OPTIMIZATION OF DAMPERS IN VISCOUSLY DAMPED MOMENT FRAMES UTILIZING MOPSO

A THESIS

Presented to the Department of Civil Engineering and Construction Engineering Management

California State University, Long Beach

In Partial Fulfillment of

the Requirements for the Degree of

Master of Science in Civil Engineering

Committee Members:

Vesna Terzic, Ph.D. (Chair)

Andrea Calabrese, Ph.D.

Mehran Rahmani, Ph.D.

College Designee:

Hamid Rahai, Ph.D.

By Mahmoud Baei

M.S., 2012, University of Tehran

August 2019

ABSTRACT

New technological developments in engineering present the opportunity for improved efficiency in structural design. Increased number and capacity of Central Processing Units (CPUs) reduce the time needed for computational calculations. Concurrently, optimization algorithms have been evolved for solving complicated problems including multi-objective goals, constraints, and boundaries. Effective optimization algorithms along with the availability of high-performance computers have been a central focus within contemporary engineering.

This thesis aims to develop the framework for optimal design of viscously damped moment frames (VDMFs) considering a set of seismic engineering demand parameters (EDPs) and the multi-objective particle swarm optimization (MOPSO) algorithm. The study provides recommendations on how to efficiently generate reliable solution set utilizing MOPSO. The provided recommendations include the following: 1) modification of original MOPSO to include constraint functions, 2) selection of cost functions, and 3) selection of MOPSO input parameters.

Furthermore, this research study develops efficient optimization strategy for finding optimal design solutions in the absence of prior engineering knowledge about suitable damper properties that would meet design requirements. The proposed optimization strategy allows utilization of broad ranges of damping coefficients, the primary damper property that governs the design and therefore the primary decision variable, to efficiently find the optimal solutions (i.e., Pareto front) under the set of design objectives and constraints. The proposed strategy is verified utilizing an engineered solution of a viscously-damped moment frame.

ACKNOWLEDGEMENTS

I would like to sincerely express my apperception to Dr. Vesna Terzic for providing the valuable opportunity to work in her research lab. Without her enormous patience and her willingness to share her vast expertise, this thesis would not have been accomplished.

Thank you to my family, especially my mother and father for your encouragement and wisdom throughout the years.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

New technological developments in engineering present the opportunity for improved efficiency in structural design. Increased number and capacity of Central Processing Units (CPUs) reduce the time needed for computational calculations. Concurrently, optimization algorithms have been evolved for solving complicated problems including multi-objective goals, constraints, and boundaries. The mixture of effective optimization algorithms along with the availability of high-performance computers has been a central focus within contemporary engineering.

The optimization algorithms have consistently evolved since their conception. Originally simple heuristic algorithms such as the genetic algorithm (GA) (Holland 1975) have improved to more complicated metaheuristic algorithms such as particle swarm optimization (PSO) (Kennedy and Eberhart 1995). In addition, the region-based selection methods of multi-objective optimization processes, such as Pareto Envelope based Selection Algorithm (PESA) (Knowles and Corne 2000), has evolved alongside of optimization algorithms.

High-performance computers, in conjunction with the optimization algorithms, enable the solving of sophisticated multi-objective non-convex problems in a timely, accurate, and effective manner. This combination of technological advancement and improved optimization algorithms can be utilized in future software generations.

This thesis develops a framework, which considers optimization algorithm and high-performance computing to propose an effective and advanced method for optimization of viscously damped moment frames.

1.1 Background, Objectives, and Scope

The research background is presented in three sections: 1) review of devices for control of structures for earthquake actions; 2) review of multi-objective optimization methods; and 3) review of the recent advancements in optimized design or retrofitting schemes of structures that employ supplemental damping system.

1.1.1 The Control of Structures

Structural performance in an earthquake is considered acceptable if the structure does not collapse during a major earthquake and the occupants can evacuate safely. The life safety level is the sufficient level of performance in design codes and is the basis for the modern seismic provisions. Nevertheless, for critical structures such as hospitals, it is imperative to achieve higher performance levels while maintaining cost-efficiency.

Much research has been conducted to develop innovative, earthquake-resistant systems and to improve the performance level and cost-efficiency of structures. The majority of these systems utilize supplemental damping mechanisms or isolator systems to dissipate the energy and to control the structural vibrations resulting from an earthquake. The supplemental damping and isolation systems can be categorized as shown in Table 1. Among seismic protection systems, the passive supplementary damping systems have been investigated by many researchers and professional practitioners as they provide cost-effective structural solution. Compared to the active supplementary damping systems, their advantage is that they do not require a source of external power.

The passive dissipating systems can be divided into three different categories: 1) the displacement-activated devices, 2) velocity-activated devices, and 3) motion-activated devices (see Table 2).

TABLE 1. Seismic Protection Systems (Filiatrault and Christopoulos 2006)

Supplemental Damping Systems		Isolation Systems
Metallic	Tuned-mass	Elastomeric bearings
Friction	Self-centering	Lead-rubber bearings
Viscous	Bracing systems	High-damping rubber
Viscoelastic	Tuned-mass	Metallic
Piezo-elastic	Variable stiffness	Lead-extrusion
Rheological	Variable damping	Friction Pendulum

The passive energy dissipating systems are added to structures in order to perturb the energy balance during earthquakes. If the passive energy dissipation is well designed, then the perturbed energy can be beneficial to the structure. The passive energy dissipators essentially change the mass, stiffness, or damping of the structure in order to reduce the inelastic energy dissipation demand on the structures (Constantinou and Symans 1993).

TABLE 2. Passive Energy Dissipating Systems (Filiatrault and Christopoulos 2006)

Displacement-Activated	Velocity-Activated	Motion-Activated
Metallic	Viscous	Tuned-mass
Friction		
Self-centering		
Viscoelastic	Viscoelastic	

1.1.2 Multi-Objective Optimization Approaches

Engineers are interested in implementing evolutionary algorithms for solving complex engineering problems. Such algorithms were inspired by nature. Between the algorithms, the multi-objective problems (MOPs) are capable of solving sophisticated problems, which have multiple goals, constraints, and boundaries in iterations.

The most prevalent multi-objective evolutionary algorithms are: the Multi-Objective Genetic Algorithm (MOGA) (Fonseca and Fleming 1993), the Non-Dominated Sorting Genetic Algorithm's third version (NSGA-III) (Deb and Jain 2013), the Classic and Intelligent Portfolio Optimization (Vedarajan 1997), the Multi-Objective Evolutionary Algorithm based on Decomposition (MOEA/D) (Zhang and Li 2007), the Strength Pareto Evolutionary Algorithm, the Pareto Envelope-based Selection Algorithm's second version (PESA-II) (Corne et al. 2001), and the Multi-Objective Particle Swarm Optimization (MOPSO) (Coello and Lechuga 2002). Among the multi-objective optimization algorithms, MOPSO effectively solves engineering problems because it utilizes an adaptive grid of PESA for maintaining the diversity of solutions as well as the small number of starting populations (Coello and Lechuga 2002). The small number of populations would benefit when using secondary software for running engineering problems because with smaller populations there is shorter runtime. MOPSO is described in detail in Chapter 2.

1.1.3 Recent Research in Optimization of Structures with Viscous Dampers

Optimization for design or retrofitting of systems with dampers in topology, single-objective versus multi-objective, local versus global, and structural response versus building performance for objective functions have been of significant interest to engineers and researchers in recent years.

Wang and Mahin (2018a) proposed an automated method for the retrofitting of an existing 35-story building; the method used fluid viscous dampers (FVDs). Their method depends on software advancements in nonlinear dynamics analysis, performance-based evaluation, management of workflow, and computational parallel processing computers. The optimization procedure applies the Performance-Based Earthquake Evaluation (PBEE) framework in conjunction with finite element software OpenSees (McKenna and Fenves 2004). In order to find the optimal solution considering one objective function and update the design variables a gradient-based algorithm available in OpenSees is used. The goal of their method was to design FVDs for improved seismic behavior of structures with the goal of avoiding collapse at the basic safety earthquake event (BSE-2E of ASCE 41, 2013). The objective function considered was the building's total loss, derived from fragility functions that relate building interstory drifts and floor accelerations to damage of building components. The results improved analytical efficiency, obtaining solutions within two to four iterations.

Furthermore, Wang and Mahin (2018b) compared three types of supplementary energy dissipaters for the existing 35-story building (Wang and Mahin 2018a): fluid viscous dampers (FVDs), viscous wall dampers (VWDs), and buckling restrained braces (BRBs). All three types of considered devices had the same location within the existing 35-story structure. The simplified modal analysis was used to find the mechanical characteristics of the devices for obtaining the equal effective total damping, as well as the consistency between the story drifts for a targeted collapse prevention performance level. The project showed buildings with the FVD retrofitted scheme were more reliable at achieving the target performance levels.

More recently, Akcelyan and Lignos (2018) implemented oil dampers in a retrofit scheme of a 40-story steel moment resisting fame that was designed in the 1970s in North America. They used nine retrofit schemes, which were proposed for three damping levels, and three vertical distribution methods of shear forces (direct, effective, and balanced). The nonlinear analyses were conducted in compliance with the standard for Seismic Evaluation and Retrofit of Existing Buildings (American Society of Civil Engineers 2013). The authors recommended that the supplementary damping devices would decrease the risk of collapse. Moreover, the results revealed that the vertical damping was influenced by the frame inelasticity and the supplementary damping levels.

Del Gobbo et al. (2018) performed research on performance-based seismic design (PBSD) assessment for FVDs-retrofitted buildings. Six damper placements were considered and structural and nonstructural repair costs were evaluated in accordance with FEMA P-58 (FEMA [Federal Emergency Management Agency] 2012). Six placements were determined utilizing the following six methods: uniform damping, stiffness proportional damping, story shear strain energy method, efficient story shear strain energy method, simplified sequential search algorithm, and fully stressed design algorithm. The results showed that none of the placements reached an optimal solution. It was revealed that the iterative methods optimized the local parameters; the performance of each independent parameter would worsen the optimal solution for other parameters. Among the placements, the story shear strain energy and the uniform damping generated the most efficient repair cost compared to the other methods.

Altieri et al. (2018) proposed a reliability-based approach that considered the intensity of the seismic inputs so as to find the optimal solution of the viscous constant and the velocity exponent. A three-story steel moment resisting frame was considered for this study. The sum of the maximum forces in dampers was considered in the cost function. A high-performance computer with thirty-one cores was used for the analysis. In addition, the starting values for the damping coefficients and velocity exponent were assumed for the analysis. In each literation, both the subset simulation and the auxiliary response method were used for performance-based assessment of the structure. The effective solution was determined by employing the constraint optimization with the linear approximation method.

Pollini et al. (2016) presented an effective method for minimizing the retrofit costs of viscous dampers. The constraints of inter-story drifts of the exterior frames were implemented. The procedure mechanism minimized the costs related to both topology and to the size of the dampers. The optimization was solved as a mixed-integer scheme. In order to improve the efficiency of the solutions, the problem was also formulated and solved in nonlinear programming, using the continuous variables. The results were achieved by utilizing the sequential linear programming algorithm for an asymmetric, three-dimensional, eight-story, three bays frame, and were compared with the results of genetic algorithms. The results showed that the sequential linear programming algorithm converged to a solution that required less effort than did genetic algorithms.

Castaldo and De Iuliis (2014) investigated the optimal solution for seismic design of structural and viscoelastic bracing-damper systems with consideration of displacement-based seismic design. The integrated energy-based optimization approach was used. The method was validated by evaluating the average displacement of time-history responses of seven unscaled records, selected according to Eurocode 8 (CEN [Comité Européen de Normalisation] 2005) provisions. Results showed that the viscoelastic damper was convenient to use for structures with high periods (longer than 0.5s). The optimal solution was achieved using viscoelastic dampers rather than the lateral stiffness of structural systems.

Extensive research has been conducted to determine the optimal seismic design solutions of dampers in moment frames utilizing different optimization techniques, mostly considering a single objective function. This study focuses on optimization of viscous dampers for seismic applications utilizing multi-objective optimization algorithm MOPSO. While Genetic Algorithms can also be used for multi-objective optimization, they are not explored in this study as they typically require larger population size than MOPSO to find the global minimum, requiring longer computation time. MOPSO with its inherently metaheuristic approach avoids getting trapped in the local optimal solutions.

The study provides recommendations on how to efficiently generate reliable solution set utilizing MOPSO. The provided recommendations include the following: 1) modification of original MOPSO to include constraint functions, 2) selection of cost functions, and 3) selection of MOPSO input parameters. Most importantly, this research study develops efficinet optimiziation strategy for finding optimal design solutions in the absence of prior engineering knowladge of suitable damper properties that would meet design requirements. The proposed optimization strategy allows utilization of broad ranges of damping coefficients, the primary damper property that governs the design and therefore the primary decision variable, to efficiently find the optimal solutions (i.e., Pareto front) under the set of design objectives and constraints. The proposed strategy is verified utilizng engineered solution of a viscously-damped moment frame.

1.2 Organization of Research

Chapter 2 of this thesis describes the optimization framework for VDMF, starting with an explanatory overview of MOPSO. Subsequently, Chapter 2 presents the utilization of MOPSO to explore optimal design of dampers. Chapter 2 also describes parallel computing categories, as well as importance of designing with constraints and selection of cost functions. Moreover, an effective approach for attaining optimal solution considering a broad range of damping coefficients is proposed. The proposed approach defines and describes the concept of interdependency between damping coefficients of adjacent floors.

Chapter 3 implements the optimization framework introduced in Chapter 2 on a 3-story VDMF adopted from Terzic and Mahin (2017). Varying the constraints of EDPs and cost functions the quality of optimization solutions of MOPSO and their runtimes are compared. Chapter 3 also explores the runtimes of optimization under circumstances where the ranges of damping coefficients are broadly selected.

Chapter 4 compares the optimization solutions to an engineered solution of a VDMF considering identical set of ground motions.

Chapter 5 concludes the outcomes of multi-objective optimization of VDMFs and proposes recommendations for future studies.

CHAPTER 2

OPTIMIZATION FRAMEWORK OF VDMF

This chapter presents the framework for optimization of viscously damped moment frames (VDMFs) utilizing MOPSO considering the following: different high-performing computation platforms, utilizations of constraints, selection of cost functions, and selection of MOPSO parameters. This chapter further proposes efficient optimization strategy for finding optimal design solution in the absence of prior engineering knowladge of sutable damper properties compliant with the design requirements. This proposed strategy allows utilization of broad ranges of damping coefficients, the primary damper property that governs the design, to efficiently find the optimal solutions (i.e., Pareto front) under the set of design objectives and constraints.

2.1 The Overview of MOPSO

In this study MOPSO is used as a preferred optimization engine. MOPSO is an effective algorithm for solving multi-objective problems. Coello and Lechuga (2002) proposed MOPSO based on the pre-existing particle swarm optimization (PSO). MOPSO utilizes the Pareto envelope and grid-based technique of PESA to handle multi-objective optimization problems (Zitzler et al. 2001). The MOPSO scheme is illustrated in Figure 1.

In MOPSO, the movement of particles is governed by their velocity. The velocity is a function of three terms: inertia, the best personal experience, and the best global experience of swarm movement, as shown in Equation 1. Position is calculated based on the position from a previous iteration and the velocity of the current iteration, as shown in Equation 2. Each parameter in MOPSO is defined in Table 3.

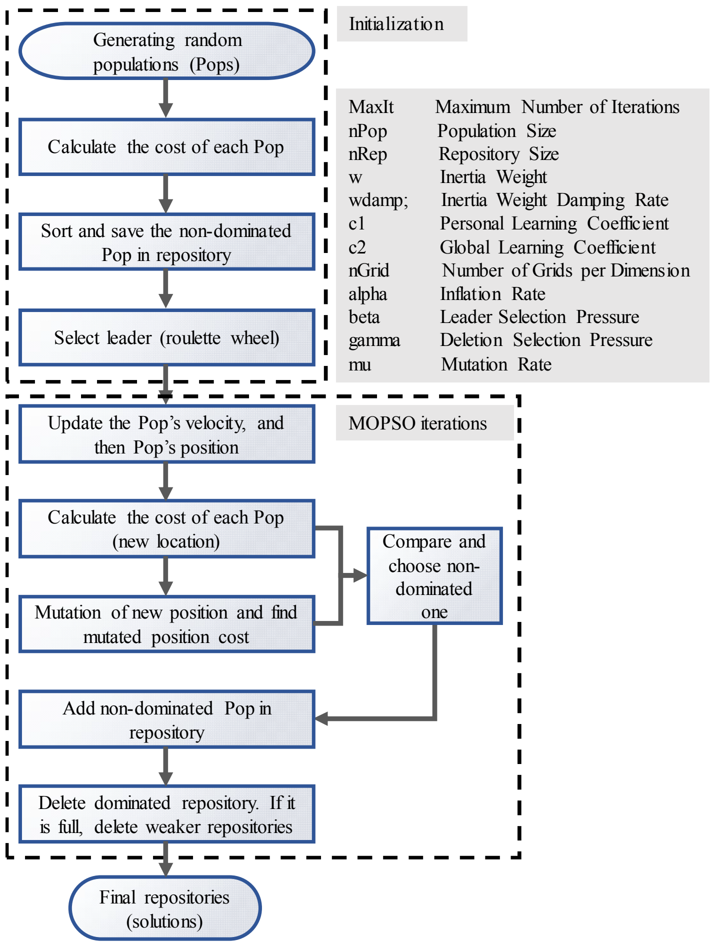
$$V_i^{t+1} = wV_i^t + c_1 r_1 \left(X_{\text{personal best}} - X_i^t\right) + c_2 r_2 \left(X_{\text{global best}} - X_i^t\right) \qquad \text{Equation 1}$$

$X_i^{t+1} = X_i^t + V_i^{t+1}$ Equation 2

In engineering problems, evolutionary algorithms enable engineers to find near-optimal non-dominated solutions. The efficiency of MOPSO depends on the quality of the Pareto front. The quality of the Pareto front is contingent on the parameters considered in multi-objective problems (Fallah-Mehdipour et al. 2011).

