collapsed, safe nodes changed and agent distribution changed.	9x

Table 4.4: Summary of Test Cases. A Test Case includes both city topology and initial agent placement. The freeflow time of all edges are same for all test cases except TC9a and TC9x where freeflow time of all edges have different values.

Test Case	City	Nds#/ Edgs#	Cnfg	Agnts init Nds	Agnts#/ Ind per agnt	Capacity	Safety	Q#
TC1a	City1	25/80	5x5	0, 1, 2, 3, 4	1000/10	same for all	0 everywhere, 1 at four nodes in corner	1
TC1b	City1	25/78	5x5	0, 1, 2, 3, 4	1000/10	same for all except bridge 1⇔2 collapsed	0 everywhere, 1 at four in corner	2,3
TC1b5	City1	25/70	5x5	0, 1, 2, 3, 4	1000/10	same for all except broken edges 1⇔2, 7⇔8, 11⇔12, 14⇔19, 22⇔23	0 everywhere, 1 at four in corner	2,3
TC1c	City1	25/80	5x5	0, 1, 2, 3, 4	1000/10	same for all	safe nodes changed from 18, 19 to 21, 22	2,3
TC1c4	City1	25/80	5x5	0, 1, 2, 3, 4	1000/10	same for all	safe node changed from 18, 19, 23, 24 to 15, 16, 20, 21	2,3
TC2a	City2	25/32	5x5	0, 5, 10, 15, 20	1000/10	same for all	0 everywhere, 1 at right edge	1
TC2b	City2	25/30	5x5	0, 5, 10, 15, 20	1000/10	same for all except bridge 1⇔2 collapsed	0 everywhere, 1 at right edge	1,2,3
TC3a	City3	25/80	5x5	0, 1, 2, 3, 4	1000/10	low for lower edge	0 everywhere, 1 at lower right and left	1
TC3d	City3	25/80	5x5	2, 5, 9, 10, 14	1000/10	low for lower edge	0 everywhere, 1 at lower right and left	2,3
TC4a	City4	25/80	5x5	0, 5, 10, 15, 20	1000/10	two high capacity paths to safe area and others have low	0 everywhere, 1 at upper right corner	1
TC4b	City4	25/78	5x5	0, 5, 10, 15, 20	1000/10	path 0-1-2-3-4 blocked; only path 10-11-16-17-18-13-14-9-4 to safe area	0 everywhere, 1 at upper right corner	2,3
TC5a	City5	25/80	5x5	0	1000/10	single snaking path of high capacity	0 everywhere, 1 at two lower left corner nodes	1
TC6a	City6	5/8	1x5	0	1000/10	same for all	0 everywhere, 1 at two nodes at one end	1
TC7a	City7	5/8	1x5	2	1000/10	same for all	0 everywhere, 1 at two nodes at opposite ends	1
TC8a	City8	5/8	NA	1	1000/10	same for all	0.5 at two nodes 1 at one node, 0 for others	1
TC9a	Boise	283	real city	all nodes	283/10	same for all	39 safe nodes with 1 at different places	1
TC9x	Boise	283	real city	all nodes	283/10	same for all	changed from 1 to 0 for 5 nodes and 10 bidirectional edges removed	2,3

4.2 Adaptive Algorithms for Test Cases Run

Figure 4.16 is a temporal subset of Figure 1.1. It shows one re-optimization step at time t_{k+1}. This uses an initial population from population produced by optimization at t_k. During the optimization step that begins at time t_k, the disaster continues to unfold in real time. When the optimization step is done it is now time t_{k+1}. The information from the successful optimization is now used to solve the new evacuation problem as presented at t_{k+1}. This will be a different problem because the real evacuees have hopefully moved, but also there are probably changes in the transportation network and in safety of nodes and edges. So we use the solution at time t_k to evacuate the current city. It is our best guess. A key objective of our adaptive algorithm is to shorten time $t_{k+1} - t_k$, providing better solutions to a constantly changing evacuation scenario, so that the evacuation plan is always up-to-date with problems.

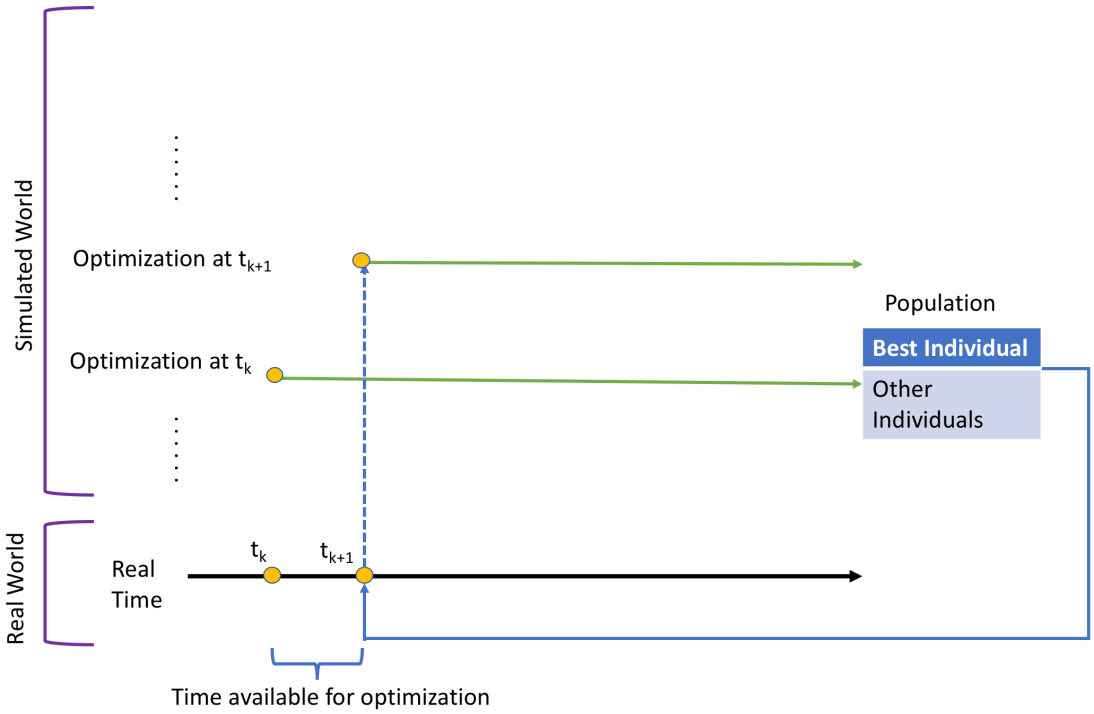


Figure 4.16: Re-optimization at time t_{k+1} using population from optimization at t_k.

There are two parameters we can change to affect this meta-algorithm composed in the steps seen in Figure 4.16. First is the algorithm we choose. In this work we are using ES and ES with Deterministic Crowding. The second is what part of the population is transferred from one optimization step to the next. A disadvantage of using exactly the previous population is that it may have converged to a very specific solution for the t_k evacuation making it harder for the population to adapt to the t_{k+1} disaster evacuation problem. We propose that we may be able to find an appropriate population to bring forward to start the next population. These two parameters define our set of adaptive algorithms that we will test.

So, we use the technique of transfer learning in city evacuation. For re-optimization, there are different ways of using population from previous optimization. Depending upon how we use starting initial population, we have different adaptive algorithms for running optimization. In the population, candidate solutions are the probabilities of edges. Probability of edge represents how probable is the edge being chosen as next edge at node during evacuation. Set of probabilities of all edges in the city forms a candidate solution. So, we may use the term probabilities and candidate solution interchangeably.

We run test cases described in Section 4.1 in different adaptive algorithms. Adaptive Algorithms are described as:

1. Running Evolution Strategy for Test Case with random probabilities set as a starting population. This adaptive algorithm is denoted by **ES** in graphs and tables.

2. Running Deterministic Crowding for Test Case with random probabilities set as a starting population. This adaptive algorithm is denoted by **DC** in graphs and tables.

3. Running Evolution Strategy for Test Case with initial starting population as all probabilities set from population produced by previous optimization using ES in the normal city (i.e. test cases with suffix 'a' like TC1a, TC2a, etc.,). This adaptive algorithm is denoted by **ESpall** in graphs and tables. This is different from **ES** above, in that the

previous algorithm starts with random initial population, where this algorithm starts with an initial population generated from previous optimization, which was run using ES algorithm in normal city. By normal we mean unchanged, see Table 4.3.

4. Running Evolution Strategy for Test Case with initial starting population as all probabilities set from population produced by previous optimization using DC in the normal city (i.e. test cases with suffix 'a' like TC1a, TC2a, etc.,). This adaptive algorithm is denoted by **DCpall** in graphs and tables. This is different from **DC** above, in that the previous algorithm starts with random initial population, where this algorithm starts with an initial population generated from previous optimization, which was run using DC algorithm in normal city. By normal we mean unchanged, see Table 4.3.

5. Running Evolution Strategy for Test Case where one of the starting solutions is the best individual (probabilities set) in the population generated by previous optimization using ES in the normal city (i.e. test cases with suffix 'a' like TC1a, TC2a, etc.,). For each of the other individuals in the initial population, we clone the best individual and make small change in genes (probability values) using mutation. This is "mutation on leash". The parameters we use in this "mutation on leash" are sigma as 0.1 and mutation probability as 0.1. This adaptive algorithm is denoted by **ESpone** in graphs and tables. This is different from ESpall in the sense that algorithm uses all set of individuals from previous population, but this algorithm uses only the best individual, clone it and make small change in genes to generate initial population.

6. Running Evolution Strategy for Test Case where one of the starting solutions is the best individual (probabilities set) in the population generated by previous optimization using DC in the normal city (i.e. test cases with suffix 'a' like TC1a, TC2a, etc.,). For each of the other individuals in the initial population, we clone the best individual and make small change in genes (probability values) using mutation. This is "mutation on leash". The parameters we use in this "mutation on leash" are sigma as 0.1 and

mutation probability as 0.1. This adaptive algorithm is denoted by **DCpone** in graphs and tables. This is different from DCpall in the sense that algorithm uses all set of individuals from previous population, but this algorithm uses only the best individual, clone it and make small change in genes to generate initial population.