TABLE 3. Parameters Associated in MOPSO's Velocity and Position

W	Inertia weight	$X_{global\ best}$	Best global position
c_1	Cognitive acceleration coefficient	V_i^t	Velocity of particle (i), iteration (t)
c_2	Social acceleration coefficient	V_i^{t+1}	Velocity of particle (i), iteration (t+1)
r_1 & r_2	Random number between 0 and 1	X_i^t	Position of particle (i), iteration (t)
$X_{personal\ best}$	Best personal position	X_i^{t+1}	Position of particle (i), iteration (t+1)


Initialization
Generating random populations (Pops)
Calculate the cost of each Pop
Sort and save the non-dominated Pop in repository
Select leader (roulette wheel)
MOPSO iterations
Update the Pop's velocity, and then Pop's position
Calculate the cost of each Pop (new location)
Mutation of new position and find mutated position cost
Compare and choose non-dominated one
Add non-dominated Pop in repository
Delete dominated repository. If it is full, delete weaker repositories
Final repositories (solutions)

MaxIt	Maximum Number of Iterations
nPop	Population Size
nRep	Repository Size
w	Inertia Weight
wdamp;	Inertia Weight Damping Rate
c1	Personal Learning Coefficient
c2	Global Learning Coefficient
nGrid	Number of Grids per Dimension
alpha	Inflation Rate
beta	Leader Selection Pressure
gamma	Deletion Selection Pressure
mu	Mutation Rate

FIGURE 1. MOPSO optimization scheme and its parameters (Coello and Lechuga 2002).

In 2013, a comprehensive survey was conducted, studying the use-cases for MOPSO in engineering. In this survey, 189 articles employed MOPSO for solving multi-objective problems in engineering; Figure 2 shows the distribution of use-cases as of mid-October, 2012 (Lalwani et al. 2013). The statistics showed that 3.7% of MOPSO applications were associated with civil engineering and were mostly related to hydraulics and water engineering.

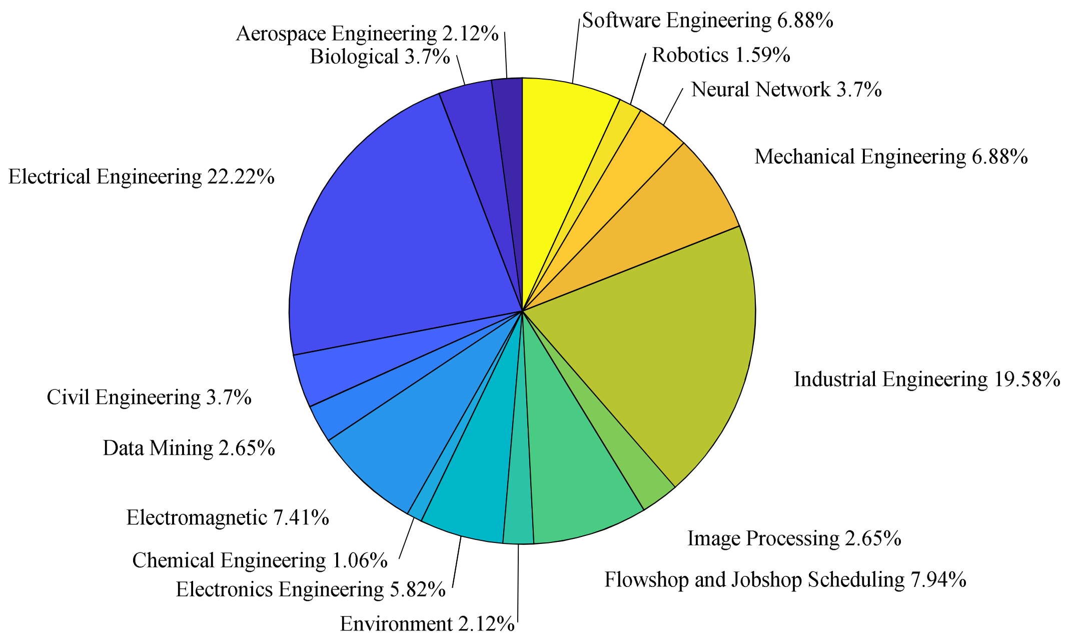
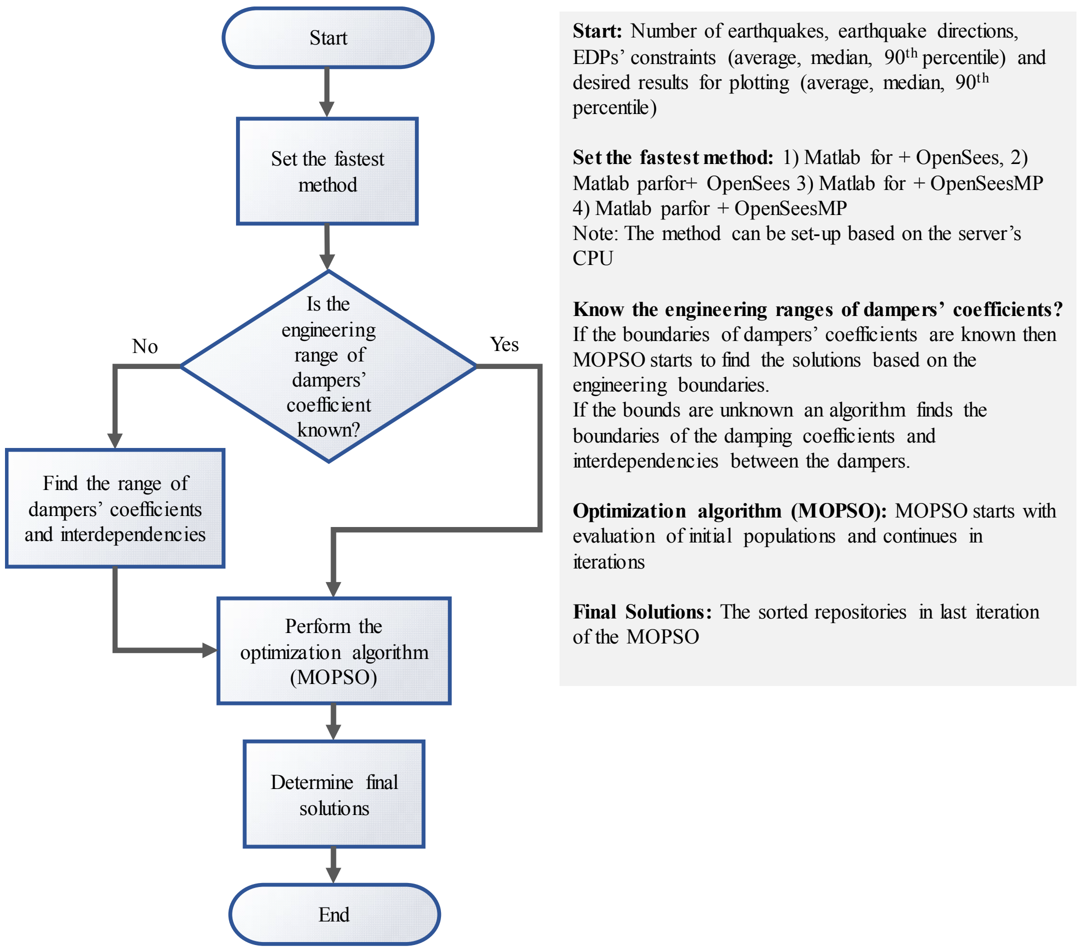


FIGURE 2. The applications of MOPSO in engineering (Lalwani et al. 2013).

2.2 Utilizing MOPSO to Find Optimal Design of Dampers

In present thesis, MOPSO is implemented to the VDMF problem so as to discover the optimal damping coefficients for a set of ground motions. The optimization scheme of VDMF is presented in Figure 3. The MOPSO code used in this thesis was downloaded from Yarpiz Academic Codes and Tutorials (http://yarpiz.com/59/ypea121-mopso). Two parts of the code were modified for use in this thesis for optimization of VDMF: the main code and the cost function (See Appendix A). In the main code, the EDP's constraints, the set of earthquakes, and desired method of results are inputted. In the cost function, the secondary software OpenSees is run for each population. In the cost function, the structural responses' constraints are evaluated, and the final results, after statistical process, are saved as cost of each population.


Start
Set the fastest method
Is the engineering range of dampers' coefficient known?
No
Yes
Find the range of dampers' coefficients and interdependencies
Perform the optimization algorithm (MOPSO)
Determine final solutions
End

Start: Number of earthquakes, earthquake directions, EDPs' constraints (average, median, 90th percentile) and desired results for plotting (average, median, 90th percentile)

Set the fastest method: 1) Matlab for + OpenSees, 2) Matlab parfor+ OpenSees 3) Matlab for + OpenSeesMP 4) Matlab parfor + OpenSeesMP
Note: The method can be set-up based on the server's CPU

Know the engineering ranges of dampers' coefficients? If the boundaries of dampers' coefficients are known then MOPSO starts to find the solutions based on the engineering boundaries.
If the bounds are unknown an algorithm finds the boundaries of the damping coefficients and interdependencies between the dampers.

Optimization algorithm (MOPSO): MOPSO starts with evaluation of initial populations and continues in iterations

Final Solutions: The sorted repositories in last iteration of the MOPSO

FIGURE 3. MOPSO to determine optimal design of dampers scheme.

For the cases where the engineering ranges of dampers are unknown, further steps are required to obtain the appropriate ranges and interdependencies of adjacent dampers, which are presented in Section 2.7 of this chapter.

2.3 Parallel Computing Strategies

Technological developments offer the possibility of new and effective tools for overcoming engineering obstacles that relate to applicability and implementation. The increased computational capacity of CPUs has increased the speed of computational calculations. This thesis develops four optimization platforms to facilitate working with MATLAB in conjunction with the secondary software OpenSees. The different platforms and their advantages and disadvantages are presented in Table 4. Appendix A contains MATLAB codes for the four presented platforms. The runtime of the VDMF optimization largely depends on the number of initial populations and the number of selected earthquakes.

TABLE 4. Different Optimization Platforms Considering Availability of High-Performing Server

Platform	Advantages	Disadvantages
I. MATLAB for + OpenSees	Can be used for any population size and number of earthquakes.	Significant runtime due to sequential analysis for each population and each earthquake.
II. MATLAB parfor + OpenSees	Efficient for a large number of populations.	The analysis is conducted for each earthquake sequentially. If the number of earthquakes is large, it is slower than Platform III. Requires pauses in the MATLAB code to assure that runs for all populations are complete before starting of the next simulation.
III. MATLAB for + OpenSeesMP	Efficient when the number of earthquakes (considering X and Y directions) is larger than the number of populations.	The code calculates cost for each population sequentially (does not utilize parallel processing).
IV. MATLAB parfor + OpenSeesMP	Efficient for cases of small number of earthquakes (considering X and Y directions) and small number of populations	Maximum population size and number of earthquakes depend on the server's capacity. Requires pauses in the MATLAB code. Occupies three additional cores (mpiexec, console windows host, windows command procedure) for communication between MATLAB and OpenSeesMP for any two earthquakes run simultaneously.

To improve the performance of the optimization platforms for particle which cannot complete the process for a pre-specified percentage of earthquakes in OpenSees, the particles are replaced with another particle. The incomplete run is determined based on the end-time of earthquake loading and end-time of the free vibration after the main earthquake have ceased. For an earthquake to be considered complete, both main earthquake and the free vibration need to reach the end of the analysis.

To demonstrate the limitations of the three presented optimization platforms that allow for parallel processes, consider that a user has 40 available CPU cores. The first platform does not have a limit on cores usage. If using Platform II, the maximum number of initial populations in MATLAB would be 30 for parallel processing (30 cores). If using Platform III, a user can simultaneously run maximum of fifteen earthquake in two directions (thirty earthquakes at the same time) utilizing 30 cores. Platform IV allows the user to start with a maximum of six populations and a maximum of two earthquakes simultaneously which reserves 24 cores for parallel processing. The remaining cores were assigned to the servers' applications.

2.4 Design with Constraints

The constraints play an important role in the results of an optimization problem. For the purpose of performance-based earthquake evaluation (PBEE), the most commonly used EDPs are inter-story drifts, floor accelerations, and inter-story residual drifts. A constraint can be applied to an individual EDP, or to any combination of selected EDPs. The constraints typically limit the solution set by improving their quality. Stiff constraints generally increase the runtime because, if any of the constraints is not satisfied, the particle failed to meet the constraint is substituted with another particle. The results of the optimization considering different set of constraints (see Table 8) are discussed in Section 3.2; the chapter then compares the different outcomes caused by these various constraints.

2.5 Selection of Cost Functions

Cost function has a great impact on the quality of the optimization solution set. From the literature review, it is clear that researchers utilized a variety of optimization cost functions; these include only drift (Castaldo and De Iuliis 2014), drift and acceleration (Wang and Mahin 2018a; Del Gobbo 2018), and only damping coefficients (Pollini et al. 2016; Altieri et al. 2018).

In this thesis, numerous cost functions are evaluated and the results are compared in terms of quality of the solutions and runtimes (see Table 10, Section 3.3); based on which recommendations for the selection of the cost functions are provided.

2.6 MOPSO Input Parameters

After selection of the cost function, it is imperative to perform sensitivity analysis for the MOPSO input parameters to improve the outcomes of the optimization. Prior to this study, Thomas (2018) conducted a sensitivity study on MOPSO input parameters considering an elastic single degree-of-freedom (SDOF) system subjected to an earthquake. They have found out that the nGrid parameter had a particularly significant impact on solutions because of its influence in PESA optimization. Furthermore, starting population size and number of iterations have a great influence on the outcomes of optimization. The present thesis uses the values from Thomas (2018) as a starting set of MOPSO parameters with the adjustment of the nGrid parameter. Given that this study focuses of systems that exhibit predominantly nonlinear seismic behavior, sensitivity analysis was performed (see Section 3.4) to propose MOPSO parameters for nonlinear analysis. Table 5 compares the optimized MOPSO parameters for an elastic SDOF system and a nonlinear VDMF.

TABLE 5. MOPSO Input Parameters for an Elastic SDOF System and Nonlinear VDMF

Parameter	Elastic SDOF System (Thomas 2018)	Present Study for Nonlinear Analysis of VDMF
w	0.05	0.05
wdamp	1	1
c1	1.9	1.9
c2	1.9	1.9
nGrid	7	15
alpha	0.01	0.01
beta	1	1
gamma	3	3
mu	0.1	0.1

2.7 Optimization Strategy for Optimal Design of Dampers in the Absence of Engineering Knowledge on Primary Decision Variables

The major goal of this research study was to develop optimiziation strategy for finding optimal design solutions using MOPSO in the absence of prior engineering knowladge of suitable damper properties that would meet design requirements. The proposed optimization strategy allows utilization of broad ranges of damping coefficients, the primary damper property that governs the design and therefore the primary decision variable, to efficiently find the optimal solutions (i.e., Pareto front) under the set of design objectives and constraints.

When the ranges of damping coefficients are broadly selected, the runtime of the original MOPSO optimization is excessive because of randomly generated populations within the broad bounds of the decision variables. In Chapter 3, this phenomenon is discussed and evaluated for the case study.

To improve the efficiency (i.e., speed) of the optimization, the proposed optimization strategy first defines new narower bounds for descision variables within the original bounds by solving for the new minimum and maximum values of damping coefficients that meet constraints imposed on different EDPs; interdepencies of dampers in adjecent stories are also considred when defining the new bounds.

The rationale for redefining the ranges (bounds) of damping coefficients is based on the characteristics of damping coefficients. Equation 3 shows the relationship of damping force with damping coefficient and velocity exponents.

$$f_d(\dot{u})=C_d|\dot{u}|^{\alpha}\text{sign}(\dot{u}) \qquad \text{Equation 3}$$

where f_d is the damping force; $\dot{u}$ is the velocity; C_d is the damping coefficient; and α is the velocity exponet.

It is expected that an increase of the velocity exponent would increase damper acceleration and decrease deformation. To meet a given set of design constraints, the linear viscous damper ($\alpha = 1$) would require smaller damping coefficient, providing the lower bound of damper coefficient. Similarly, lower values of the velocity exponent ($\alpha < 1$) would require higher damping coefficinet, providing the upper bound. Figure 4 depicts behavior of viscous dampers as a funcion of velocity exponent.