We run test cases in these adaptive algorithms and place all the results together so that we can compare the results from all adaptive algorithms.

4.3 Results and Analysis

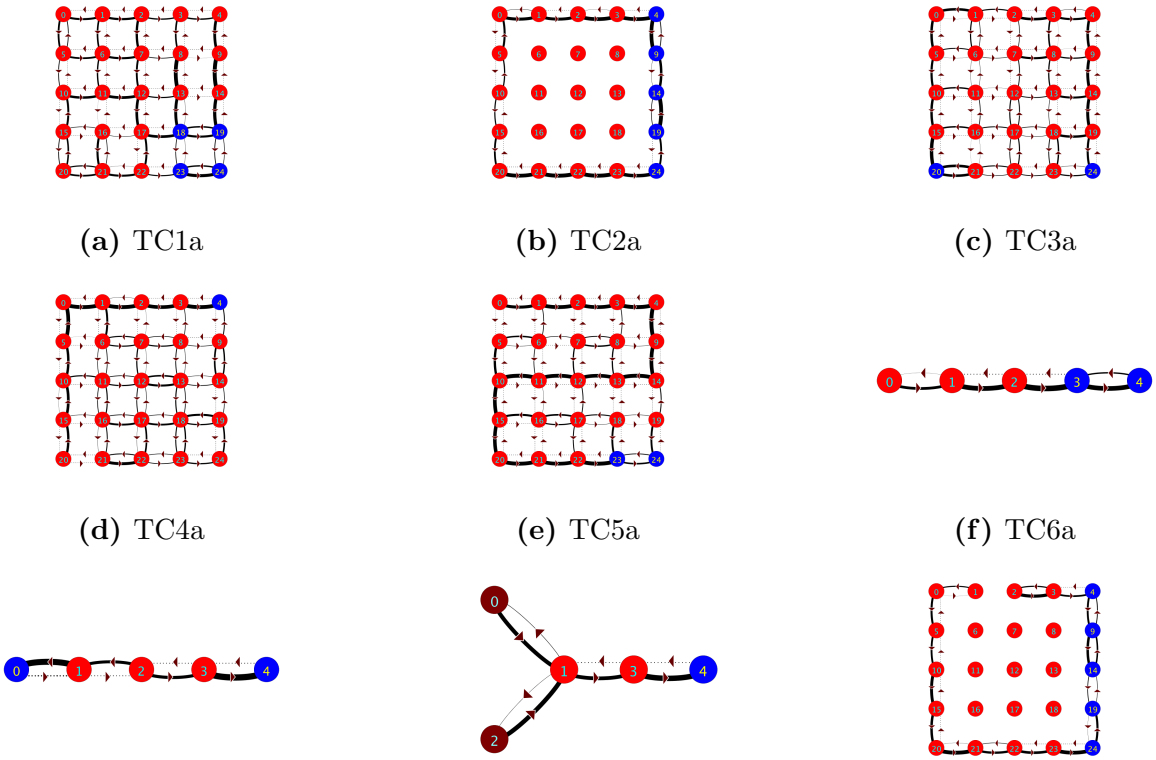
We have performed experiments and tried to find answers to the questions mentioned in Section 1.1. We have carried out tests using different test cases mentioned in Section 4.1. We have presented result in **Table** and **Graph**. We have repeated all test cases 30 times and taken average of fitness from all the repetitions for the result. In the plot we can also see error bars which is the standard deviation of fitness value from all those 30 repetitions. The terms ES, DC, ESpall, DCpall, ESpone and DCpone used in result are mentioned in Section 4.2. We have arranged the experiments according to our research questions.

4.3.1 Experiments for Question Q1

Question Q1 asks whether evolutionary algorithms can solve the city evacuation problem. That means, is it reasonable to use an evolution based algorithm to solve this kind of problem? If for our test cases we can find one or more evolution based algorithms which can solve them, that would suggest that these algorithms are worth investigating. Since we have chosen as our basic representation a probability vector it is reasonable to try evolutionary algorithms that focus on vectors of real values. We chose Evolution Strategy (ES) and ES with Deterministic Crowding for our base algorithms. We embed each of these algorithms as the optimization step in realtime solver seen in Figure 1.1. We perform experiments on variety of test cases using this realtime solver. But, as we focus on answering our research

questions, we perform experiments for optimization steps in simulated world as shown in Figure 4.16 only in order to avoid the confounding issues of complex experiments.

Figure 4.17 illustrates the topology of test cases after running evolutionary algorithm where thickness of edge shows the probability of that edge. We present the result of the optimization as Tables 4.6 and 4.7 and Figures 4.18 and 4.19. The result shown is the mean safety of agents obtained with best individual in population from the last generation. We also show the standard deviation of safety values in tables and graphs. The mean and standard deviation are calculated from 30 replicates.



(a) TC1a **(b)** TC2a **(c)** TC3a

(d) TC4a **(e)** TC5a **(f)** TC6a

(g) TC7a **(h)** TC8a **(i)** TC2b

Figure 4.17: Topology of test cases after running ES (similar diagram with DC). Thickness of edge shows the probability of edge being chosen.

From Figures 4.17, we can see that the paths with high probability lead towards the safe nodes. This shows that agents are able to find paths to the safe nodes in all test cases. This means evolutionary algorithms are able to discover paths to move agents to safe nodes

during evacuation. The result from these figures are summarized in Table 4.5.

Table 4.5: Summary of result from test cases used for question Q1.

Test Case	Description
TC1a	Agents are able to find paths towards safe nodes.
TC2a	Agents are splitting to safe nodes from north and south highways utilizing all highways capacity.
TC3a	Agents are able to find paths towards safe nodes at both corners of city.
TC4a	Agents are able to find shortest paths towards safe nodes.
TC5a	Agents are able to find paths to safe node though high capacity highway is snaky.
TC6a	Agents use both safe nodes located at one end of highway.
TC7a	Agents split to both directions to use safe nodes at opposite ends of highway.
TC8a	Agents find paths to fully safe node though partial safe nodes were present.
TC2b	Agents are able to find path from south highway when north highway is blocked.

Table 4.6: Part 1: Mean safety of agents obtained with best individual in population from last generation over simulation hour for test cases after running ES, DC. Parenthesis indicates the standard deviation of safety values. Mean and standard deviation are calculated from optimization carried out for 30 repetitions.

(a) TC1a

Simulation Hour [hr]	Mean Safety with ES	Mean Safety with DC
0	0 (0)	0 (0)
0.05	0.042 (0.011)	0.041 (0.012)
0.1	0.281 (0.041)	0.297 (0.041)
0.15	0.558 (0.061)	0.595 (0.047)
0.2	0.766 (0.053)	0.796 (0.038)
0.25	0.883 (0.032)	0.901 (0.025)
0.3	0.945 (0.018)	0.954 (0.013)
0.35	0.975 (0.008)	0.979 (0.007)
0.4	0.989 (0.005)	0.991 (0.004)
0.45	0.996 (0.002)	0.997 (0.002)
0.5	1 (0)	1 (0)

(b) TC2a

Simulation Hour [hr]	Mean Safety with ES	Mean Safety with DC
0	0 (0)	0 (0)
0.05	0.04 (0.013)	0.045 (0.013)
0.1	0.254 (0.042)	0.275 (0.038)
0.15	0.520 (0.071)	0.548 (0.057)
0.2	0.745 (0.073)	0.762 (0.061)
0.25	0.876 (0.045)	0.890 (0.037)
0.3	0.945 (0.024)	0.951 (0.020)
0.35	0.976 (0.012)	0.979 (0.009)
0.4	0.991 (0.006)	0.991 (0.005)
0.45	0.997 (0.002)	0.997 (0.002)
0.5	1 (0)	1 (0)

(c) TC3a

Simulation Hour [hr]	Mean Safety with ES	Mean Safety with DC
0	0 (0)	0 (0)
0.05	0.037 (0.017)	0.042 (0.013)
0.1	0.287 (0.081)	0.317 (0.049)
0.15	0.568 (0.120)	0.619 (0.069)
0.2	0.779 (0.106)	0.814 (0.051)
0.25	0.906 (0.055)	0.915 (0.031)
0.3	0.962 (0.018)	0.961 (0.016)
0.35	0.984 (0.006)	0.983 (0.008)
0.4	0.993 (0.003)	0.992 (0.005)
0.45	0.996 (0.001)	0.997 (0.001)
0.5	0.999 (0.000)	0.999 (0.000)

(d) TC4a

Simulation Hour [hr]	Mean Safety with ES	Mean Safety with DC
0	0 (0)	0 (0)
0.05	0.028 (0.008)	0.028 (0.009)
0.1	0.173 (0.023)	0.183 (0.024)
0.15	0.364 (0.031)	0.377 (0.033)
0.2	0.579 (0.042)	0.596 (0.044)
0.25	0.796 (0.053)	0.809 (0.052)
0.3	0.931 (0.027)	0.939 (0.024)
0.35	0.975 (0.012)	0.982 (0.009)
0.4	0.991 (0.006)	0.993 (0.003)
0.45	0.997 (0.001)	0.998 (0.001)
0.5	1 (0)	1 (0)

(e) TC5a

Simulation Hour [hr]	Mean Safety with ES	Mean Safety with DC
0	0 (0)	0 (0)
0.05	0 (0)	0 (0)
0.1	3.333 (0.000)	0 (0)
0.15	0.001 (0.000)	0.001 (0.000)
0.2	0.004 (0.001)	0.004 (0.001)
0.25	0.132 (0.052)	0.032 (0.013)
0.3	0.562 (0.105)	0.227 (0.071)
0.35	0.858 (0.057)	0.528 (0.100)
0.4	0.965 (0.019)	0.766 (0.081)
0.45	0.995 (0.003)	0.916 (0.036)
0.5	1 (0)	0.959 (0.018)

(f) TC6a

Simulation Hour [hr]	Mean Safety with ES	Mean Safety with DC
0	0 (0)	0 (0)
0.05	0.061 (0.005)	0.061 (0.005)
0.1	0.171 (0.015)	0.173 (0.015)
0.15	0.332 (0.026)	0.341 (0.026)
0.2	0.451 (0.025)	0.459 (0.022)
0.25	0.605 (0.041)	0.622 (0.039)
0.3	0.731 (0.035)	0.744 (0.033)
0.35	0.879 (0.052)	0.898 (0.049)
0.4	0.969 (0.023)	0.975 (0.020)
0.45	0.997 (0.002)	0.996 (0.003)
0.5	1 (0)	1 (0)

Table 4.7: Part 2: Mean safety of agents obtained with best individual in population from last generation over simulation hour for test cases after running ES, DC. Parenthesis indicates the standard deviation of safety values. Mean and standard deviation are calculated from optimization carried out for 30 repetitions.

(a) TC7a

Simulation Hour [hr]	Mean Safety with ES	Mean Safety with DC
0	0 (0)	0 (0)
0.05	0.114 (0.034)	0.116 (0.029)
0.1	0.350 (0.100)	0.355 (0.082)
0.15	0.538 (0.138)	0.550 (0.107)
0.2	0.730 (0.171)	0.756 (0.126)
0.25	0.847 (0.142)	0.887 (0.094)
0.3	0.924 (0.085)	0.960 (0.046)
0.35	0.974 (0.033)	0.988 (0.017)
0.4	0.995 (0.005)	0.996 (0.004)
0.45	0.999 (0.001)	0.999 (0.001)
0.5	1 (0)	1 (0)

(b) TC8a

Simulation Hour [hr]	Mean Safety with ES	Mean Safety with DC
0	0 (0)	0 (0)
0.05	0.093 (0.008)	0.090 (0.008)
0.1	0.293 (0.024)	0.286 (0.028)
0.15	0.500 (0.040)	0.495 (0.048)
0.2	0.702 (0.059)	0.696 (0.063)
0.25	0.870 (0.054)	0.863 (0.059)
0.3	0.952 (0.027)	0.952 (0.033)
0.35	0.982 (0.011)	0.984 (0.009)
0.4	0.993 (0.005)	0.994 (0.004)
0.45	0.998 (0.001)	0.999 (0.001)
0.5	1 (0)	1 (0)

(c) TC9a (Boise city)

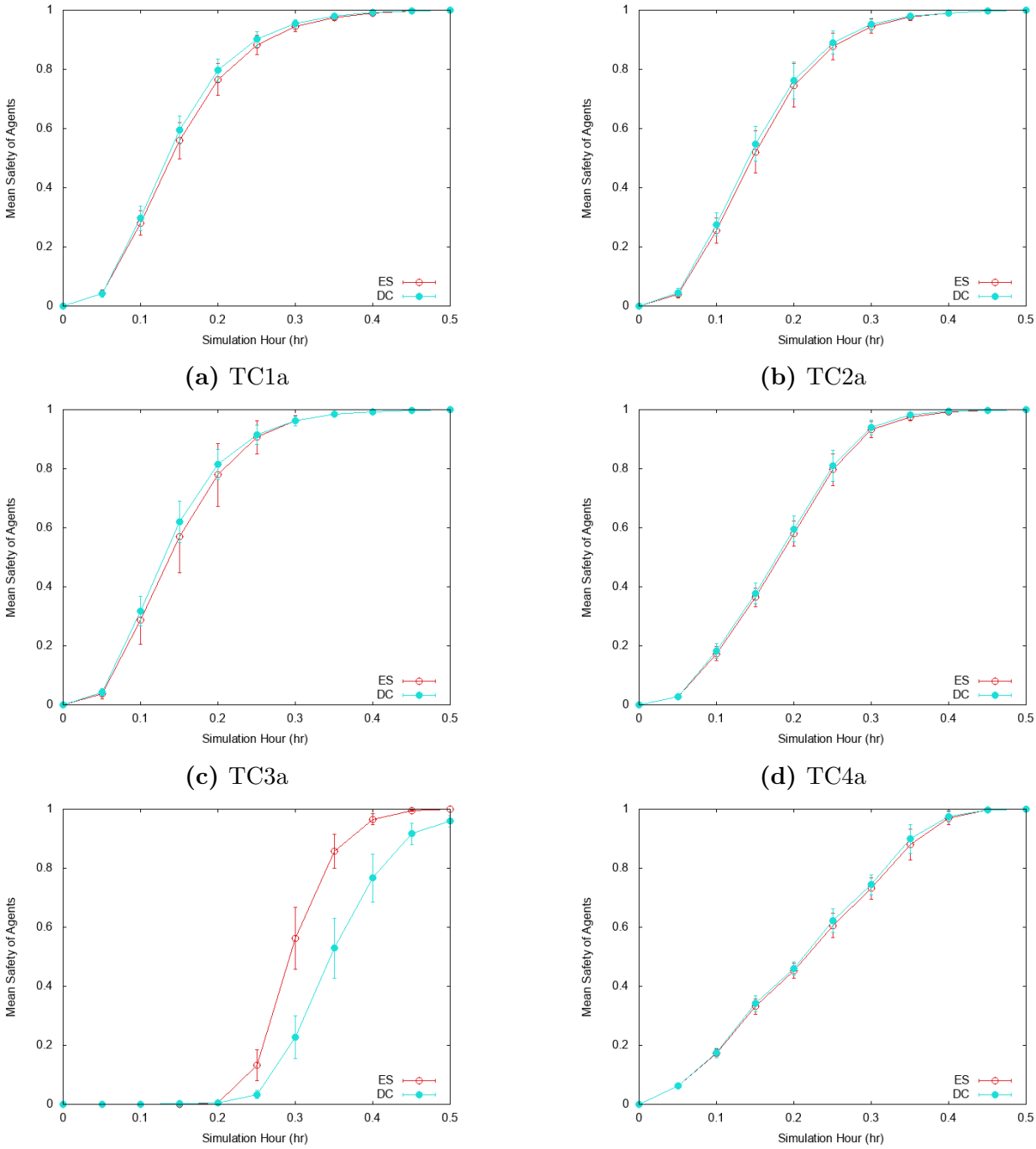
Simulation Hour [hr]	Mean Safety with ES	Mean Safety with DC
0	0.137 (1.490)	0.137 (1.490)
0.5	0.207 (0.094)	0.253 (0.084)
1	0.475 (0.087)	0.485 (0.073)
1.5	0.676 (0.064)	0.671 (0.072)
2	0.802 (0.047)	0.785 (0.070)
2.5	0.883 (0.038)	0.856 (0.065)
3	0.928 (0.026)	0.905 (0.053)
3.5	0.960 (0.016)	0.938 (0.050)
4	0.980 (0.011)	0.962 (0.042)
4.5	0.991 (0.006)	0.974 (0.036)
5	1 (0)	0.984 (0.028)

(d) TC2b

Simulation Hour [hr]	Mean Safety with ES	Mean Safety with DC
0	0 (0)	0 (0)
0.04	0 (0)	0 (0)
0.08	0.098 (0.023)	0.099 (0.023)
0.12	0.293 (0.035)	0.295 (0.028)
0.16	0.445 (0.039)	0.447 (0.034)
0.2	0.630 (0.040)	0.635 (0.034)
0.24	0.800 (0.046)	0.803 (0.037)
0.28	0.935 (0.027)	0.939 (0.023)
0.32	0.979 (0.008)	0.981 (0.007)
0.36	0.996 (0.002)	0.996 (0.002)
0.4	1 (0)	1 (0)

(e) TC5a with enough evacuation time limit

Simulation Hour [hr]	Mean Safety with ES	Mean Safety with DC
0	0 (0)	0 (0)
0.06	0 (0)	0 (0)
0.12	0.000 (0.000)	0.000 (0.000)
0.18	0.002 (0.001)	0.002 (0.001)
0.24	0.016 (0.007)	0.018 (0.007)
0.3	0.309 (0.081)	0.310 (0.082)
0.36	0.733 (0.079)	0.717 (0.082)
0.42	0.929 (0.037)	0.914 (0.034)
0.48	0.978 (0.013)	0.968 (0.016)
0.54	0.995 (0.002)	0.992 (0.004)
0.6	0.999 (0.000)	0.998 (0.001)