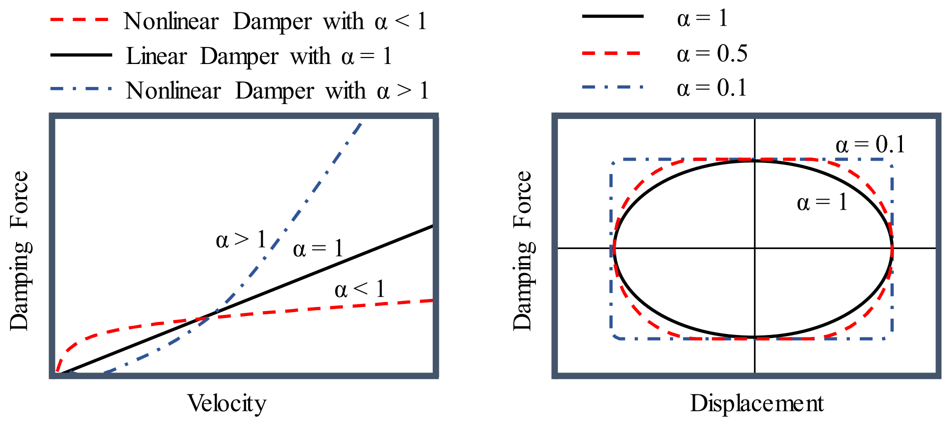
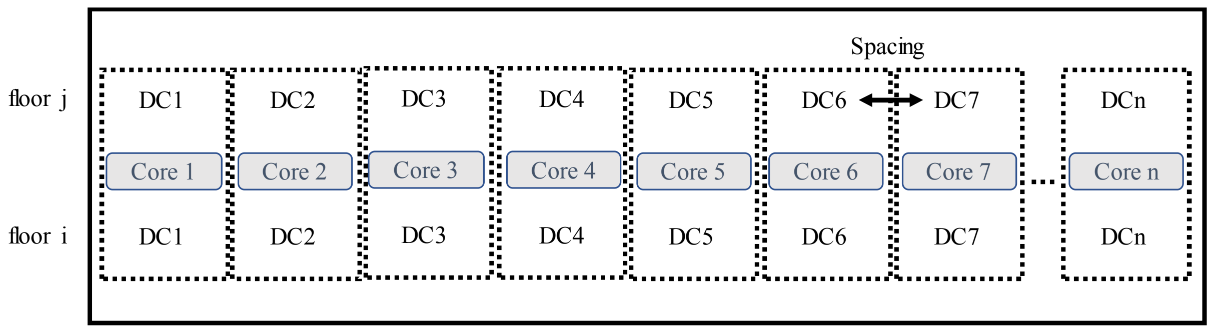


FIGURE 4. Characteristic of damper properties a) damping force vs. velocity b) damping force vs. displacement (Narkhede and Sinha 2014).

To find the new narrower bounds of damping coefficients, the initial broad bounds are split across a number of assigned CPUs. For example, if there are *n* available cores, the initial range of damping coefficient is split to include *n* equidistant damping coefficients. Each equal-distant damping coefficient (with population size equal to the number of assigned CPUs) are assigned to all floors' dampers as shown in Figure 5. Afterwards, two velocity exponents are considered: 1) one that corresponds to nonlinear damper typically used in design (e.g., $\alpha = 0.5$) and 2) one representative of approximatelly linear damper (e.g., $\alpha = 0.8 - 1.0$).

For the predefined population of damping coefficients (with population size equal to the number of assigned CPUs) and for each velocity exponent in turn, the structural analysis of VDMF under a set of constraints imposed on EDPs is performed subjected to ground motions.



Spacing = length of large bracket/(maximum cores-1)
DC = damping coefficient (equaldistant damping coefficient with size of CPU's cores and are assigned to all floors)
i and j are adjacent floors

FIGURE 5. CPU assigning for parallel runs scheme.

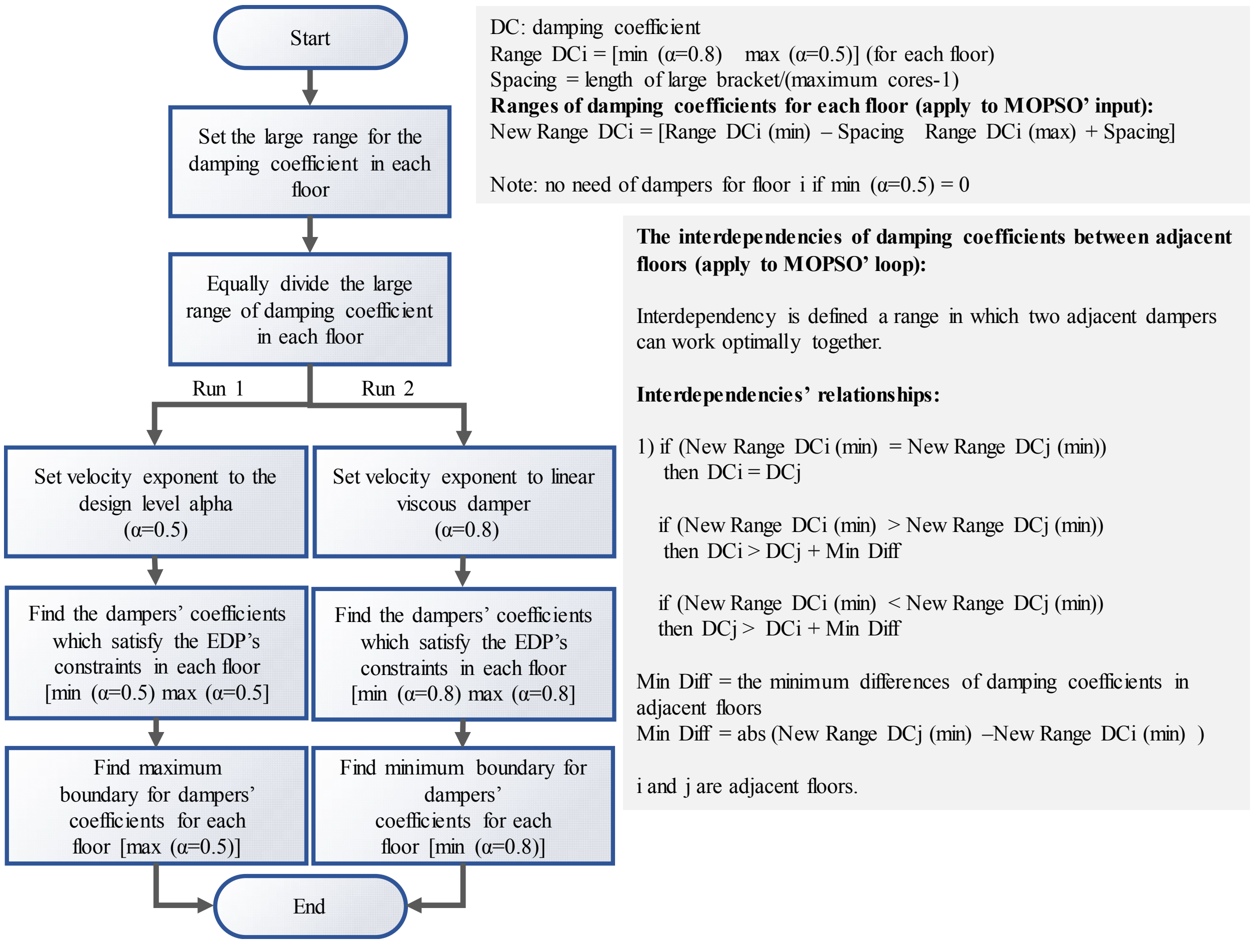
The maximum bound for the new narrow range is then represented by the highest damping coefficient (DC) that meets imposed constraints when nonlinear damper was used (e.g., $\alpha = 0.5$), whereas the minimum bound for the new range is represented by the lowest damping coefficient that meets imposed constraints when linear dampers (e.g., $\alpha = 0.8 - 1.0$) are utilized. For the cases that the acceptable damping coefficient for lower bound of any floor results in a zero value when nonlinear viscous dampers used in design (e.g., $\alpha = 0.5$), there is no need of dampers on the corresponded floor ($DCi = 0$).

After establishing the new range for damping coefficients at each story, the relationships (i.e., interdependences) between dampers at adjacent stories are determined based on the lower new bounds. Even though, in earlier paragraph it was explained that the new bounds are found, however, the acceptable minimums of damping coefficients for adjacent floors can provide additional information regarding the optimal solution for dampers' properties. The minimum differences of the damping coefficients in adjacent floors are obtained from the differences of lower bound of damping coefficients in adjacent floors. The interdependencies of adjacent floors' damping coefficients improve MOPSO to generate the optimal random damping coefficients in new ranges of damping coefficients (DC) for each floor.

For instance, if the new ranges for floor *i* and *j* are [20 60] and [100 160] (kip/(in/s)$^{\alpha}$) respectively, firstly, the new ranges of damping coefficients imply floor *j* requires stronger dampers than floor *i* ($DCi < DCj$). For the given example ($20 < 100$ (kip/(in/s)$^{\alpha}$)) then it can be concluded ($DCi < DCj$). Further, the minimum differences of between optimal damping coefficients of adjacent dampers for any solution would follow ($DCj(min) - DCi(min) = 80$ (kip/(in/s)$^{\alpha}$)). For the given example, if the random damping coefficient of 110 (kip/(in/s)$^{\alpha}$) for floor *j* is proposed by MOPSO, then for the floor *i*, by considering the interdependencies of adjacent floors' dampers, the optimal damping coefficient would be selected from [20 30] (kip/(in/s)$^{\alpha}$) (where $110 - 80 = 30$ (kip/(in/s)$^{\alpha}$)) by MOPSO, instead of selecting from [20 60] (kip/(in/s)$^{\alpha}$).

Interdependencies' relationships can be utilized in MOPSO' loops whenever it requires generating random solutions for adjacent floors' damping coefficients. These relationships would decrease MOPSO runtime. Figure 6 shows the proposed methodology for finding new ranges of damping coefficients and their interdependencies of adjacent dampers. Appendix B contains Matlab code for parallel processing of finding the narrow new bounds from the broad bands as well as the interdependencies of adjacent damping coefficients. The proposed methodology implicitly considers the nonlinearity of modeling of VDMF subjected to earthquakes. The MOPSO can be then performed for new ranges of damping coefficients and the interdependencies of dampers' coefficients in adjacent floors.

Starting with narrow ranges of damping coefficients as well as consideration of interdependencies of damping coefficients of adjacent floors drastically can decrease the runtime of MOPSO in absence of engineering knowledge on the range of damping coefficients.


Start
Set the large range for the damping coefficient in each floor
Equally divide the large range of damping coefficient in each floor
Run 1
Run 2
Set velocity exponent to the design level alpha (α=0.5)
Set velocity exponent to linear viscous damper (α=0.8)
Find the dampers' coefficients which satisfy the EDP's constraints in each floor [min (α=0.5) max (α=0.5]
Find the dampers' coefficients which satisfy the EDP's constraints in each floor [min (α=0.8) max (α=0.8]
Find maximum boundary for dampers' coefficients for each floor [max (α=0.5)]
Find minimum boundary for dampers' coefficients for each floor [min (α=0.8)]
End

DC: damping coefficient
Range DCi = [min (α=0.8) max (α=0.5)] (for each floor)
Spacing = length of large bracket/(maximum cores-1)
Ranges of damping coefficients for each floor (apply to MOPSO' input):
New Range DCi = [Range DCi (min) – Spacing Range DCi (max) + Spacing]

Note: no need of dampers for floor i if min (α=0.5) = 0

The interdependencies of damping coefficients between adjacent floors (apply to MOPSO' loop):

Interdependency is defined a range in which two adjacent dampers can work optimally together.

Interdependencies' relationships:

1) if (New Range DCi (min) = New Range DCj (min))
then DCi = DCj

if (New Range DCi (min) > New Range DCj (min))
then DCi > DCj + Min Diff

if (New Range DCi (min) < New Range DCj (min))
then DCj > DCi + Min Diff

Min Diff = the minimum differences of damping coefficients in adjacent floors
Min Diff = abs (New Range DCj (min) –New Range DCi (min))

i and j are adjacent floors.

FIGURE 6. Determining the range of damping coefficients and interdependencies of adjacent dampers scheme.

To summarize Section 2.7, the following steps are proposed to manage the broad range of damping coefficients to new ranges of damping coefficients as well as to consider interdependencies between the adjacent floors' dampers.

1. Choosing a large bracket of damping coefficients for all stories.
2. Dividing the large bracket of damping coefficients over the CPUs and assigning CPUs to each of the equidistant damping coefficients (see Figure 5).

3. Analyze VDMF subjected to earthquakes for the set of selected damping coefficients and velocity exponents (see Appendix B). The scheme is illustrated in Figure 6 for nonlinear and linear dampers.
4. At each story based on the constraints of EDPs determine the acceptable range of damping coefficients which satisfy the constraints (see Figure 6).
5. The maximum bound for the new narrow range is then represented by the highest damping coefficient that meets imposed constraints when nonlinear damper was used (e.g., $\alpha = 0.5$), while the minimum bound can be obtained form the when linear dampers (e.g., $\alpha = 0.8 - 1.0$) are used. In this step the ranges of damping coefficients are established.
6. Determining the interdependencies of dampers in adjacent floors (see Figure 6' instruction).
7. Running the MOPSO framework considering the appropriate range and interdependencies of adjacent dampers (see Appendix A).

CHAPTER 3

IMPLEMENTATION OF OPTIMIZATION FRAMEWORK

This chapter explores the implementation of MOPSO on a 3-story VDMF. For different combinations of constraints, the optimization of the VDMF is evaluated in terms of the quality of solution outcomes and the runtimes. Different cost functions are evaluated following the implementation of constraints. The most effective cost function is selected and sensitivity analysis is performed to improve the quality of solutions of MOPSO. Finally, the optimization strategy for optimal design of dampers in the absence of engineering knowledge on primary decision variables is implemented and its efficiency is demonstrated on the case study.

For implementation of the optimization methodology for the VDMF, the initial population size is assumed to be thirty and number of MOPSO iterations is set to five. Five iterations are utilized to explore suitable parameters of MOPSO as well as to compare the runtimes of different models. Optimization platform II (per Table 4) is utilized as a computation platform. The velocity exponents of dampers are set to 0.5 for all models presented in this chapter.

3.1 Description of VDMF, Site Characteristics, and Selected Ground Motion

A two-dimensional, three-story steel office building located in a high seismic hazard zone characteristic of Western North America was designed by Miyamoto International. The building had six bays with spacing of 30ft, typical story height of 15ft, and a first story height of 17ft. The site soil class was D, with shear wave velocity of 180 to 360 m/s. The building was located in Los Angeles at a site characterized with spectral accelerations of Ss=2.2g (for short period) and S1=0.74g (for period of 1s). The building was modeled in OpenSees (McKenna and Fenves (2004)) by Terzic and Mahin (2017). The configuration of the building is shown in Figure 7.

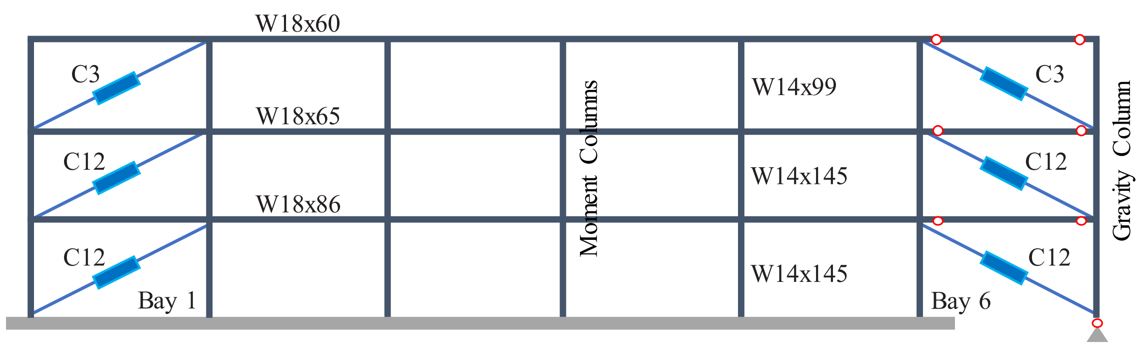


FIGURE 7. Configurations of VDMF (Terzic and Mahin 2017).

The viscous dampers' parameters are presented in Table 6. For the first and second stories, the dampers were designed to be identical. However, for the third story, the dampers were non-identical to the first and second stories.

TABLE 6. Characteristic of Dampers' Parameters in VDMF

Story	Stiffness (kip/in)	Damping Coefficient (kip/(in/s)$^{\alpha}$)	Velocity Exponent
1st and 2nd	2000	C12 = 135	0.5
3rd	2000	C3 = 35	0.5

Table 7 provides basic information for the ground motion utilized in this study for demonstration of the optimization framework. The selected ground motion is scaled to represent high seismic hazard for the considered site; it is representative of an earthquake with 2% probability of exceedance in 50 years. The time history of the scaled ground motion's acceleration is displayed in Figure 8.

TABLE 7. Ground Motion Information

NGA Record Sequence Number	Earthquake Name	Year	Station	Magnitude	Vs30 (m/s)	Direction	Scale Factor
6	Imperial Valley-02	1940	El Centro Array #9	7.0	213	Fault Parallel	3.63

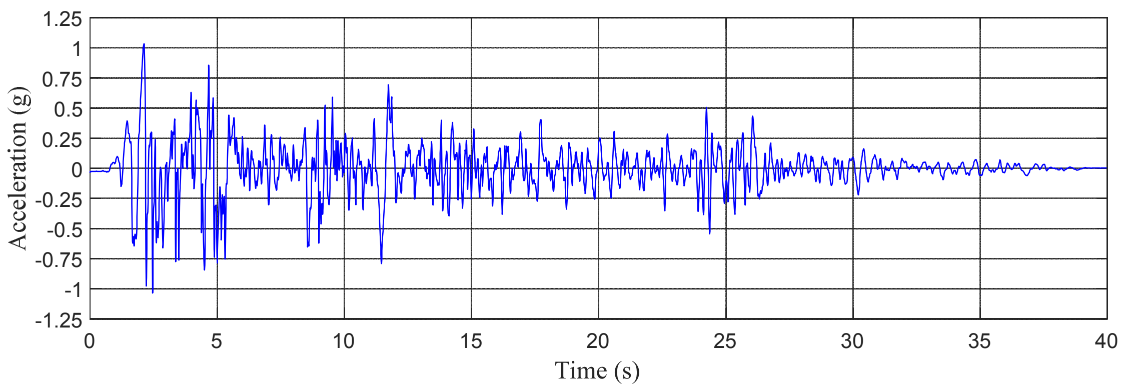


FIGURE 8. Time history of acceleration of selected ground motion (scaled).