(a) TC1a

(b) TC2a

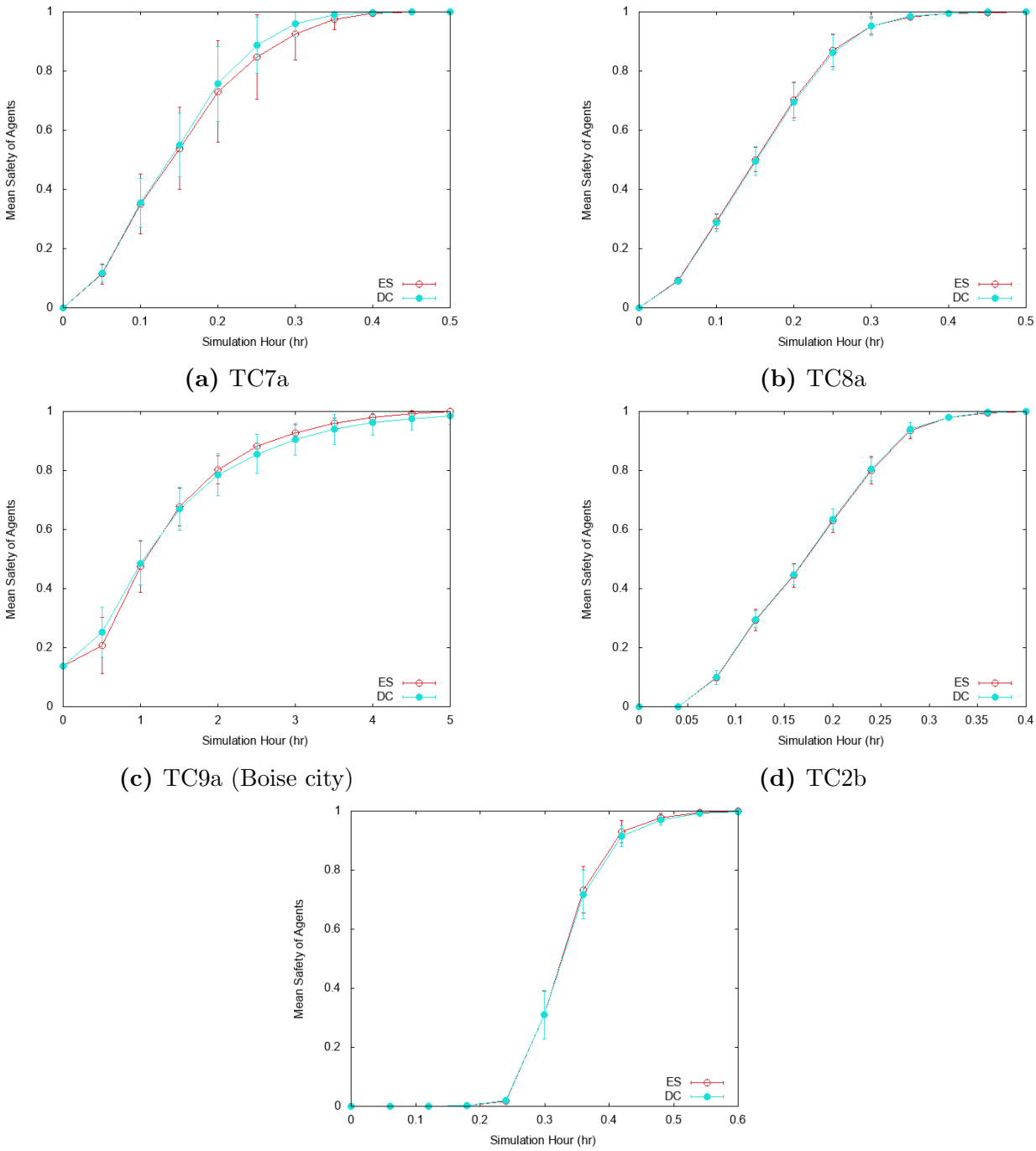
(c) TC3a

(d) TC4a

(e) TC5a

(f) TC6a

Figure 4.18: Part 1: Mean safety of agents obtained with best individual in population from last generation over simulation hour for test cases after running ES, DC. Error bars are standard deviation of safety values. Mean and standard deviation are calculated from optimization carried out for 30 repetitions.



(a) TC7a

(b) TC8a

(c) TC9a (Boise city)

(d) TC2b

(e) TC5a with enough evacuation time limit

Figure 4.19: Part 2: Mean safety of agents obtained with best individual in population from last generation over simulation hour for test cases after running ES, DC. Error bars are standard deviation of safety values. Mean and standard deviation are calculated from optimization carried out for 30 repetitions.

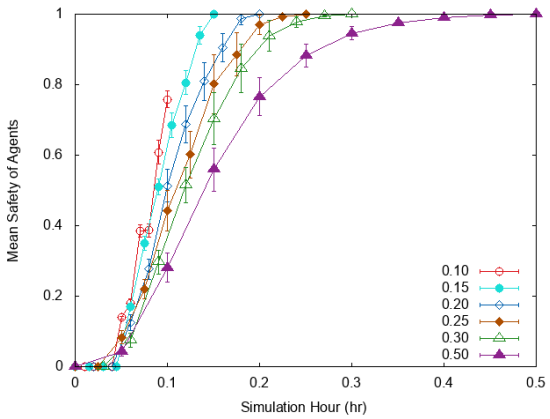
From Tables 4.6 and 4.7 and Figures 4.18 and 4.19, we can see that ES, DC are able to

generate feasible solutions for city evacuation problems. The mean safety value of agents obtained with best individual in population from last generation is increasing over the simulation hour. We can see that Figure 4.18e is different from other figures. This is because 0.5 hour is not sufficient for TC5a to evacuate all people using DC algorithm. Figure 4.19e shows the similar graph for TC5a with evacuation time of 0.6 hours.

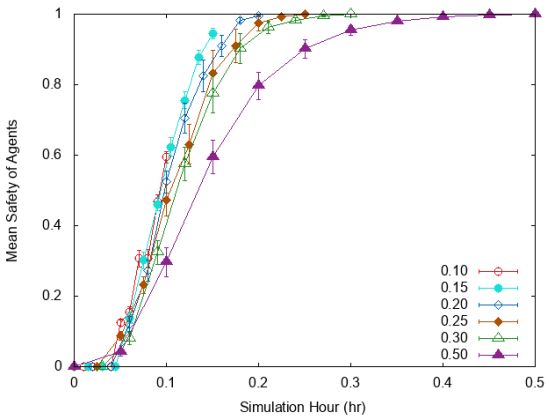
Though almost all graphs in Figures 4.18 and 4.19 seem to reach safety value 1.0 at simulation time 0.5 hours, it is not necessary that those test cases take 0.5 hours to make safety 1.0. When we run those test cases in different objective evacuation times like 0.10, 0.15, 0.20, 0.25, 0.3, 0.5, we found that some test cases can find a safety of 1.0 even at 0.2 hours. The graphs of goodness (safety) over different evacuation time limit as shown in Figures 4.20 and 4.21 suggest to us that evacuation time limit causes slower routes to be acceptable during evacuation. When there is more time to evacuate city, it may choose longer routes, but when there is less time to evacuate, it may find shorter routes. This will have important practical implications. However, there is some time limit which is at least required to safely evacuate all people. Table 4.8 shows the minimum time in hours required to safely evacuate people with fitness value greater than 0.98 using ES and DC algorithms in different test cases. From the table, we can see that DC requires more evacuation time than ES to find optimal solutions.

Table 4.8: Minimum evacuation time required to evacuate people with fitness greater than 0.98 using ES and DC algorithms in different test cases.

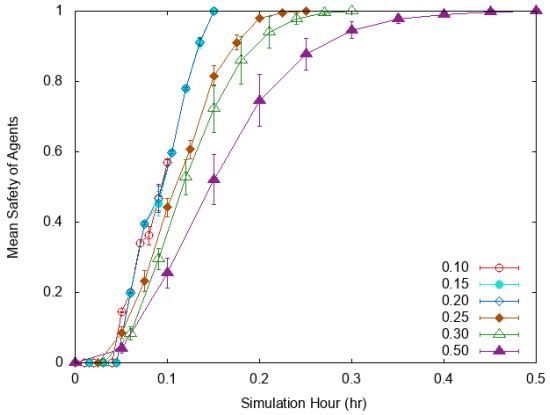
Test Case	ES simulation hour (hr)	DC simulation hour (hr)
TC1a	0.15	0.20
TC2a	0.15	0.15
TC3a	0.15	0.20
TC5a	0.35	0.55
TC6a	0.35	0.35



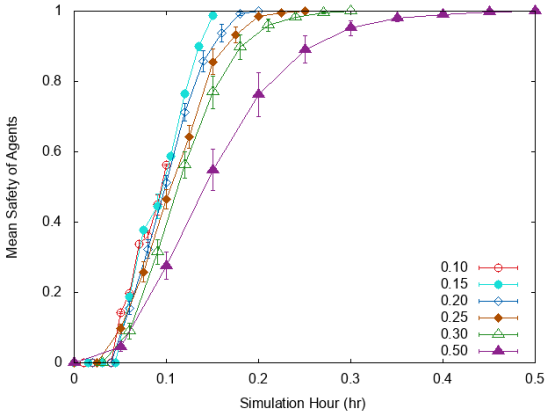
(a) TC1a with ES



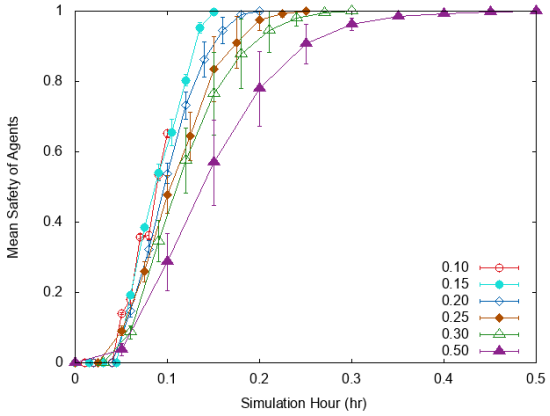
(b) TC1a with DC



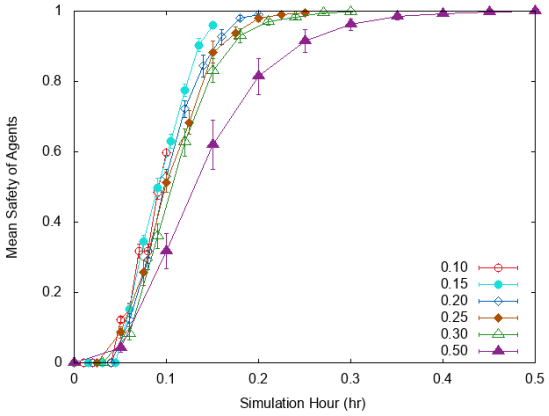
(c) TC2a with ES



(d) TC2a with DC

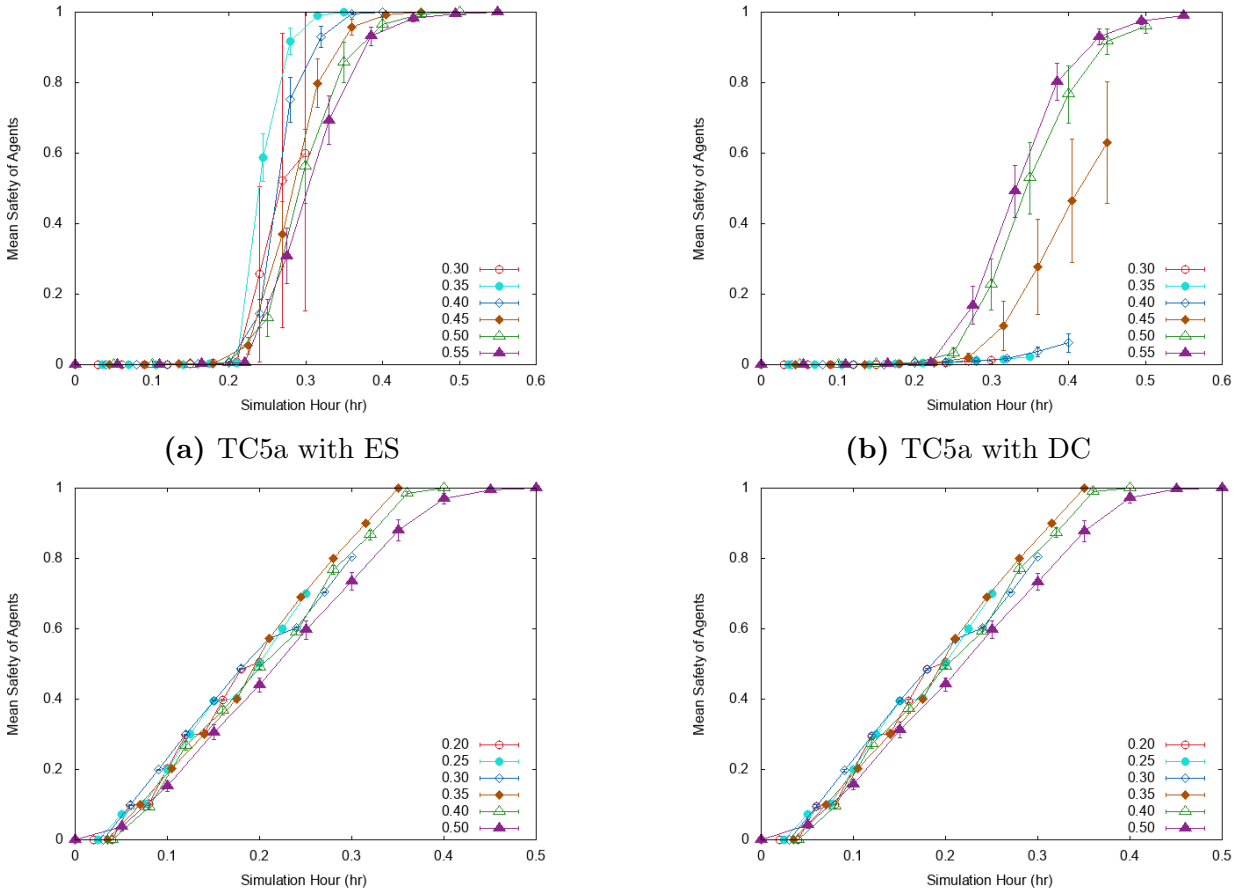


(e) TC3a with ES



(f) TC3a with DC

Figure 4.20: Part1: Mean safety of agents obtained with best individual in population from last generation over different simulation hour for test cases after running ES, DC. Simulation hour means evacuation time limit. Error bars are standard deviation of safety values. Mean and standard deviation are calculated from optimization carried out for 30 repetitions.



(a) TC5a with ES

(b) TC5a with DC

(c) TC6a with ES

(d) TC6a with DC

Figure 4.21: Part2: Mean safety of agents obtained with best individual in population from last generation over different simulation hour for test cases after running ES, DC. Simulation hour means evacuation time limit. Error bars are standard deviation of safety values. Mean and standard deviation are calculated from optimization carried out for 30 repetitions.

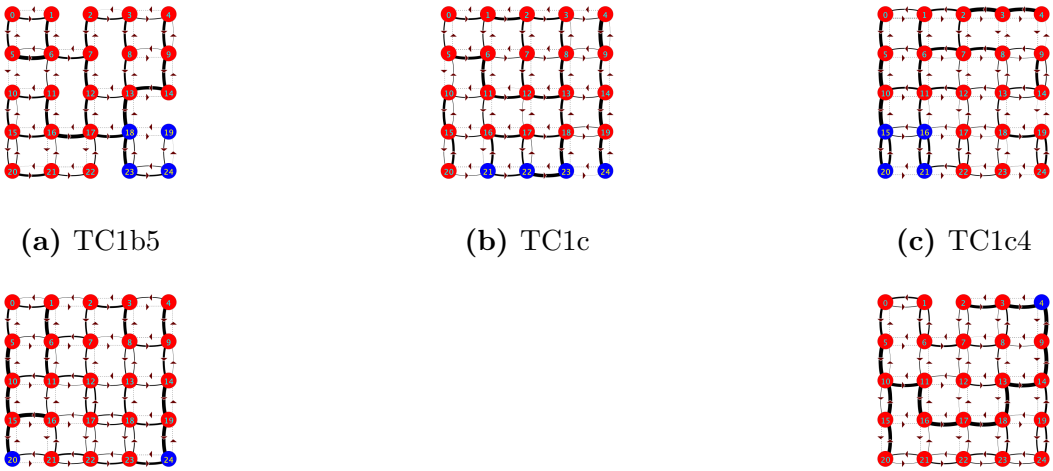
From all these experiments performed to find answer for question Q1, we saw that evolutionary algorithms can solve the real time evacuation problem in an efficient and robust way.

4.3.2 Experiments for Question Q2

Question Q2 asks can we use transfer learning in the city evacuation problem?

To find answer to question Q2, we perform experiments using TC1b5, TC1c, TC1c4, TC3d, TC4b and TC9x test cases. We have chosen these test cases to address all types

of new possible problems in the topology of original city. For example, TC1b5 and TC4b for bridges collapse, TC1c and TC1c4 for safe areas change, TC3d for people not following instructions. TC9x addresses the real city scenario and combined problems of safe nodes change and bridges collapse. We run all adaptive algorithms mentioned in Section 4.2 for all test cases. We plot graph for fitness over generations and see how use of previous population plays role in optimization. Figures 4.22 show the topology of cities after running evolutionary algorithm where thickness of edge shows the probability of that edge. Figures 4.23 present the result of fitness over generations.



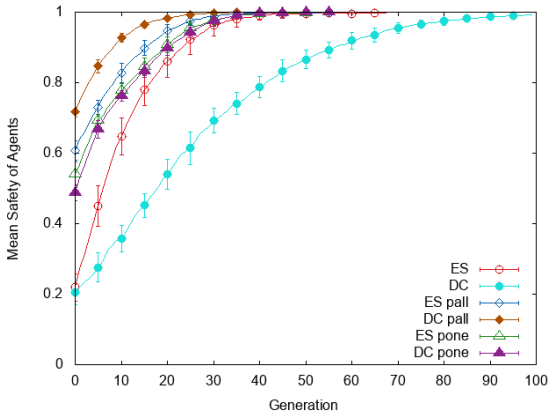
(a) TC1b5 **(b)** TC1c **(c)** TC1c4

(d) TC3d **(e)** TC4b

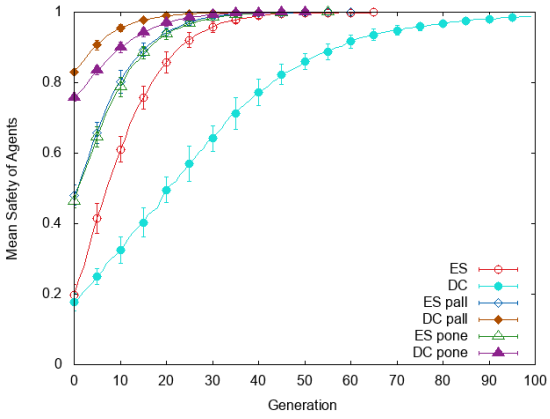
Figure 4.22: Topology of test cases after running ES with initial probabilities set from previous DC optimization. Thickness of edge shows the probability of edge being chosen.

Table 4.9: Analysis of result from test cases used for question Q2

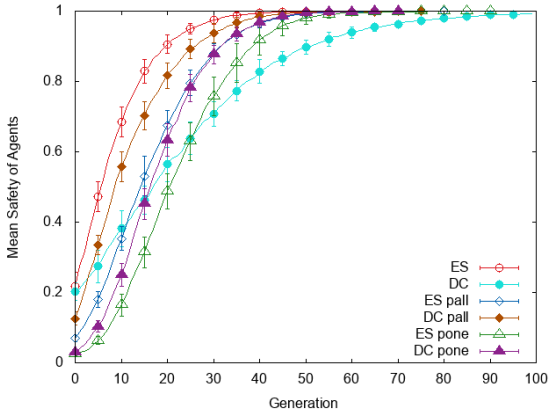
Test Case	Description
TC1b5	Agents are able to find paths towards safe nodes.
TC1c	Agents are able to find paths towards safe nodes though location of some safe nodes changed.
TC1c4	Agents are able to find paths towards new safe nodes though location of some safe nodes changed drastically.
TC3d	Agents are able to find paths towards safe nodes though agents initial distribution to nodes changed.
TC4b	Agents are able to find longer path to safe node though short path is blocked.