3.2 Effect of EDPs' Constraints on Optimization Solutions

Table 8 shows different combination of constraints applied on some or all of the considered EDPs (including inter-story drifts, floor accelerations, and inter-story residual drifts), utilized within the optimization framework to show their effect on the solution outcomes. Total of eight different combinations of constraints were considered (designated with symbols from C1 through C8), ranging from "no constraints" (C1) to constraints applied on all three considered EDPs (C8).

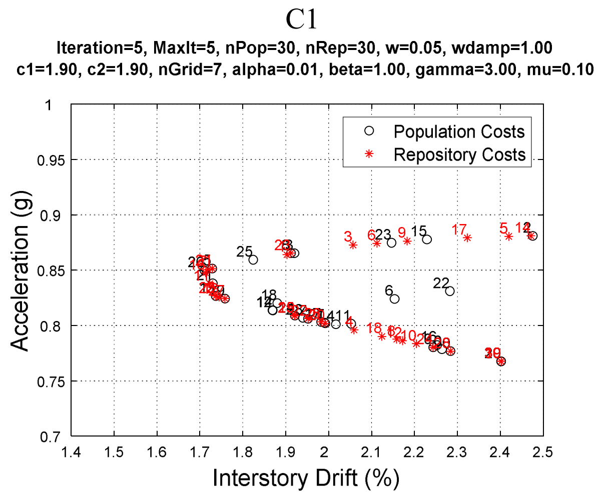
The same cost function was utilized for all models, represented by combination of maximum EDPs over the building height attained in the earthquake. The same range of damping coefficient is used for all models. It is selected to encompass damper properties of the VDMF design presented in Section 3.1. For the first and the second story the damping coefficient (C12) is set to 120 – 160 (kip/(in/s)$^{\alpha}$) and for the third story (C3) it is set to 20 – 60 (kip/(in/s)$^{\alpha}$).

The results of eight constraint models are compared in terms of solution outcomes and their runtimes. The Pareto fronts for the fifth iteration are presented for inter-story drift versus floor acceleration in Figures 9-12, for inter-story drift versus inter-story residual drift in Figures 13-16, and for inter-story residual drift versus floor acceleration in Figures 17-20.

TABLE 8. EDPs' Constraints

Model	Constraints	Cost Function	Damping Coefficient (kip/(in/s)$^{\alpha}$)
C1	No Constraints	Max Drift – Max Acc – Max Res Drift	C12 [120 160] C3 [20 60]
C2	Drift 2%	Max Drift – Max Acc – Max Res Drift	C12 [120 160] C3 [20 60]
C3	Acc 0.85g	Max Drift – Max Acc – Max Res Drift	C12 [120 160] C3 [20 60]
C4	Res Drift 0.5%	Max Drift – Max Acc – Max Res Drift	C12 [120 160] C3 [20 60]
C5	Drift 2% Acc 0.85g	Max Drift – Max Acc – Max Res Drift	C12 [120 160] C3 [20 60]
C6	Drift 2% Res Drift 0.5%	Max Drift – Max Acc – Max Res Drift	C12 [120 160] C3 [20 60]
C7	Acc 0.85g Res Drift 0.5%	Max Drift – Max Acc – Max Res Drift	C12 [120 160] C3 [20 60]
C8	Drift 2% Acc 0.85g Res Drift 0.5%	Max Drift – Max Acc – Max Res Drift	C12 [120 160] C3 [20 60]
Drift: Inter-Story Drift; Acc: Floor Acceleration; Res Drift: Residual Inter-Story Drift; g: Gravitational Acceleration			

Furthermore, the values of damping coefficients C12 (at the first and second story) versus C3 (at the third story) are graphed in Figures 21-24.



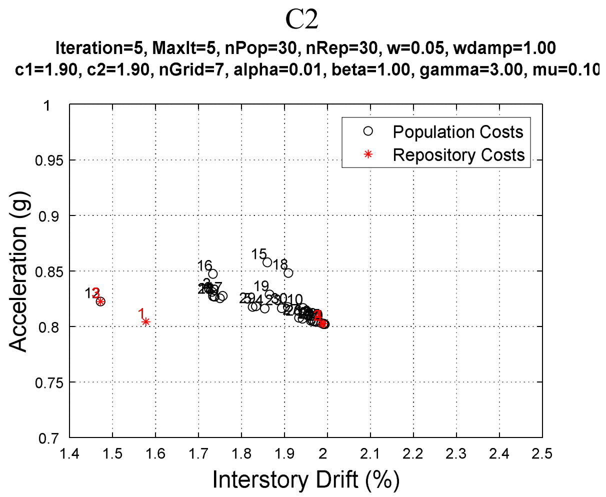


FIGURE 9. Effect of EDPs' constraints; inter-story drift versus floor acceleration (C1 and C2).

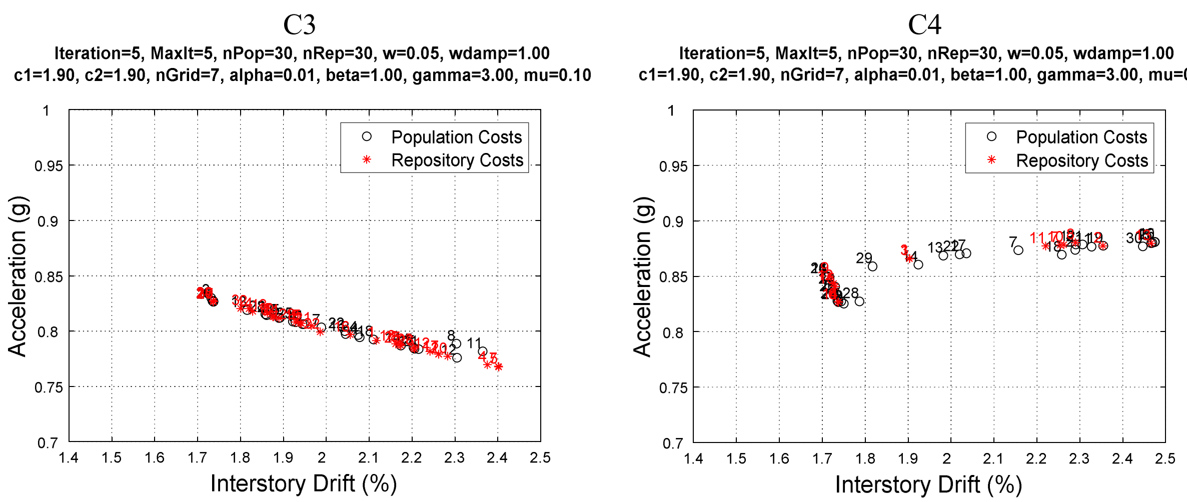


FIGURE 10. Effect of EDPs' constraints; inter-story drift versus floor acceleration (C3 and C4).

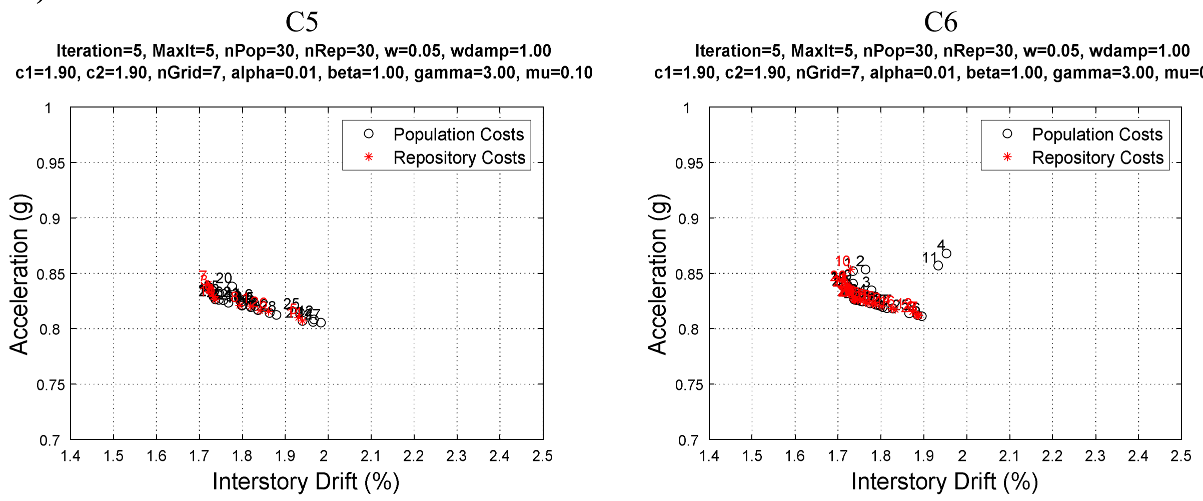


FIGURE 11. Effect of EDPs' constraints; inter-story drift versus floor acceleration (C5 and C6).

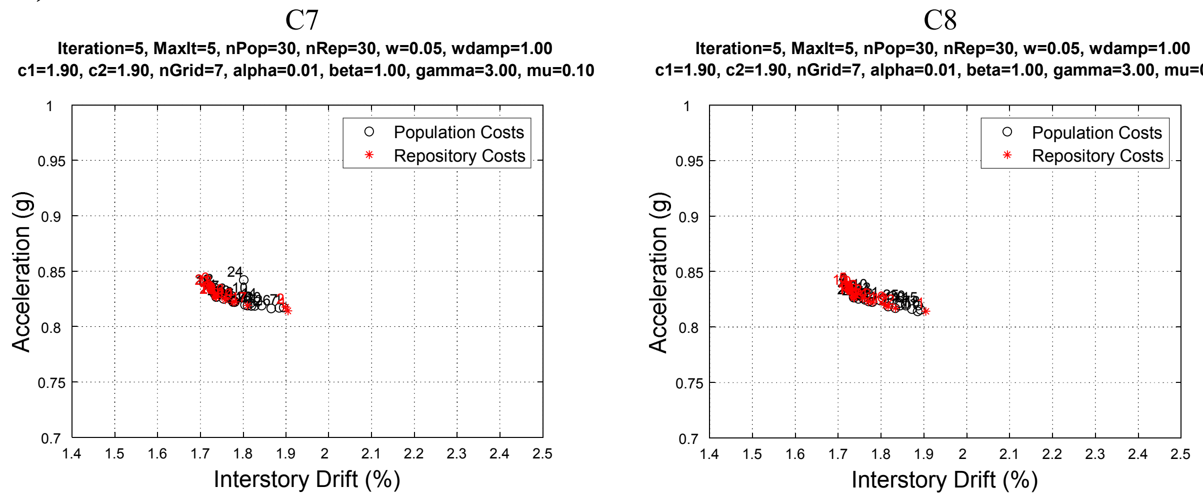
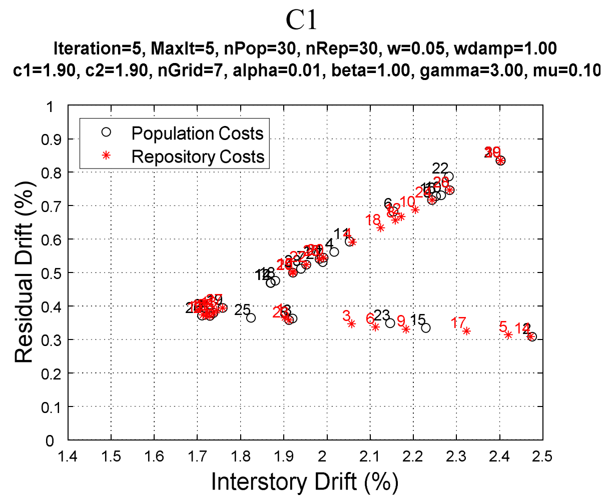


FIGURE 12. Effect of EDPs' constraints; inter-story drift versus floor acceleration (C7 and C8).



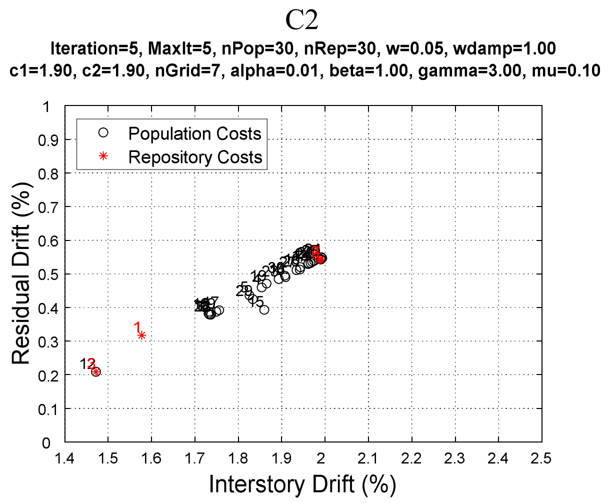
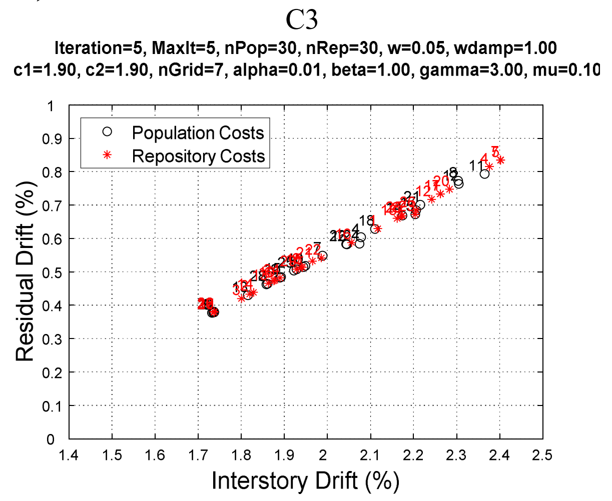


FIGURE 13. Effect of EDPs' constraints; inter-story drift versus residual drift (C1 and C2).



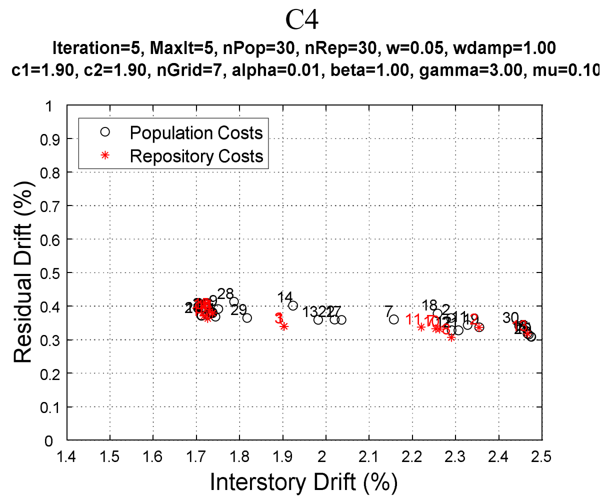
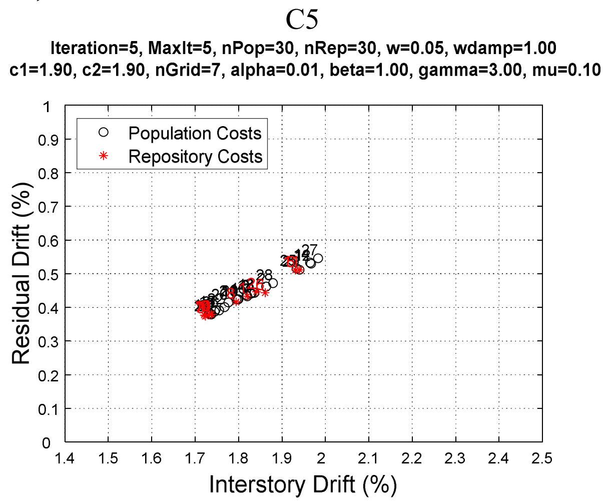


FIGURE 14. Effect of EDPs' constraints; inter-story drift versus residual drift (C3 and C4).



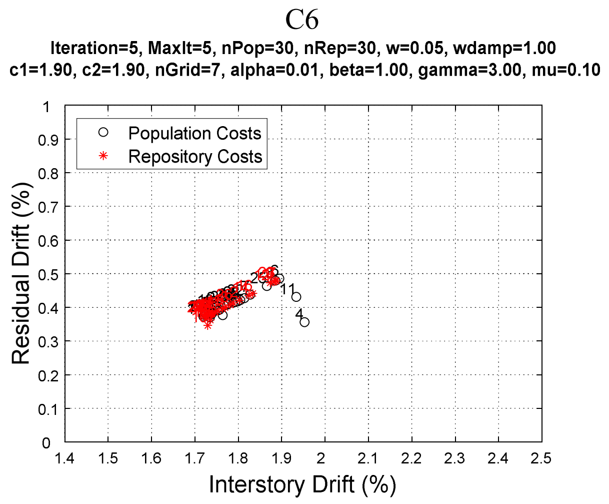
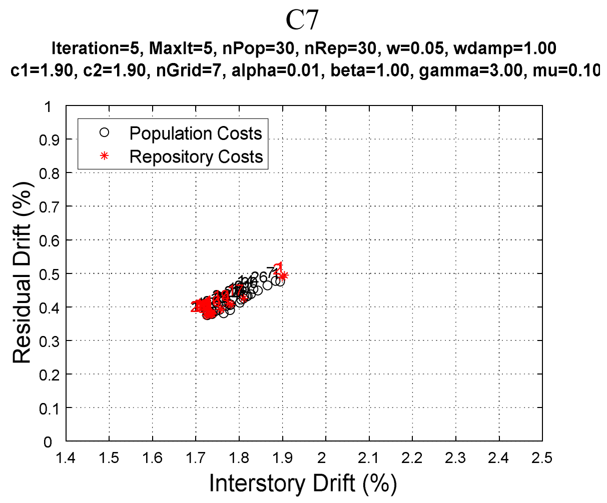


FIGURE 15. Effect of EDPs' constraints; inter-story drift versus residual drift (C5 and C6).