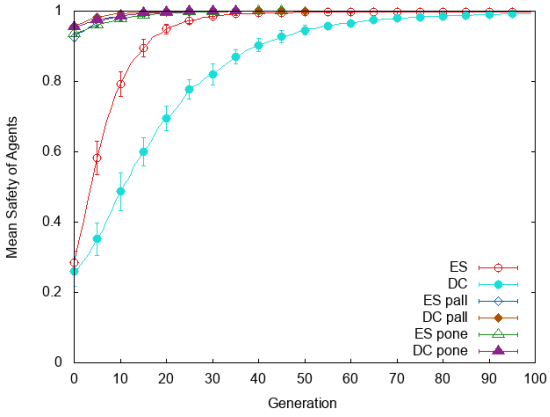
(a) TC1b5 with simulation hour 0.2



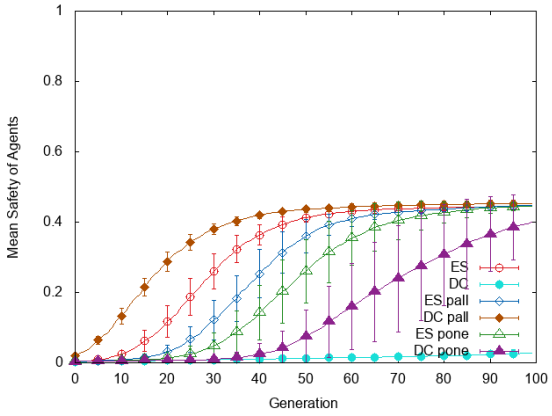
(b) TC1c with simulation hour 0.2



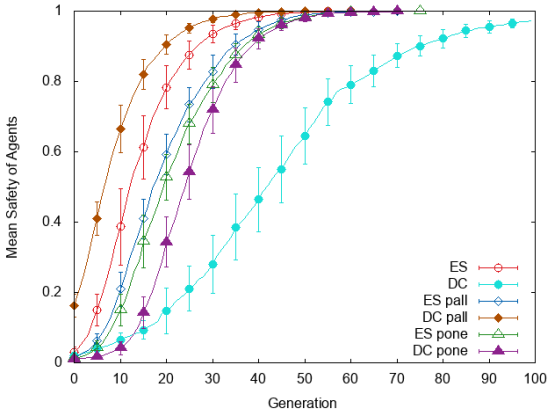
(c) TC1c4 with simulation hour 0.2



(d) TC3d with simulation hour 0.2



(e) TC4b with simulation hour 0.2



(f) TC4b with simulation hour 0.4

Figure 4.23: Mean safety of agents obtained with best individual in population over generation for test cases after running ES, DC, ES using all initial probabilities from previous ES optimization (ESpall), ES using all initial probabilities from previous DC optimization (DCpall), ES using initial probabilities from best probabilities of previous ES optimization (ESpone) and ES using initial probabilities from best probabilities of previous DC optimization (DCpone). Error bars are standard deviation of safety values. Mean and standard deviation are calculated from optimization carried out for 30 repetitions.

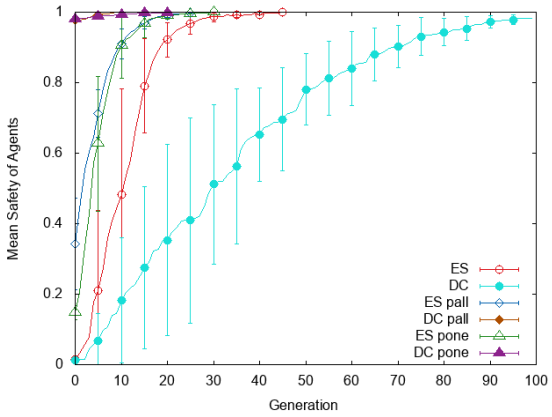


Figure 4.24: Mean safety of agents obtained with best individual in population over generation for Boise city with changed topology (TC9x with simulation hour 4.0) after running ES, DC, ES using all initial probabilities from previous ES optimization (ESpall), ES using all initial probabilities from previous DC optimization (DCpall), ES using initial probabilities from best probabilities of previous ES optimization (ESpone) and ES using initial probabilities from best probabilities of previous DC optimization (DCpone). Error bars are standard deviation of safety values. Mean and standard deviation are calculated from optimization carried out for 30 repetitions.

In Figures 4.23c, We can see that ES with random probabilities set performs better than using probabilities set from previous population because there is drastic change in safe areas. Safe areas are moved from one corner of the city to another corner of city while comparing with original city TC1a. In the case of drastic change in city topology, it is better to start ES with random probabilities than using previous knowledge.

For test case TC4b, because of its topology, 0.2 hours is not sufficient to evacuate all people to safe area as shown in Figure 4.23e, but with 0.4 hours, all people can evacuate safely as illustrated in Figure 4.23f. Despite the evacuation time limit, mean safety of agents over generation in Figures 4.23e and 4.23f show that optimization with previous diverse population has better performance over others because this diverse population has one hidden solution which is the best path to safe area.

From Figure 4.23 and 4.24, we can see that graphs of DCpall, ESpall, DCpone and ESpone are above the graphs of ES and DC. This shows that when we start the evacuation of city with previous solutions as starting population in the condition like bridges collapse,

safe areas change and agents do not following instructions; they perform better than just starting with random population. This is because they have prior knowledge of which routes to follow to reach safe places, so they have good safety value even at early generations. All these experiments help us to find answer to question Q2. We will also use these experiments to answer question Q3.

We now perform the experiments to run evolutionary algorithms using solutions (probabilities set) from previous optimization for a fewer number of generations (10 generations). Again, we use TC1b5, TC1c, TC3d and TC4b to run for only 10 generations. We have chosen these test cases to address all possible problems like bridges collapse (TC1b5 and TC4b), safe areas change (TC1c) and people not following instructions (TC3d). We run algorithms in all adaptive algorithms defined in Section 4.2 and see how all adaptive algorithms behave in these cases. Table 4.10 and Figure 4.25 show the result.

Table 4.10: Mean safety of agents obtained with best individual in population from last generation over simulation hour for test cases after running ES, DC, ES using all initial probabilities from previous ES optimization (ESpall), ES using all initial probabilities from previous DC optimization (DCpall), ES using initial probabilities from best probabilities of previous ES optimization (ESpone) and ES using initial probabilities from best probabilities of previous DC optimization (DCpone). Parenthesis indicates the standard deviation of safety values. Mean and standard deviation are calculated from optimization carried out for 30 repetitions. Optimization done only for 10 generations. The best result (column) is highlighted in bold.

(a) TC1b5

Simulation Hour [hr]	Mean Safety with ES	Mean Safety with DC	Mean Safety with ESpall	**Mean Safety with DCpall**	Mean Safety with ESpone	Mean Safety with DCpone
0	0 (0)	0 (0)	0 (0)	**0 (0)**	0 (0)	0 (0)
0.04	0 (0)	0 (0)	0 (0)	**0 (0)**	0 (0)	0 (0)
0.08	0.068 (0.024)	0.037 (0.012)	0.075 (0.016)	**0.107 (0.029)**	0.069 (0.017)	0.085 (0.020)
0.12	0.261 (0.042)	0.144 (0.037)	0.332 (0.031)	**0.407 (0.028)**	0.309 (0.036)	0.320 (0.035)
0.16	0.401 (0.045)	0.221 (0.051)	0.508 (0.042)	**0.615 (0.031)**	0.486 (0.034)	0.484 (0.042)
0.2	0.579 (0.049)	0.332 (0.058)	0.741 (0.042)	**0.836 (0.022)**	0.715 (0.035)	0.675 (0.043)
0.24	0.669 (0.047)	0.397 (0.060)	0.842 (0.031)	**0.910 (0.015)**	0.813 (0.029)	0.759 (0.036)
0.28	0.773 (0.041)	0.481 (0.060)	0.924 (0.017)	**0.964 (0.011)**	0.904 (0.019)	0.851 (0.027)
0.32	0.825 (0.036)	0.531 (0.058)	0.954 (0.011)	**0.981 (0.006)**	0.939 (0.013)	0.894 (0.021)
0.36	0.883 (0.030)	0.591 (0.056)	0.979 (0.005)	**0.994 (0.003)**	0.971 (0.007)	0.937 (0.013)
0.4	0.909 (0.026)	0.625 (0.056)	0.988 (0.003)	**0.998 (0.001)**	0.983 (0.005)	0.957 (0.010)

(b) TC1c

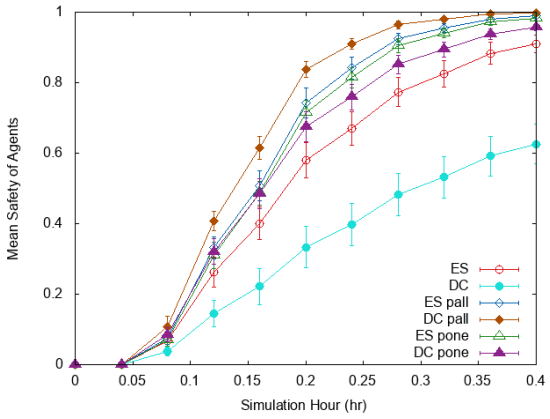
Simulation Hour [hr]	Mean Safety with ES	Mean Safety with DC	Mean Safety with ESpall	**Mean Safety with DCpall**	Mean Safety with ESpone	Mean Safety with DCpone
0	0 (0)	0 (0)	0 (0)	**0 (0)**	0 (0)	0 (0)
0.04	6.666 (0.000)	0 (0)	0 (0)	**0 (0)**	0 (0)	0 (0)
0.08	0.053 (0.014)	0.030 (0.010)	0.064 (0.013)	**0.114 (0.026)**	0.065 (0.012)	0.104 (0.013)
0.12	0.220 (0.036)	0.125 (0.033)	0.299 (0.042)	**0.445 (0.046)**	0.305 (0.030)	0.424 (0.025)
0.16	0.347 (0.044)	0.198 (0.046)	0.477 (0.046)	**0.651 (0.049)**	0.492 (0.034)	0.622 (0.036)
0.2	0.516 (0.052)	0.290 (0.051)	0.709 (0.040)	**0.855 (0.031)**	0.729 (0.032)	0.817 (0.032)
0.24	0.606 (0.051)	0.344 (0.054)	0.822 (0.031)	**0.924 (0.019)**	0.837 (0.024)	0.898 (0.023)
0.28	0.711 (0.050)	0.421 (0.058)	0.918 (0.019)	**0.973 (0.009)**	0.930 (0.014)	0.961 (0.011)
0.32	0.766 (0.050)	0.455 (0.061)	0.951 (0.012)	**0.986 (0.005)**	0.960 (0.010)	0.979 (0.006)
0.36	0.826 (0.047)	0.506 (0.065)	0.980 (0.006)	**0.997 (0.001)**	0.984 (0.005)	0.992 (0.003)
0.4	0.859 (0.043)	0.537 (0.062)	0.988 (0.003)	**1 (0)**	0.993 (0.002)	0.998 (0.001)