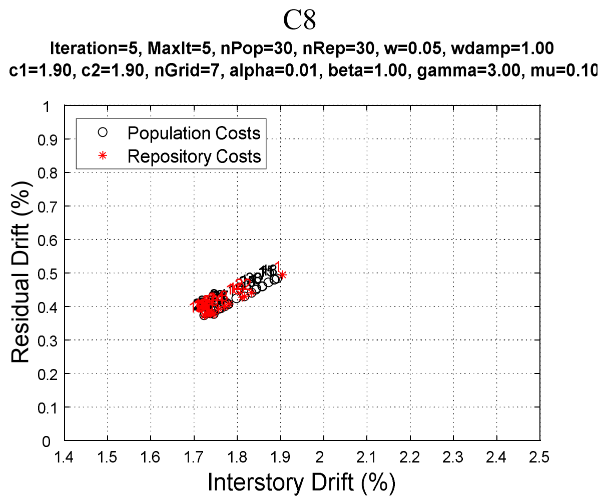
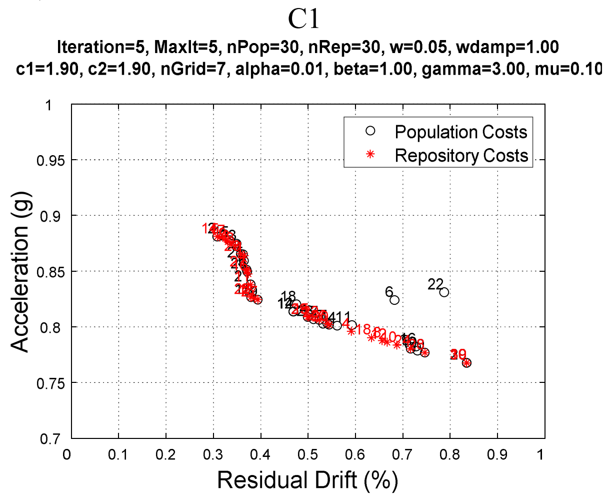


FIGURE 16. Effect of EDPs' constraints; inter-story drift versus residual drift (C7 and C8).



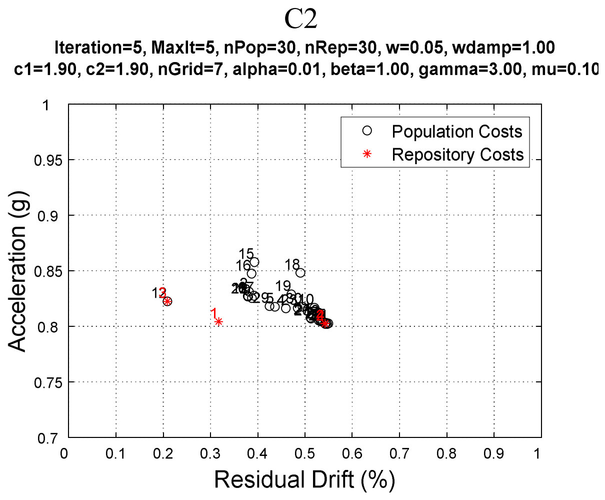
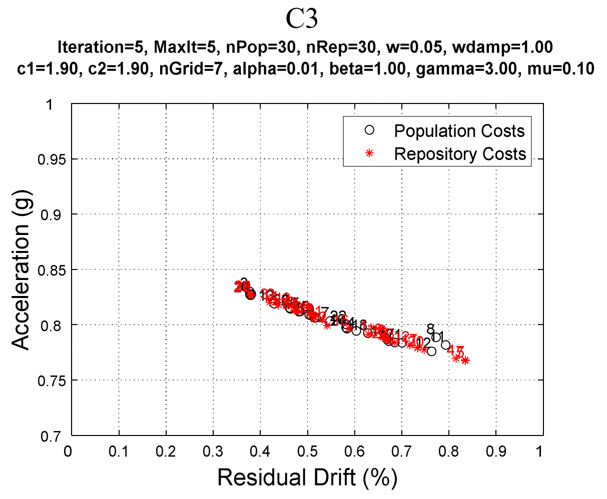


FIGURE 17. Effect of EDPs' constraints; residual drift versus floor acceleration (C1 and C2).



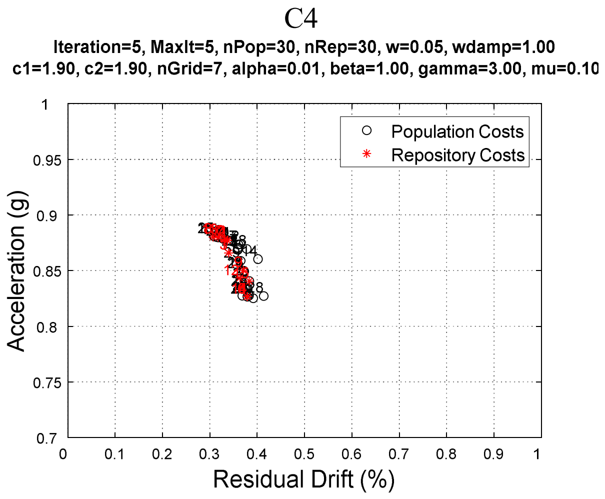


FIGURE 18. Effect of EDPs' constraints; residual drift versus floor acceleration (C3 and C4).

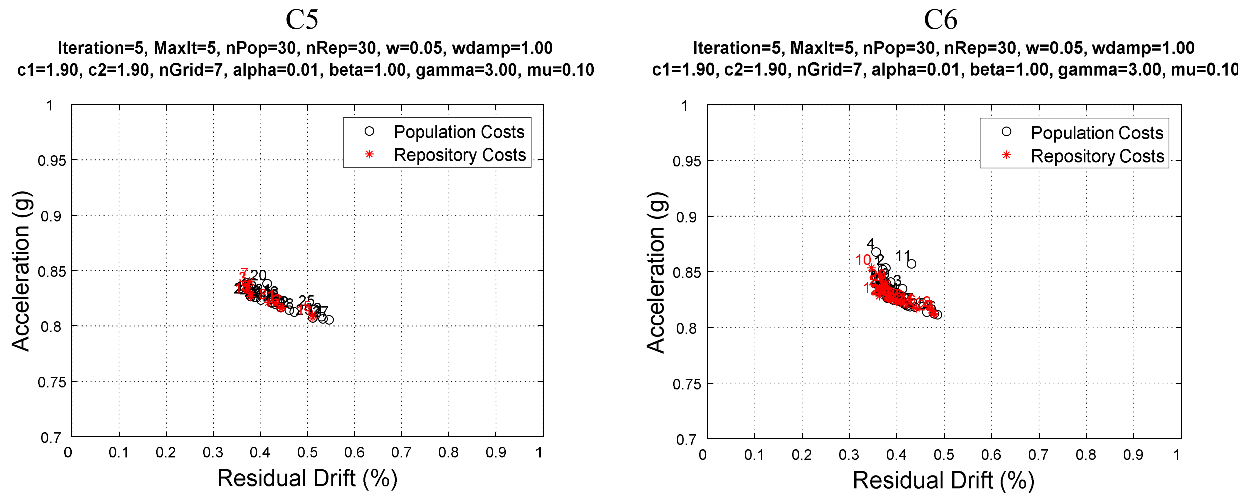


FIGURE 19. Effect of EDPs' constraints; residual drift versus floor acceleration (C5 and C6).

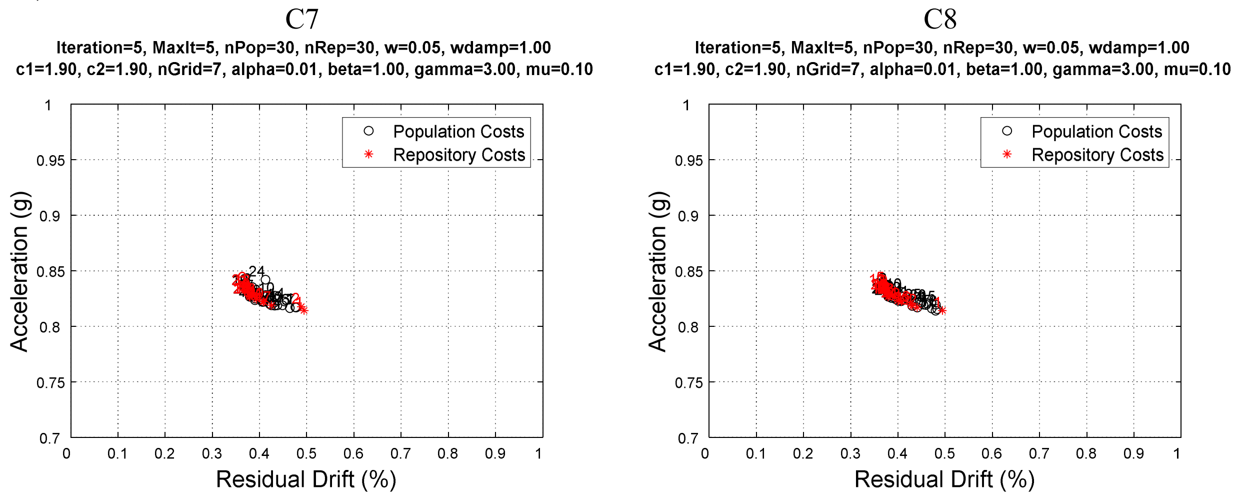
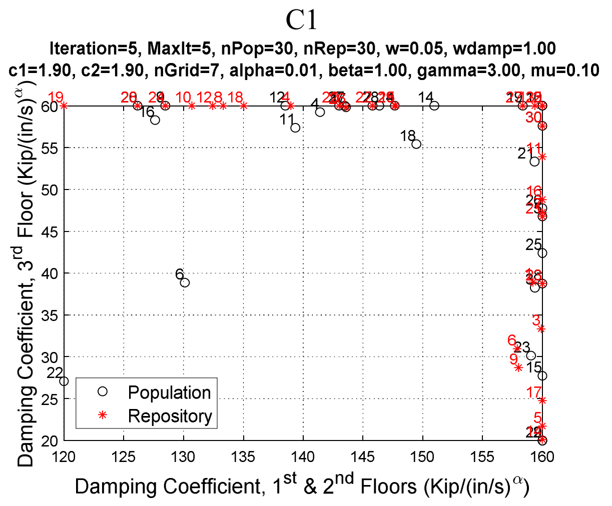


FIGURE 20. Effect of EDPs' constraints; residual drift versus floor acceleration (C7 and C8).

The optimal values of the damping coefficients are crucial for design of dampers. The results presented in Figures 21-24 show that adding constraints on inter-story drift, acceleration, and residual drift (C8) results in concentration of solutions for decision variables toward the top-right corner of the graph. Attained solutions may yield non-optimal design of dampers because there are possible deviant values with relation to other cost functions not considered at this time. Section 3.3 thoroughly investigates the effect of cost function on the quality of solutions (repositories of MOPSO).



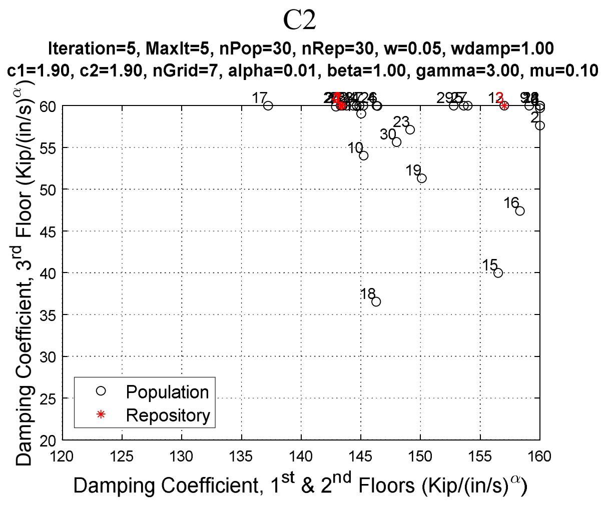
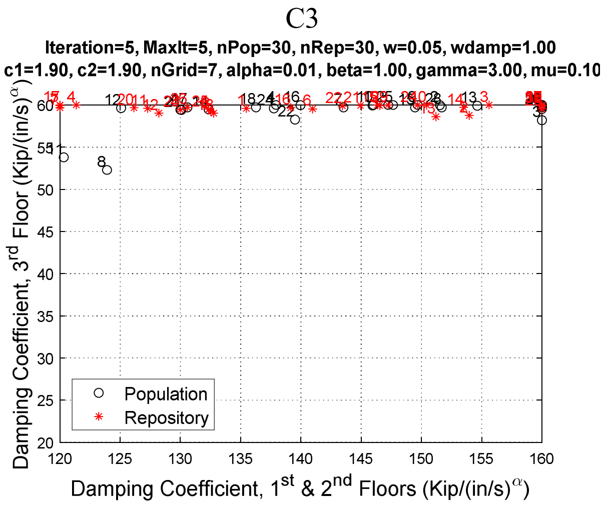


FIGURE 21. Effect of EDPs' constraints on the solution outcomes (C1 and C2).



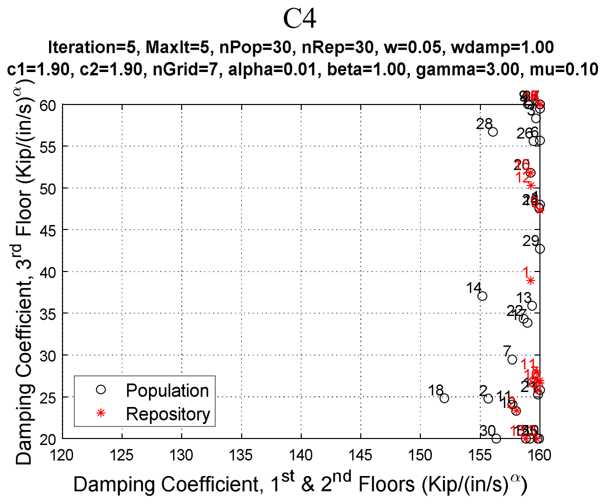
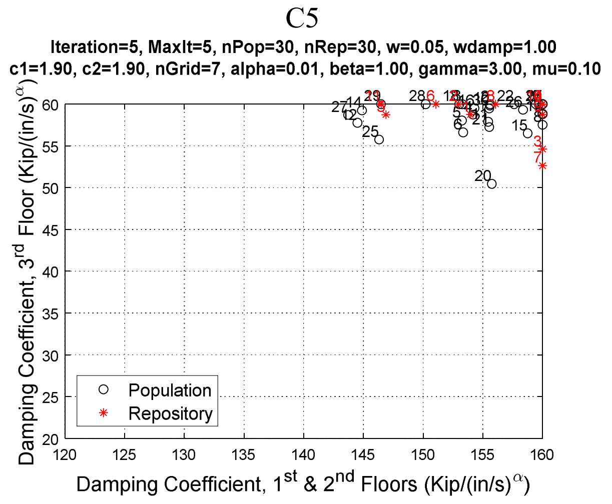


FIGURE 22. Effect of EDPs' constraints on the solution outcomes (C3 and C4).



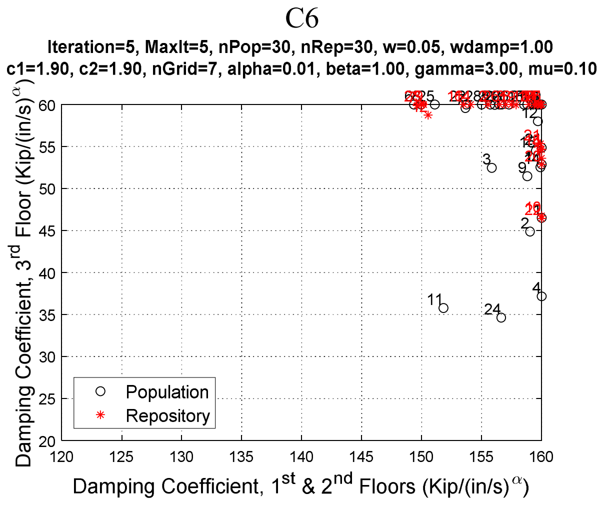
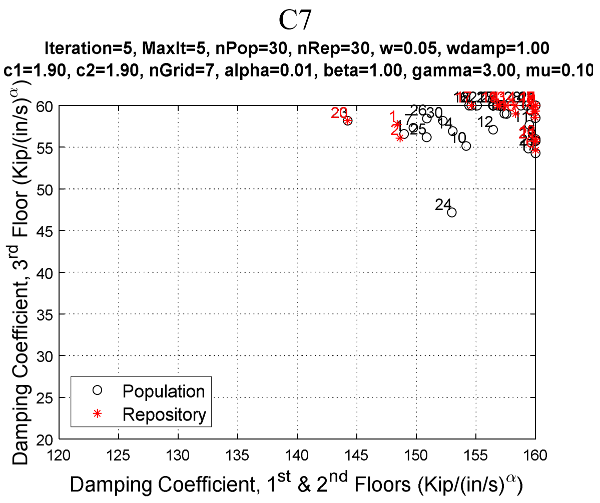


FIGURE 23. Effect of EDPs' constraints on the solution outcomes (C5 and C6).