(c) TC3d

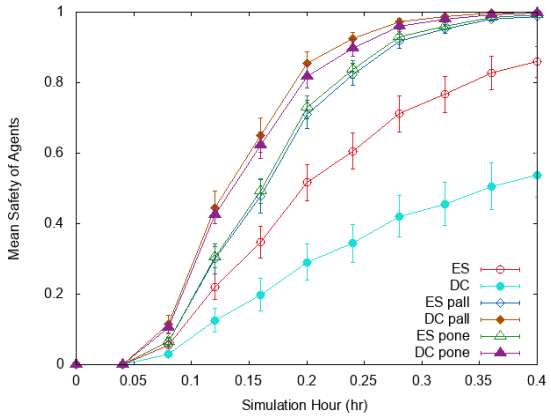
Simulation Hour [hr]	Mean Safety with ES	Mean Safety with DC	Mean Safety with ESpall	**Mean Safety with DCpall**	Mean Safety with ESpone	Mean Safety with DCpone
0	0 (0)	0 (0)	0 (0)	**0 (0)**	0 (0)	0 (0)
0.04	0.176 (0.028)	0.116 (0.028)	0.264 (0.022)	**0.254 (0.032)**	0.245 (0.011)	0.243 (0.014)
0.08	0.322 (0.045)	0.198 (0.040)	0.481 (0.024)	**0.470 (0.033)**	0.457 (0.014)	0.470 (0.011)
0.12	0.500 (0.048)	0.296 (0.048)	0.778 (0.029)	**0.756 (0.033)**	0.753 (0.016)	0.779 (0.007)
0.16	0.594 (0.042)	0.353 (0.049)	0.883 (0.023)	**0.865 (0.024)**	0.862 (0.016)	0.880 (0.005)
0.2	0.707 (0.037)	0.429 (0.047)	0.953 (0.014)	**0.943 (0.013)**	0.944 (0.010)	0.942 (0.005)
0.24	0.764 (0.035)	0.472 (0.048)	0.973 (0.009)	**0.969 (0.008)**	0.969 (0.007)	0.964 (0.003)
0.28	0.828 (0.029)	0.535 (0.049)	0.987 (0.005)	**0.988 (0.004)**	0.987 (0.005)	0.982 (0.001)
0.32	0.861 (0.027)	0.569 (0.048)	0.993 (0.002)	**0.994 (0.002)**	0.992 (0.003)	0.987 (0.002)
0.36	0.895 (0.021)	0.616 (0.048)	0.996 (0.001)	**0.998 (0.001)**	0.996 (0.001)	0.995 (0.000)
0.4	0.912 (0.020)	0.643 (0.046)	0.999 (0.000)	**1 (0)**	0.999 (0.000)	1 (0)

(d) TC4b

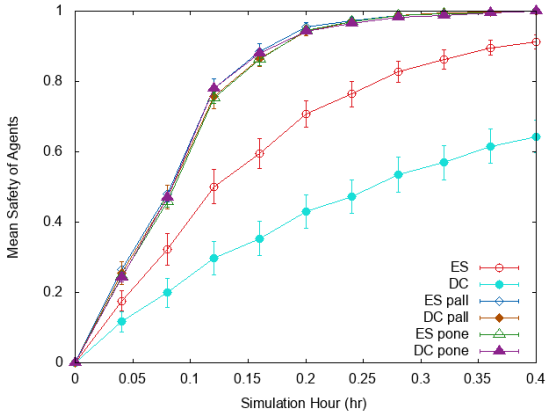
Simulation Hour [hr]	Mean Safety with ES	Mean Safety with DC	Mean Safety with ESpall	**Mean Safety with DCpall**	Mean Safety with ESpone	Mean Safety with DCpone
0	0 (0)	0 (0)	0 (0)	**0 (0)**	0 (0)	0 (0)
0.04	0 (0)	0 (0)	0 (0)	**0 (0)**	0 (0)	0 (0)
0.08	0 (0)	0 (0)	3.333 (0.000)	**3.333 (0.000)**	0 (0)	6.666 (0.000)
0.12	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	**0.002 (0.001)**	0.000 (0.000)	0.000 (0.000)
0.16	0.004 (0.002)	0.000 (0.000)	0.002 (0.001)	**0.018 (0.005)**	0.001 (0.001)	0.001 (0.001)
0.2	0.024 (0.009)	0.003 (0.002)	0.012 (0.006)	**0.088 (0.018)**	0.007 (0.003)	0.004 (0.002)
0.24	0.054 (0.019)	0.007 (0.005)	0.027 (0.012)	**0.164 (0.028)**	0.015 (0.006)	0.008 (0.004)
0.28	0.115 (0.033)	0.017 (0.007)	0.061 (0.023)	**0.298 (0.037)**	0.036 (0.013)	0.014 (0.007)
0.32	0.168 (0.045)	0.027 (0.011)	0.092 (0.031)	**0.402 (0.041)**	0.056 (0.019)	0.021 (0.009)
0.36	0.247 (0.059)	0.045 (0.017)	0.143 (0.040)	**0.536 (0.043)**	0.091 (0.029)	0.031 (0.014)
0.4	0.305 (0.070)	0.059 (0.020)	0.179 (0.048)	**0.622 (0.048)**	0.118 (0.037)	0.040 (0.019)



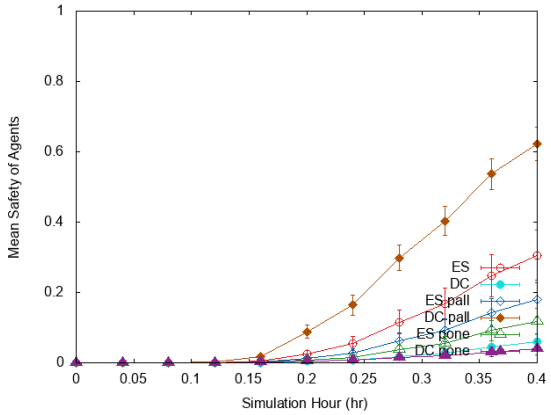
(a) TC1b5 for 10 generations



(b) TC1c for 10 generations



(c) TC3d for 10 generations



(d) TC4b for 10 generations

Figure 4.25: Mean safety of agents obtained with best individual in population from last generation over simulation hour for test cases after running ES, DC, ES using all initial probabilities from previous ES optimization (ESpall), ES using all initial probabilities from previous DC optimization (DCpall), ES using initial probabilities from best probabilities of previous ES optimization (ESpone) and ES using initial probabilities from best probabilities of previous DC optimization (DCpone). Error bars are standard deviation of safety values. Mean and standard deviation are calculated from optimization carried out for 30 repetitions. Optimization done only for 10 generations.

From Table 4.10 and Figure 4.25, it is clear that for fewer number of generations, evolutionary algorithms using starting probabilities set generated from previous optimization produce better result than using random probabilities set. This shows that previous knowledge (gene) is useful to adapt in new changes very quickly in few generations. So, the transfer learning can be used in city evacuation problem and is really helpful for re-optimization in new changes. This gives us answer to question Q2.

4.3.3 Experiments for Question Q3

Question Q3 is how diversity in population influences the transfer learning. If we have diverse solutions in the population from previous optimization for a city, then are they helpful in evacuation of same city with some problems like bridges collapse, safe areas change, etc.,?

To answer question Q3, we use experiments from Figure 4.23. In Figure 4.23, we can see that the previous diverse solutions generated using DC optimization are very useful for re-optimization of city in the condition when bridges collapse, safe areas change and people not following instructions. The graphs of evolutionary algorithms run with the previous DC optimized population (DCpall) are at the top of other graphs. This helps us in finding answer to question Q3 about how diversity influences transfer learning. So, all these experiments show that diverse solutions from previous optimization play a crucial role in re-optimization of city with new problems.

Also, we can notice that pall algorithms are better than pone algorithms in respective cases. For example, DCpall gives better result than DCpone and ESpall gives better result than ESpone.