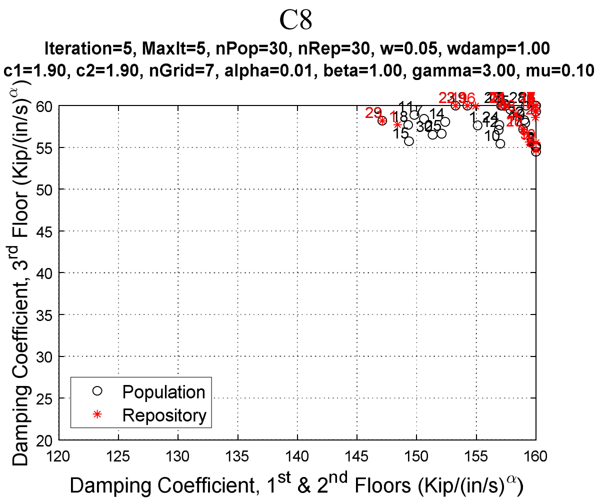


FIGURE 24. Effect of EDPs' constraints on the solution outcomes (C7 and C8).

The results of models with constraints (C1 through C8) show strong influence of constraints on the optimization outcomes depicted with Pareto fronts; larger set of constraints results in larger reduction of Pareto front. Similarly, the range of decision variables (i.e., damping coefficients) reduces significantly under applied constraints.

Table 9 provides the runtimes of optimization models calculated in five iterations utilizing optimization platform II (see Table 4). The calculation times indicate that adding constraints on EDPs increases runtimes. This is expected as some of the initial randomly selected populations will not meet constraint and therefore will be replaced and simulation repeated. Further, when constraints are applied to a single EDP the runtimes are less than when adding multiple constraints.

TABLE 9. Effect of EDP's Constraints on Runtime

Model	Runtime (s)	Runtime (hour)
C1	5.744903018375642e+03	1.60
C2	1.130974939764995e+04	3.14
C3	6.750744037154407e+03	1.88
C4	8.534024762605959e+03	2.37
C5	1.146833277157542e+04	3.19
C6	1.159019403930066e+04	3.22
C7	1.475793564699208e+04	4.10
C8	1.461195876678275e+04	4.06

3.3 Effect of Cost Function

In Section 3.2, it is shown that the position of dampers may not be optimized under the set of constraints on EDPs when cost function is the combination of inter-story drifts, floor accelerations, and inter-story residual drifts. Table 10 shows different cost functions explored in this section. Total of nine different combinations of cost functions were considered (designated with symbols from CF1 through CF9), ranging from consideration of only maximum inter-story drifts (CF1) to inclusion of all three considered EDPs along with the damping coefficients and their linear combination (CF7).

TABLE 10. Cost Functions

Model	Constraints	Cost Function	Damping Coefficient (kip/(in/s)$^{\alpha}$)
CF1	Drift 2% Acc 0.85g Res Drift 0.5%	Max Drift	C12 [120 160] C3 [20 60]
CF2	Drift 2% Acc 0.85g Res Drift 0.5%	Max Drift – Max Acc	C12 [120 160] C3 [20 60]
CF3	Drift 2% Acc 0.85g Res Drift 0.5%	Max Drift – Max Acc – Max Res Drift	C12 [120 160] C3 [20 60]
CF4	Drift 2% Acc 0.85g Res Drift 0.5%	Max Drift – C12 – C3	C12 [120 160] C3 [20 60]
CF5	Drift 2% Acc 0.85g Res Drift 0.5%	Max Drift – Max Acc – C12 – C3	C12 [120 160] C3 [20 60]
CF6	Drift 2% Acc 0.85g Res Drift 0.5%	Max Drift – Max Acc – Max Res Drift – C12 – C3	C12 [120 160] C3 [20 60]
CF7	Drift 2% Acc 0.85g Res Drift 0.5%	Max Drift – Max Acc – Max Res Drift – C12 – C3 – C12+C3	C12 [120 160] C3 [20 60]
CF8	Drift 2% Acc 0.85g Res Drift 0.5%	Max Drift – C12 – C3 – C12+C3	C12 [120 160] C3 [20 60]
CF9	Drift 2% Acc 0.85g Res Drift 0.5%	C12 – C3 – C12+C3	C12 [120 160] C3 [20 60]
Drift: Inter-Story Drift; Acc: Floor Acceleration; Res Drift: Residual Inter-Story Drift; g: Gravitational Acceleration			

The same set of constraints was utilized for all models, including the constraints applied to inter-story drifts, floor accelerations, and inter-story residual drifts. The same range of damping coefficient is used for all models. Finally, different considered cost functions are evaluated and the results of the cost functions are compared in terms of the quality the solution outcomes. The runtimes of the models are then compared.

The results of MOPSO, shown in Figures 25-29, indicate significant effect of the cost function on the quality of the solutions. It is apparent that the model CF7 which includes of all three considered EDPs along with the damping coefficients and their linear combination provides the most comprehensive set of solutions for the considered decision variables and is therefore selected for further use.

Moreover, it is to be noted that cost functions typically used by research community which include drift, combination of drift and acceleration, or only damping coefficients, in case of MOPSO would not generate optimized solutions.

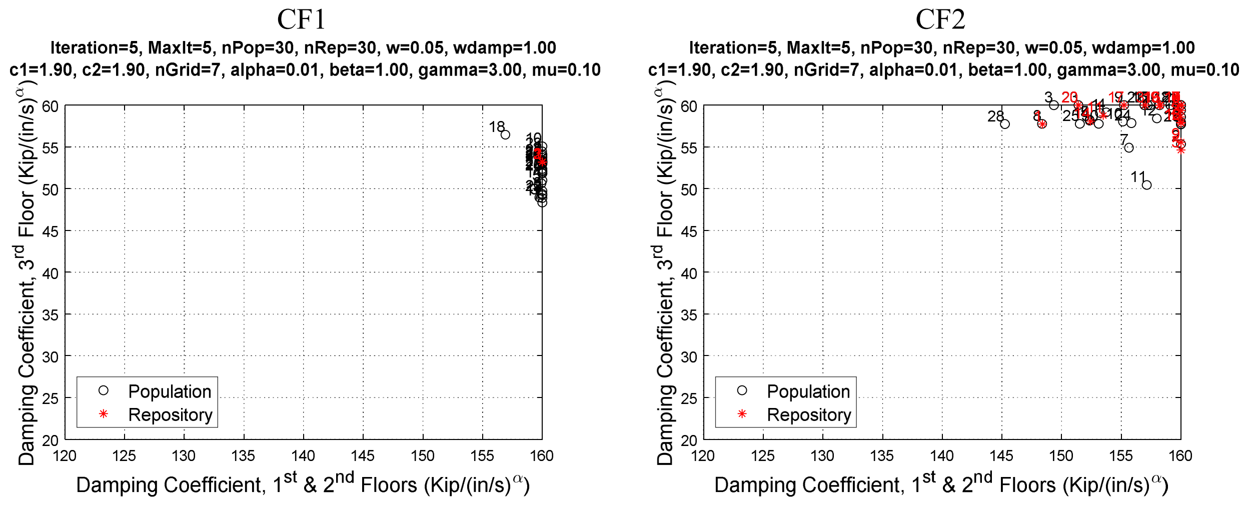


FIGURE 25. Effect of cost function on the solution outcomes (CF1 and CF2).

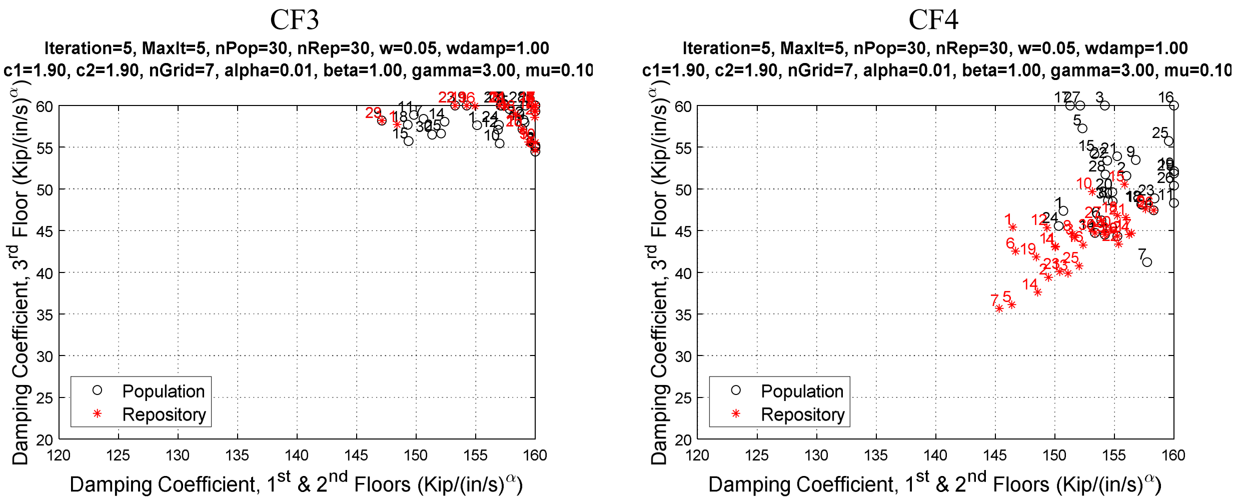


FIGURE 26. Effect of cost function on the solution outcomes (CF3 and CF4).

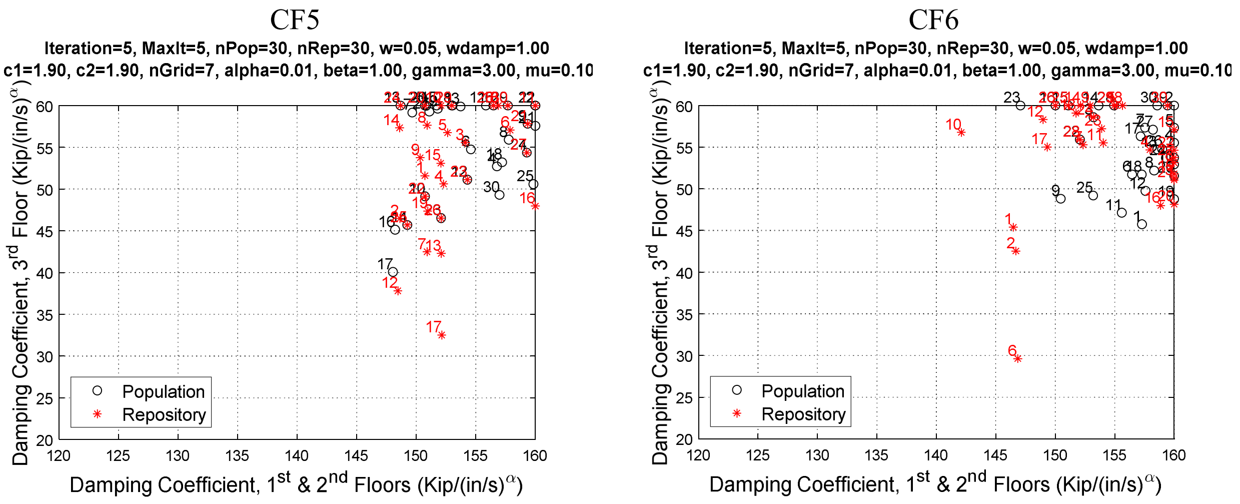


FIGURE 27. Effect of cost function on the solution outcomes (CF5 and CF6).

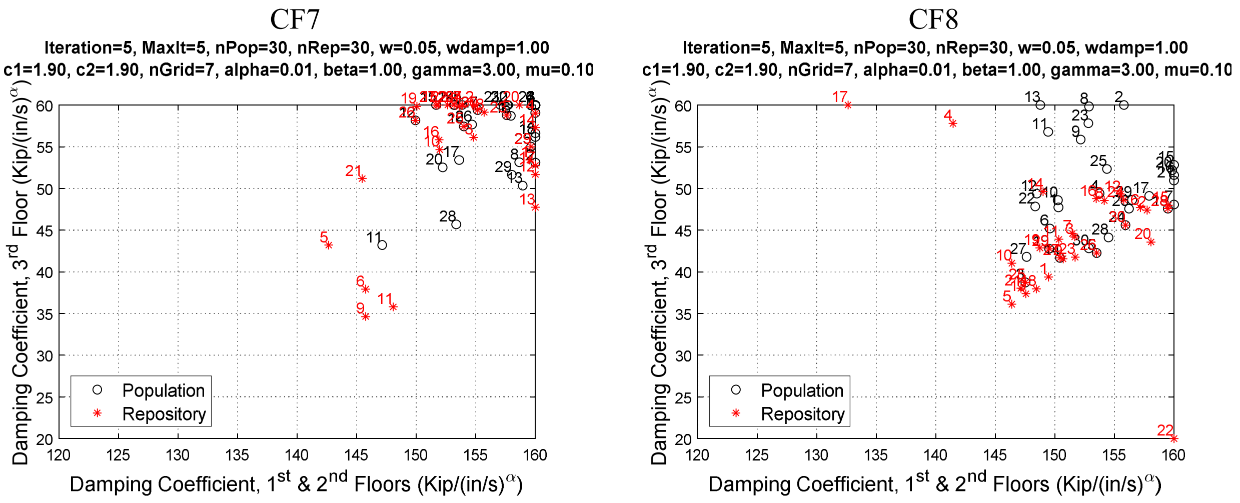


FIGURE 28. Effect of cost function on the solution outcomes (CF7 and CF8).

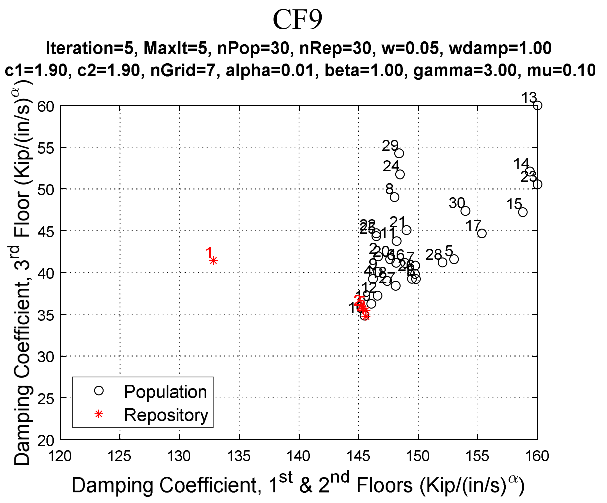


FIGURE 29. Effect of cost function on the solution outcomes (CF9).

Table 11 shows the runtimes of considered models. The results indicate that considering only damping coefficients in cost function would significantly increase the runtime (CF9) while not achieving the optimal solutions. However, the runtime of CF7 model is comparable to the runtimes of all the other models excluding CF9.

TABLE 11. Effect of Cost Functions on Runtime

Model	Runtime (s)	Runtime (hour)
CF1	1.421079683306645e+04	3.95
CF2	1.379509803702003e+04	3.83
CF3	1.461195876678275e+04	4.06
CF4	1.815766190805460e+04	5.04
CF5	1.603671973179326e+04	4.45
CF6	1.587033613605739e+04	4.41
CF7	1.626236772140914e+04	4.52
CF8	1.901903654179218e+04	5.28
CF9	6.882535902395962e+04	19.12

3.4 Effect of nGrid, Population Size, and Number of Iterations on MOPSO Solution Outcomes

Previously, Thomas (2018) investigated the effect of different MOPSO input parameters on the quality of solution outcomes of a SDOF system subjected to an earthquake. They have found out that the nGrid parameter had a particularly significant impact on solutions because of its influence in PESA optimization. Furthermore, starting population size and number of iterations have a great influence on the outcomes of optimization.

The Table 12 shows the three considered models used to explore sensitivity of MOPSO solution outcomes on the nGrid parameter. The models utilize CF7 as a base (defined in Table 10) and vary nGrid parameter considering the following values: 7 (CF7-S7), 10 (C7-S10) and 15 (C7-S15). From the three considered nGrid values, as can be seen from Figures 30-31, nGrid of 15 provides the best set of MOPSO solution outcomes. Therefore, all further studies will use nGrid of 15.

TABLE 12. Models Used to Explore Sensitivity of MOPSO Solution Outcomes on the nGrid Parameter

Model	Constraints	Cost Function	Damping Coefficient (kip/(in/s)$^{\alpha}$)
CF7-S7 [nGrid=7]	Drift 2% Acc 0.85g Res Drift 0.5%	Max Drift – Max Acc – Max Res Drift – C12 – C3 – C12+C3	C12 [120 160] C3 [20 60]
CF7-S10 [nGrid=10]	Drift 2% Acc 0.85g Res Drift 0.5%	Max Drift – Max Acc – Max Res Drift – C12 – C3 – C12+C3	C12 [120 160] C3 [20 60]
CF7-S15 [nGrid=15]	Drift 2% Acc 0.85g Res Drift 0.5%	Max Drift – Max Acc – Max Res Drift – C12 – C3 – C12+C3	C12 [120 160] C3 [20 60]
Drift: Inter-Story Drift; Acc: Floor Acceleration; Res Drift: Residual Inter-Story Drift; g: Gravitational Acceleration			

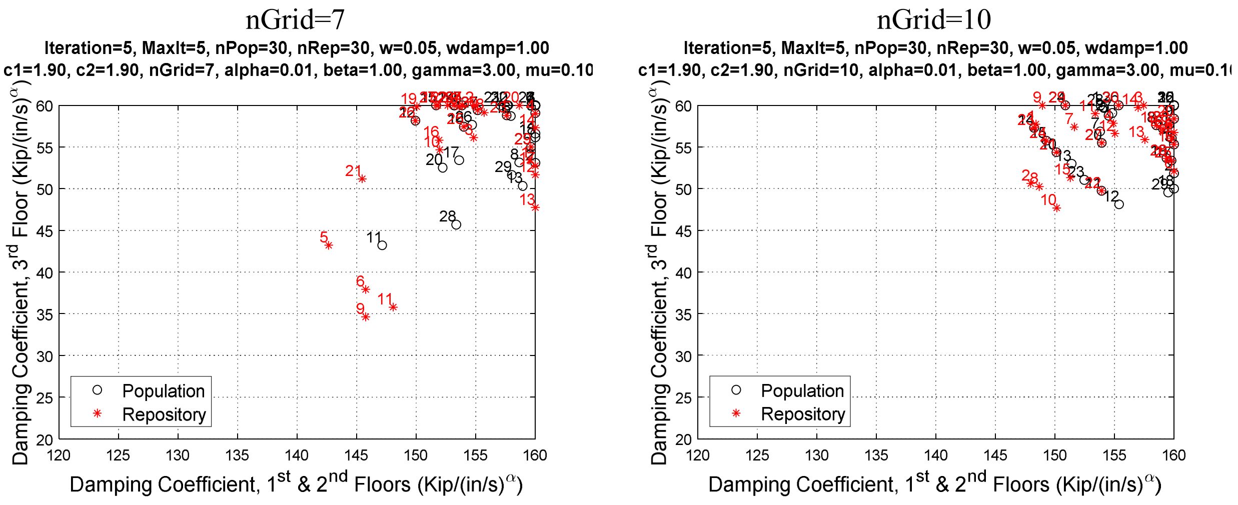


FIGURE 30. Effect of nGrid MOPSO parameter on the solution outcomes (nGrid=7 and nGrid=10).

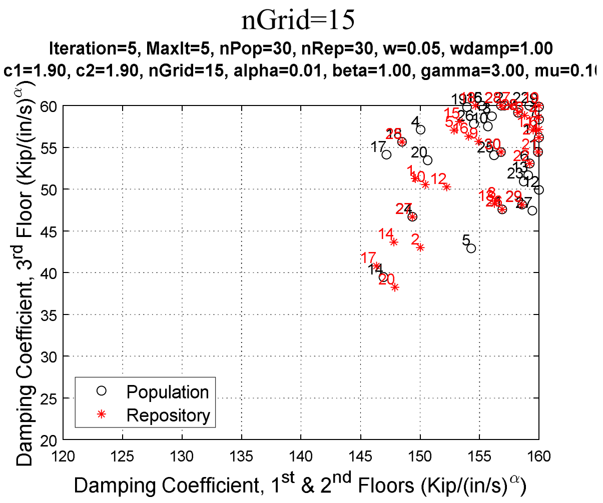


FIGURE 31. Effect of nGrid MOPSO parameter on the solution outcomes (nGrid=15).

To explore the effect of the population size (designated as nPop) on the quality of MOPSO solution outcomes, the following values of nPop are considered: 10, 15, 20, 25, and 30. The Models are introduced in Table 13. The MOPSO solution outcomes are compared in Figures 32-33, showing that nPop of 30 provides the best set of MOPSO solution outcomes. Therefore, all further studies will use nPop of 30.

TABLE 13. Models Used to Explore Sensitivity of MOPSO Solution Outcomes on the nPop Parameter

Model	Constraints	Cost Function	Damping Coefficient (kip/(in/s)$^\alpha$)
CF7-S15-P10 [nPop=10]	Drift 2% Acc 0.85g Res Drift 0.5%	Max Drift – Max Acc – Max Res Drift – C12 – C3 – C12+C3	C12 [120 160] C3 [20 60]
CF7-S15-P15 [nPop=15]	Drift 2% Acc 0.85g Res Drift 0.5%	Max Drift – Max Acc – Max Res Drift – C12 – C3 – C12+C3	C12 [120 160] C3 [20 60]
CF7-S15-P20 [nPop=15]	Drift 2% Acc 0.85g Res Drift 0.5%	Max Drift – Max Acc – Max Res Drift – C12 – C3 – C12+C3	C12 [120 160] C3 [20 60]
CF7-S15-P25 [nPop=15]	Drift 2% Acc 0.85g Res Drift 0.5%	Max Drift – Max Acc – Max Res Drift – C12 – C3 – C12+C3	C12 [120 160] C3 [20 60]
CF7-S15-P30 [nPop=30]	Drift 2% Acc 0.85g Res Drift 0.5%	Max Drift – Max Acc – Max Res Drift – C12 – C3 – C12+C3	C12 [120 160] C3 [20 60]
Drift: Inter-Story Drift; Acc: Floor Acceleration; Res Drift: Residual Inter-Story Drift; g: Gravitational Acceleration			

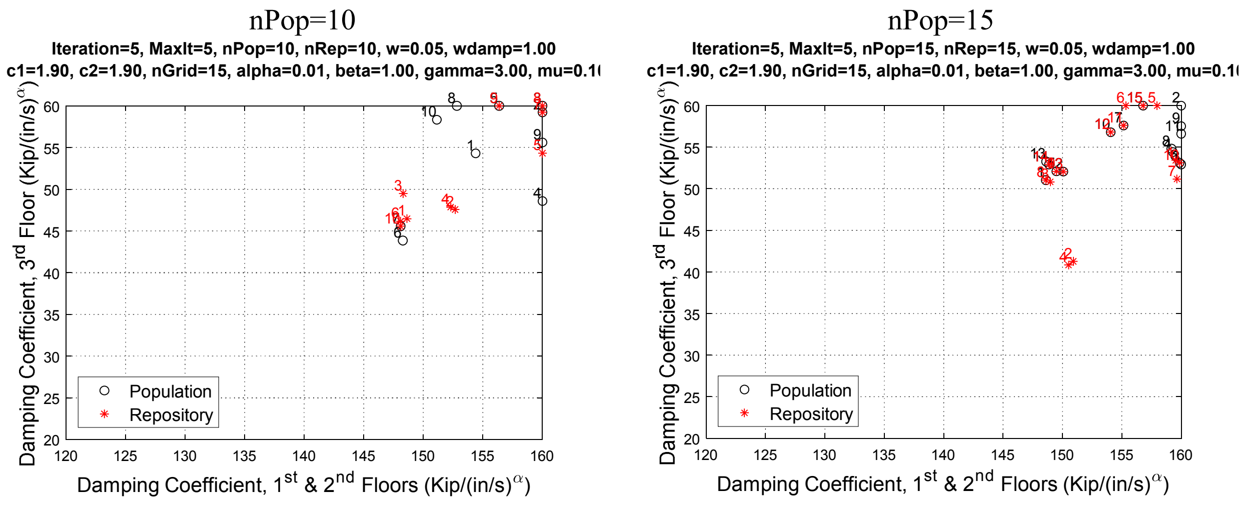


FIGURE 32. Effect of nPop MOPSO parameter on the solution outcomes (nPop=10 and nPop=15).

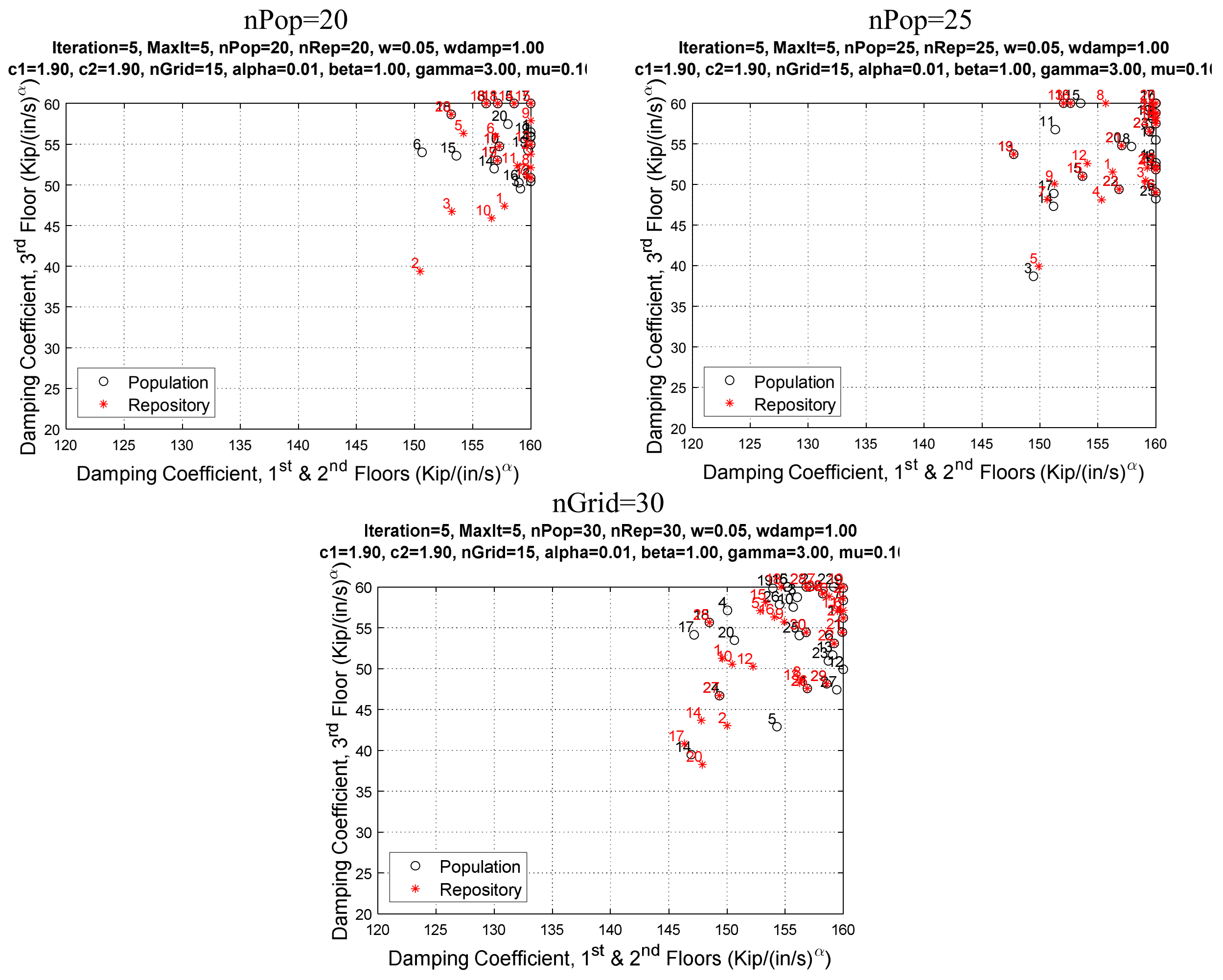


FIGURE 33. Effect of nPop MOPSO parameter on the solution outcomes (nPop=20to nPop=30).

Figure 34 shows the solution of CF7-S15-P30 for the first five iterations, which is equivalent to set the maximum number of iterations (designated as "MaxIt") to 1, 2, 3, 4, and 5.

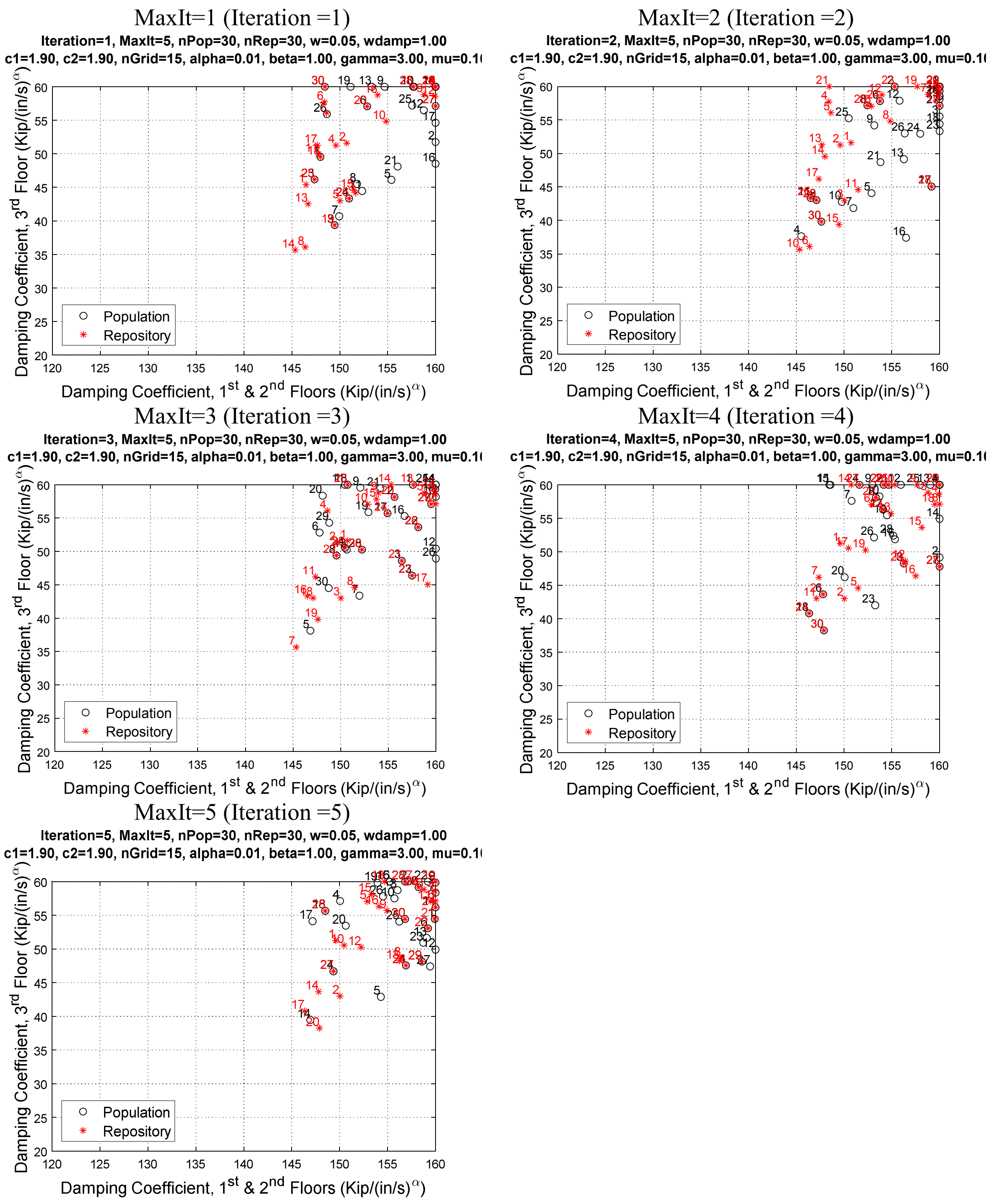


FIGURE 34. MOPSO solution outcomes for different number of considered iterations.

3.5 Maximums versus Averages of EDPs in Cost Function

After determining cost function and performing sensitivity analysis, the maximums and averages of EDPs in cost function are compered. Table 14 shows the models for comparing the maximums or averages of EDPs in Cost Function. Maxima of EDPs refer to the absolute maximum values along all stories/floors of inter-story drifts, floor acceleration, and inter-story residual drifts. Averages of EDPs refer to the averages of maximum values of inter-story drifts, floor accelerations, and inter-story residual drifts in stories/floors.

TABLE 14. Models for Comparing Maxima versus Averages of EDPs in Cost Function

Model	Constraints	Cost Function	Damping Coefficient (kip/(in/s)$^{\alpha}$)
CF7-S15-Max	Drift 2% Acc 0.85g Res Drift 0.5%	Max Drift – Max Acc – Max Res Drift – C12 – C3 – C12+C3	C12 [120 160] C3 [20 60]
CF7-S15-Avg	Drift 2% Acc 0.85g Res Drift 0.5%	Avg Drift – Avg Acc – Avg Res Drift – C12 – C3 – C12+C3	C12 [120 160] C3 [20 60]
Drift: Inter-Story Drift; Acc: Floor Acceleration; Res Drift: Residual Inter-Story Drift; g: Gravitational Acceleration			

MOPSO solution outcomes for maxima and averages of EDPs in cost function are shown in Figure 35. This figure depicts the solutions of two cost functions that are located in similar ranges and compose similar patterns; however, the solutions of maxima are less dense.

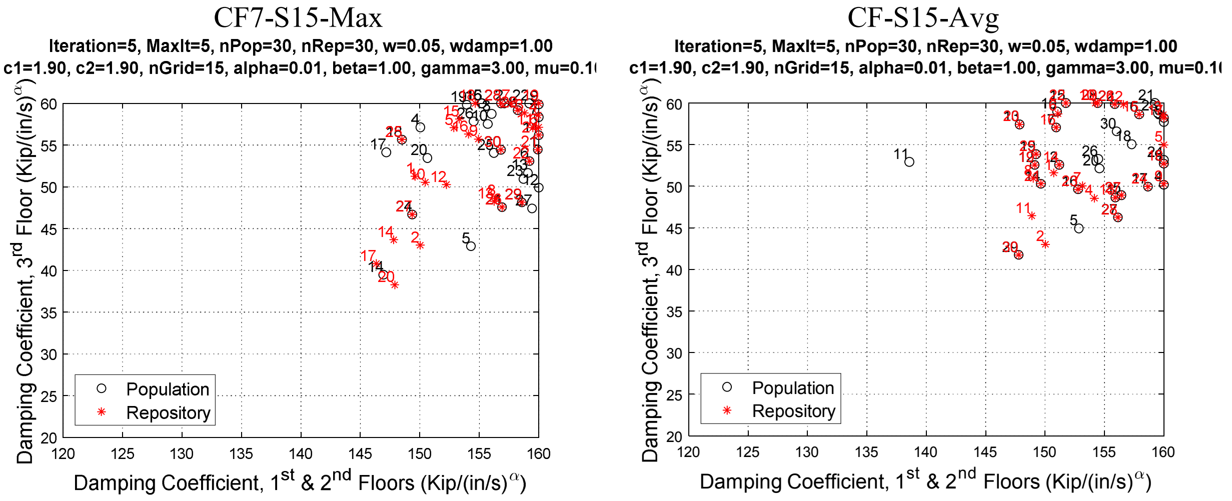


FIGURE 35. MOPSO solution outcomes for maxima and averages of EDPs in cost function.