4.4 Diversity Measure and Control

We have measured the diversity in population with ES and DC algorithms. Figure 4.26 shows the diversity in population with ES and DC algorithms. From the graphs in Figure 4.26, we can see that the diversity is decreasing over generations for Evolution Strategy (ES) algorithm, but the diversity is preserved for Deterministic Crowding (DC) algorithm. Deterministic Crowding (DC) algorithm prevents premature convergence and generates diverse solutions controlling the diversity in the population over generations. For measuring diversity, we calculate the variation in each gene (i.e. probability value of each edge) of all individuals in the population. We determine the mean for each gene in the population, then calculate standard deviation for each gene. Finally, add standard deviation of all genes and

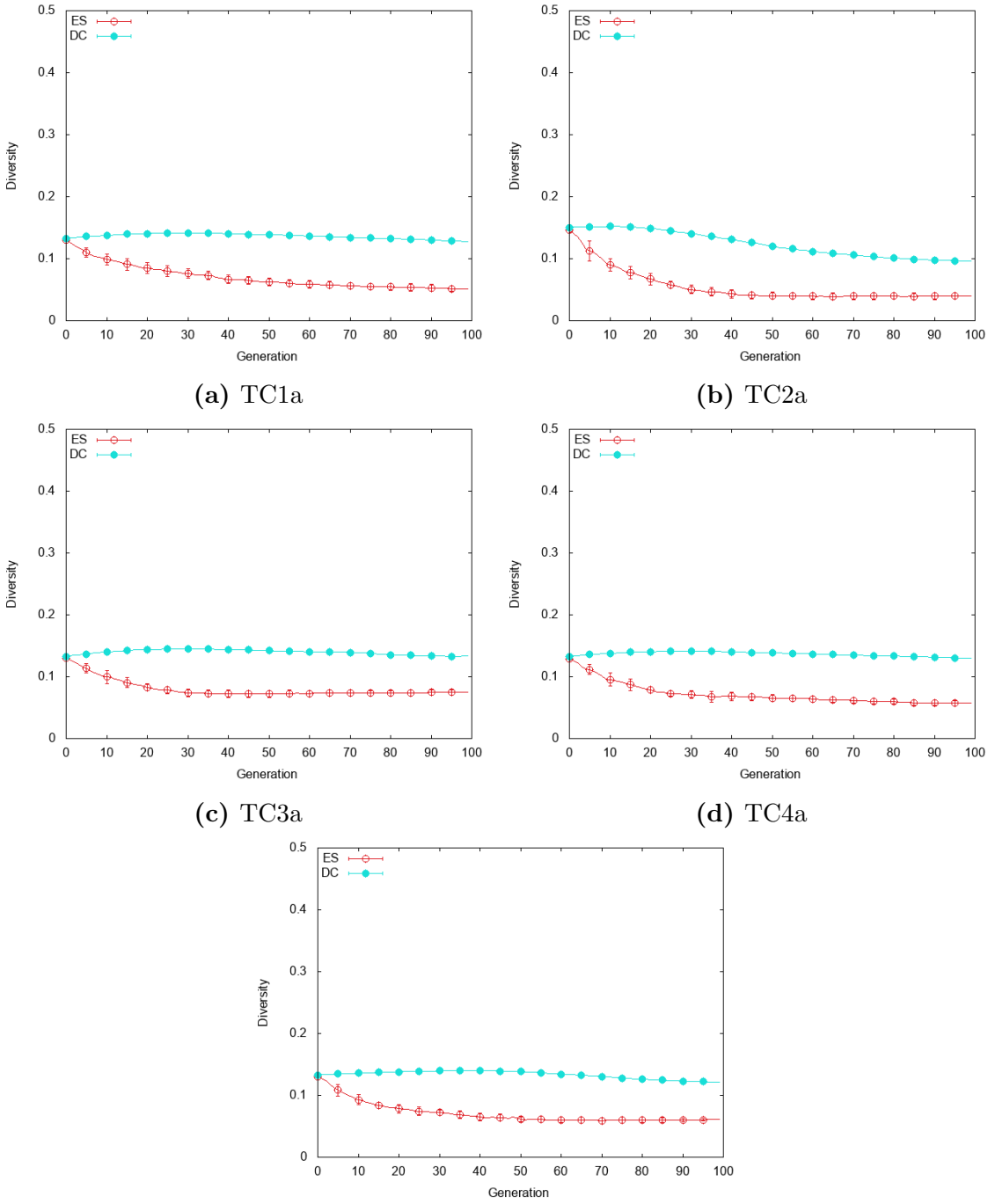
divide by total number of genes (edges).

Suppose X is an individual in population P of size N with $x_1, x_2, x_3, ...x_M$ genes. Individual has gene index as $j = 1, 2, 3, ..., M$ and population has index as $i = 1, 2, 3, ..., N$. Then, mean (μ) of a gene x_j is given by Equation 4.1 and standard deviation (σ) of a gene x_j is given by Equation 4.2. We calculate mean and standard deviation for all the genes. Then, we find diversity in population using formula in Equation 4.3.

$$\mu_j = \frac{1}{N} \sum_{i=1}^{N} x_{ij} \tag{4.1}$$

$$\sigma_j = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_{ij} - \mu_j)^2} \tag{4.2}$$

$$diversity = \left(\sum_{j=1}^{M} \sigma_j \right) / M \tag{4.3}$$



(a) TC1a

(b) TC2a

(c) TC3a

(d) TC4a

(e) TC5a

Figure 4.26: Diversity in population over generation with ES and DC for test cases.

Chapter 5: Conclusions

From the experiments conducted and analysis done on the results, we conclude that evolutionary algorithms (i.e. ES, DC) can solve the city evacuation problem in an efficient and robust way even in the changing conditions of disaster like bridges collapse, sudden change in the location of safe areas or people not following instructions.

When transfer learning is used in evolutionary algorithms, they are very effective to produce better results than just starting with random initial solutions while reoptimizing the city in the changing conditions of disaster. Transfer learning helps to generate robust solutions for the city evacuation plan in short amount of time in fewer number of generations.

Also, transfer learning with previous diverse population using Deterministic Crowding (DC) to preserve population diversity gave better results than previous population using Evolution Strategy (ES) or random population . However, Figure 4.23c suggests that when there is huge changes in the topology of city like change in location of safe areas from one corner of the city to other corner, then it is good to use ES with random probabilities.

5.1 Future Work

There are some tasks that can be carried out as future work in this research. Some of them are as follows:

1. Self evolving sigma value for using in mutation. Now, we are using mutation step size as 0.1, but it can be made self adaptive.

2. Spatially aware crossover that knows the network and reduces the disruptive effects of crossover.

3. Network aware mutation which can understand that most streets are two way and so evacuation traffic is usually going in only one direction. So, mutation should not create cases where individuals are moving in both directions.

4. Co-evolution between simulation and problem that could exist during evacuation.

5. Currently, we only use people's safety as a fitness function. But this can be modified to multi-objective optimization.

REFERENCES

[Abdelgawad and Abdulhai, 2009] Abdelgawad, H. and Abdulhai, B. (2009). Emergency evacuation planning as a network design problem: a critical review. *Transportation Letters*, 1(1):41–58.

[Brown, 1988] Brown, R. (1988). Calendar queues: A fast 0(1) priority queue implementation for the simulation event set problem. *Commun. ACM*, 31(10):1220–1227.

[Bureau of Public Roads, 1964] Bureau of Public Roads, U. S. (1964). *Traffic assignment manual for application with a large, high speed computer*. Traffic Assignment Manual for Application with a Large, High Speed Computer. U.S. Dept. of Commerce, Bureau of Public Roads, Office of Planning, Urban Planning Division.

[Chalmet et al., 1982] Chalmet, L. G., Francis, R. L., and Saunders, P. B. (1982). Network models for building evacuation. *Management Science*, 28(1):86–105.

[Coutinho-Rodrigues et al., 2016] Coutinho-Rodrigues, J., Sousa, N., and Natividade-Jesus, E. (2016). Design of evacuation plans for densely urbanised city centres. *Proceedings of the Institution of Civil Engineers - Municipal Engineer*, 169(3):160–172.

[De Jong, 1975] De Jong, K. (1975). An analysis of the behavior of a class of genetic adaptive systems.

[Drew et al., 2017] Drew, K. J., Heckendorn, R. B., Abdel-Rahim, A., Marisetty, H. P. K., and Stalick, A. (2017). Evolving a real-time evacuation for urban disaster management. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, pages 1089–1096, New York, NY, USA. ACM.

[Eiben and Schoenauer, 2002] Eiben, A. and Schoenauer, M. (2002). Evolutionary computing. *Information Processing Letters*, 82(1):1 – 6. Evolutionary Computation.

[Eiben and Smith, 2015] Eiben, A. E. and Smith, J. E. (2015). *Introduction to Evolutionary Computing*. Springer Publishing Company, Incorporated, 2nd edition.

[Lu et al., 2005] Lu, Q., George, B., and Shekhar, S. (2005). Capacity constrained routing algorithms for evacuation planning: A summary of results. In Bauzer Medeiros, C., Egenhofer, M. J., and Bertino, E., editors, *Advances in Spatial and Temporal Databases*, pages 291–307, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Mahfoud, 1992] Mahfoud, S. W. (1992). Crowding and preselection revisited. *Parallel Problem Solving from Nature*, pages 27–36.

[Manzo et al., 2013] Manzo, S., Nielsen, O., and Prato, C. (2013). Investigating uncertainty in bpr formula parameters: a case study.

[Pan and Yang, 2010] Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359.

[Pel et al., 2010] Pel, A. J., Hoogendoorn, S. P., and Bliemer, M. C. (2010). Evacuation modeling including traveler information and compliance behavior. *Procedia Engineering*, 3:101 – 111. 1st Conference on Evacuation Modeling and Management.

[Pelechano and Badler, 2006] Pelechano, N. and Badler, N. I. (2006). Modeling crowd and trained leader behavior during building evacuation. *IEEE Computer Graphics and Applications*, 26(6):80–86.

[Rechenberg, 1973] Rechenberg, I. (1973). *Evolutionsstrategie: optimierung technischer systeme nach prinzipien der biologischen evolution*. Frommann-Holzboog.

[Saadatseresht et al., 2009] Saadatseresht, M., Mansourian, A., and Taleai, M. (2009). Evacuation planning using multiobjective evolutionary optimization approach. *European Journal of Operational Research*, 198(1):305 – 314.

[Schwefel, 1995] Schwefel, H.-P. (1995). *Evolution and Optimum Seeking*.

[Stepanov and Smith, 2009] Stepanov, A. and Smith, J. M. (2009). Multi-objective evacuation routing in transportation networks. *European Journal of Operational Research*, 198(2):435 – 446.

[Yin, 2009] Yin, D. (2009). A scalable heuristic for evacuation planning in large road network. In *Proceedings of the Second International Workshop on Computational Transportation Science*, IWCTS '09, pages 19–24, New York, NY, USA. ACM.

[Yuan and Han, 2010] Yuan, F. and Han, L. D. (2010). A multi-objective optimization approach for evacuation planning. *Procedia Engineering*, 3:217 – 227. 1st Conference on Evacuation Modeling and Management.