EDP profiles associated with the solution outcomes for the two considered cost functions are illustrated in Figure 36; namely inter-story drifts, floor accelerations, and inter-story residual drifts. Figure 36 shows that the profiles of all EDPs comprise wider ranges of solutions when maxima of EDPs are considered for optimization.

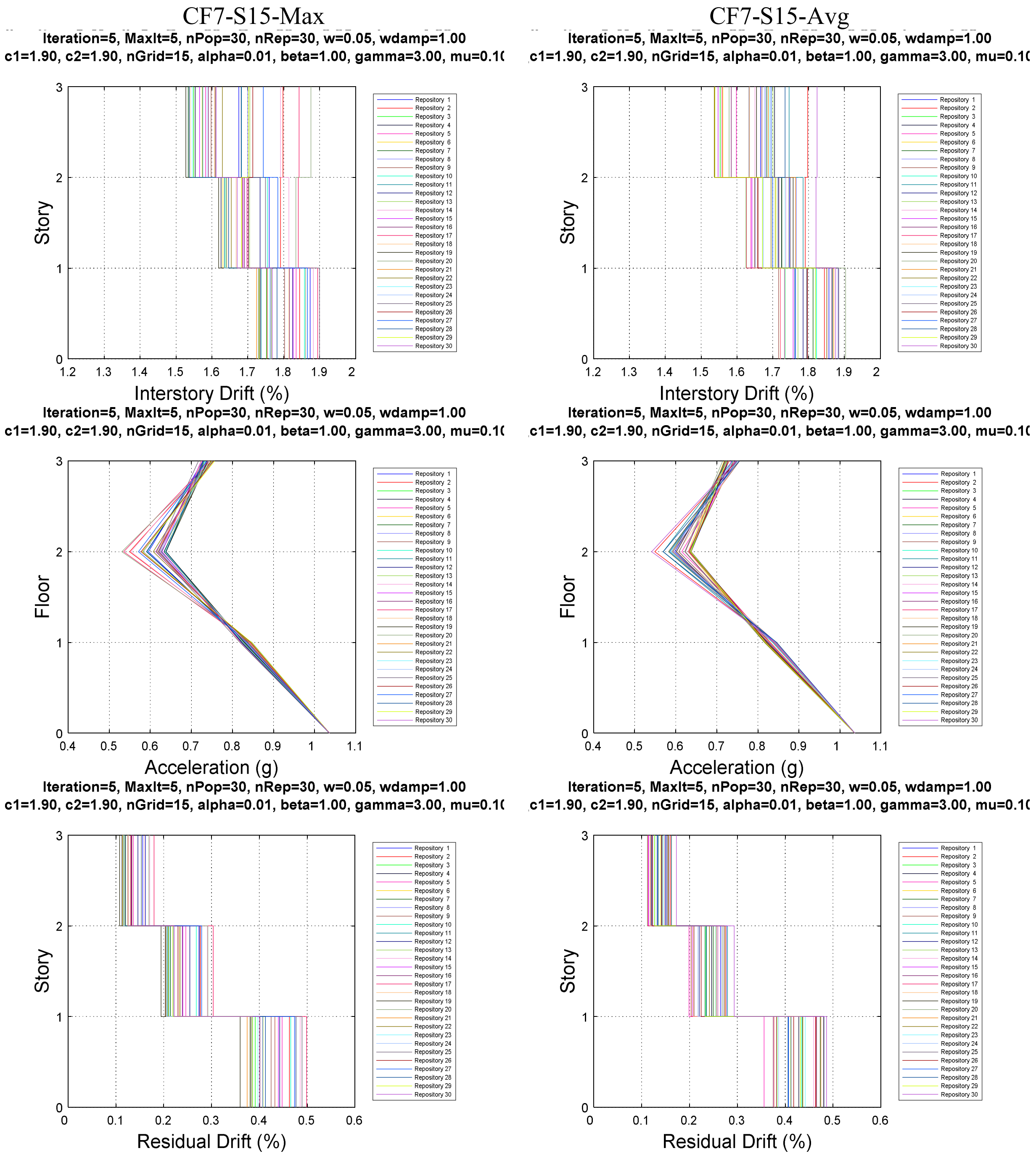


FIGURE 36. Profiles of EDPs for maxima and averages of EDPs in cost function.

Figure 37 shows the minima, averages and maxima of each pair of the three considered EDPs (inter-story drifts, floor accelerations, and inter-story residual drifts) for each repository.

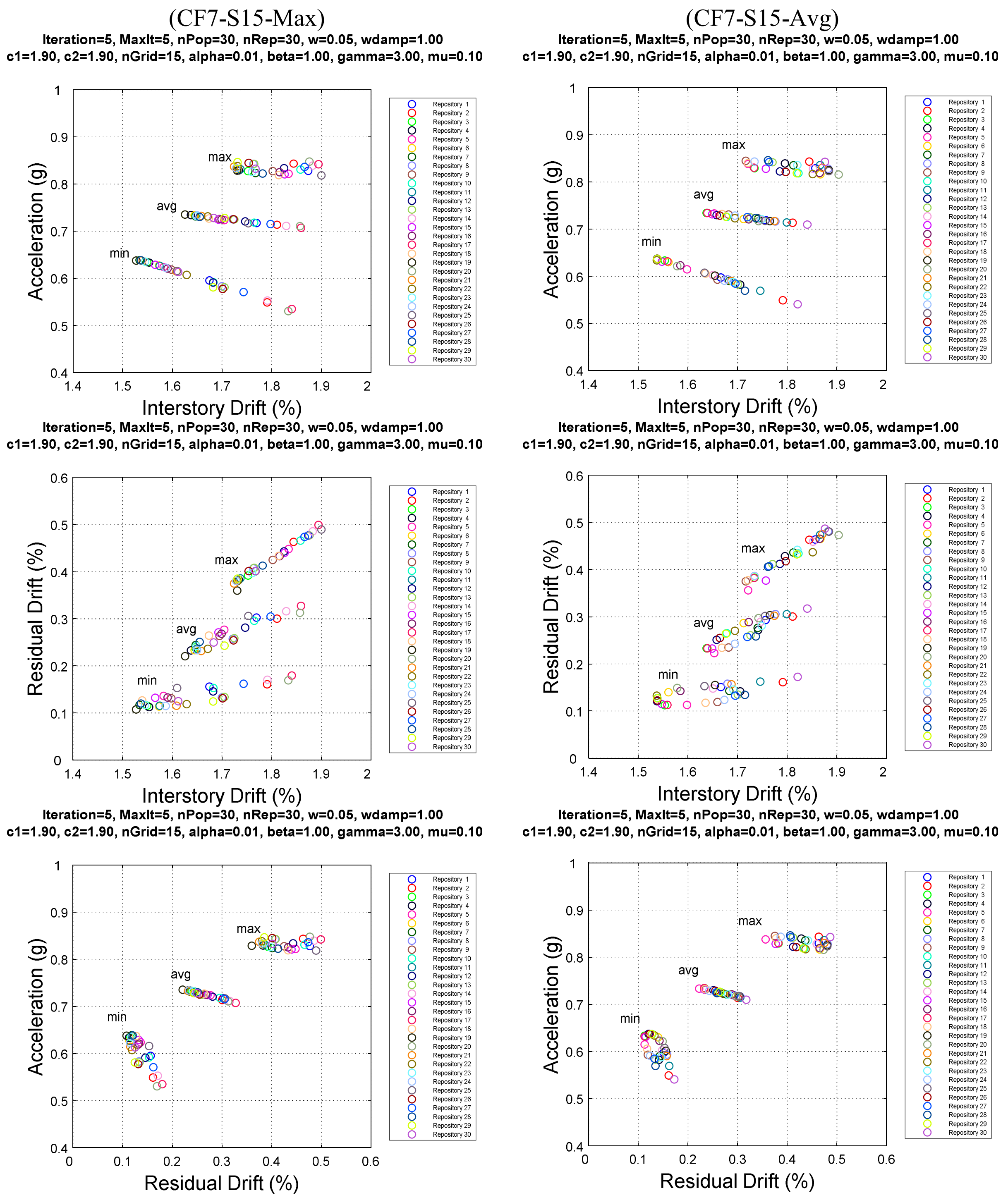


FIGURE 37. Minima, averages, and maxima of EDPs maxima versus averages of EDPs in cost function.

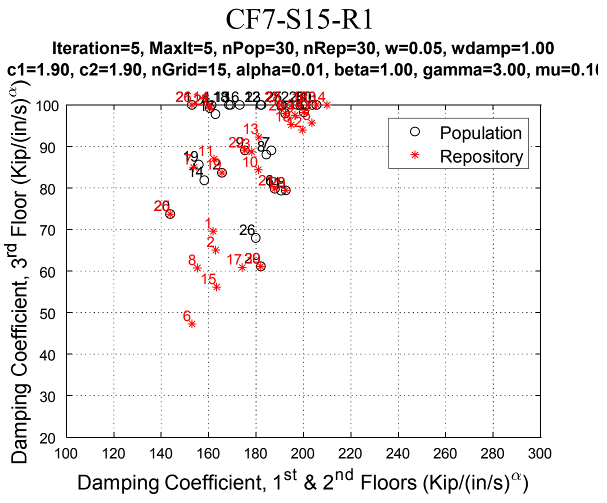
Figure 37 demonstrates that considering both maxima and averages of EDPs in cost function would result in similar patterns of solutions.

3.6 Effect of Selected Range of Damping Coefficients on MOPSO Solution Outcomes

For Sections 3.2 to 3.5 of this chapter, the ranges of damping coefficients for the first and second story were assumed to range from 120 – 160 (kip/(in/s)$^{\alpha}$), and 20 – 60 (kip/(in/s)$^{\alpha}$) for the third story. The question is: "can MOPSO find the non-convex solutions for a broad range of damping coefficients?" To answer this question, two models (CF7-S15-R1 and CF7-S15-R2) that consider wide ranges of damping coefficients as shown in Table 15. From the two models, CF7-S15-R1 considers smaller range for damping coefficient of the third story than CF7-S15-R2. MOPSO solution outcomes are shown in Figure 38.

TABLE 15. Models that Consider Broad Range of Damping Coefficients

Model	Constraints	Cost Function	Damping Coefficient (kip/(in/s)$^{\alpha}$)
CF7-S15-R1	Drift 2% Acc 0.85g Res Drift 0.5%	Max Drift – Max Acc – Max Res Drift – C12 – C3 – C12+C2	C12 [100 300] C3 [20 100]
CF7-S15- R2	Drift 2% Acc 0.85g Res Drift 0.5%	Max Drift – Max Acc – Max Res Drift – C12 – C3 – C12+C2	C12 [100 300] C3 [20 200]
Drift: Inter-Story Drift; Acc: Floor Acceleration; Res Drift: Residual Inter-Story Drift; g: Gravitational Acceleration			



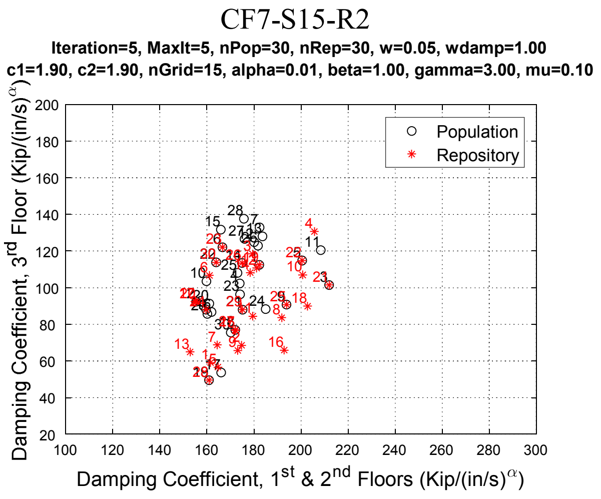


FIGURE 38. The effect of the selected ranges of damping coefficients on the MOPSO solution outcomes.

The results indicate that MOPSO can find similar solutions when the selected ranges for decision variables are broad. Moreover, the CF7-S15-R2 model, which considers wider range of damping coefficient compared to CF7-S15-R1 model, provides larger set of solution outcomes that meet imposed constraints.

The runtimes of models with the broad range of damping coefficients, CF7-S15-R1 and CF7-S15- R2, are provided in Table 16. The runtimes of these models are about 2 times longer when compared to narrow ranges considered in Section 3.2. To improve the runtime for optimization of VDMF, Section 3.7 investigates the effects of adding interdependencies between adjacent stories and utilizing a method presented in Section 2.7 for reduction of initially defined wide ranges of damping coefficients.

TABLE 16. Effect of Broad Ranges of Damping Coefficients on Runtime

Model	Runtime (s)	Runtime (hour)
CF7-S15- IER1	3.934731963091945e+04	10.93
CF7-S15- IER2	3.489704382271593e+04	9.69

3.7 Ranges and Interdependencies of Dampers

Section 3.6 demonstrates that MOPSO has the capability to find non-convex solutions of VDMF; however, it also shows that model runtimes increase drastically when broad ranges of dampers' coefficients are selected.

Section 2.7 of Chapter 2 suggests approaches to finding both interdependencies between dampers in adjacent stories and effective ranges of damping coefficients. This section, first, investigates the models with and without interdependencies of dampers in adjacent stories. Next, it investigates the effect of combining approaches for establishing interdependencies with effective ranges for damping coefficients. Table 17 shows the model CF7-S15-I, where "I" stands for interdependencies. The solution outcomes of model that considers interdependences

(CF7-S15-I) are compared with the outcomes of model without considering interdependences (CF7-S15) in Figure 39.

TABLE 17. Interdependencies of Dampers in Adjacent Floors

Model	Constraints	Cost Function	Damping Coefficient (kip/(in/s)$^{\alpha}$)
CF7-S15-I	Drift 2% Acc 0.85g Res Drift 0.5%	Max Drift – Max Acc – Max Res Drift – C12 – C3 – C12+C3	C12 [120 160] C3 [20 60]
Results of Interdependencies: C 3 < C12 & 51.72 (kip/(in/s)$^{\alpha}$) < C12-C3			
Acc: Acceleration; Res Drift: Residual Drift; g: Gravitational Acceleration			

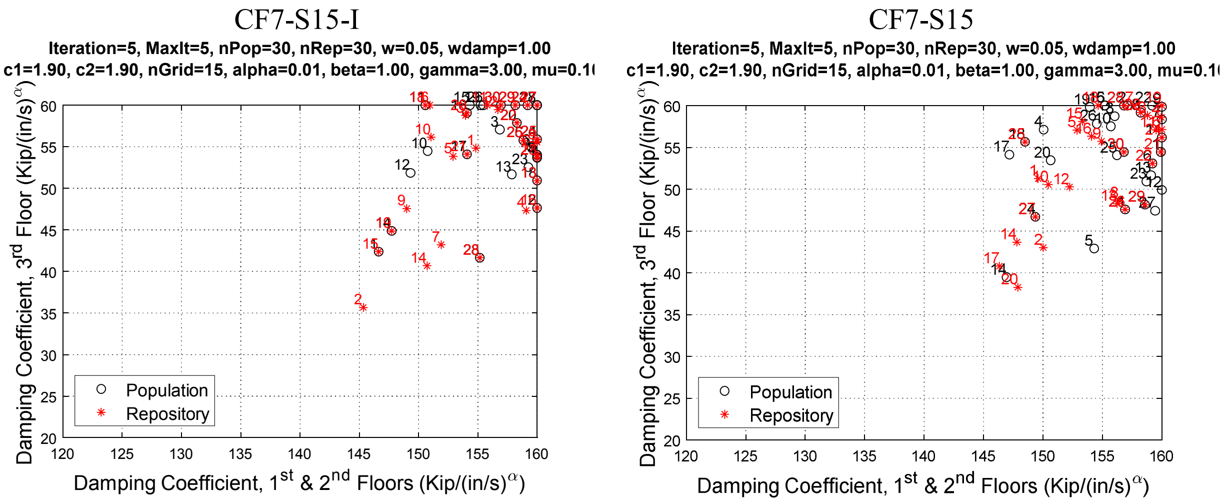


FIGURE 39. Comparison of solution outcomes with and without considering interdependencies.

The presented results demonstrate that consideration of interdependencies between dampers of adjacent stories produce slightly wider (better) range of solution outcomes. The major advantage when considering independencies is realized through shorter runtime, 4.22 hours, as opposed to 5.37 hours for the case that did not consider interdependences.

The effect of combining interdependencies with the effective range of damping coefficients was next investigated. This model is named CF7-S15-IR and its constraints, cost function, and ranges of damping coefficients are presented in Table 18. In the model designation "IR" stands for interdependencies and ranges of damping coefficients. The ranges of damping coefficients were set to 0 - 500 for all floors to cover very broad range of possible damper coefficients. The runtime for defining the interdependencies and effective range following the procedure of Section 2.7 was about forty minutes. The results of MOSPO optimization of VDMF are shown in Figure 40, which depicts the Pareto front for each pair of considered EDPs, as well as the sum of EDPs for each population and repository (solution).

TABLE 18. Model that Considers Broad Ranges of Damping Coefficients and Interdependencies between Dampers of Adjacent Stories.

Model	Constraints	Cost Function	Damping Coefficient (kip/(in/s)$^{\alpha}$)
CF7-S15-IR	Drift 2% Acc 0.85g Res Drift 0.5%	Max Drift – Max Acc – Max Res Drift – C12 – C3 – C12+C3	C12 [0 500] C3 [0 500]
Results of Interdependencies: C 3 < C12 & 51.72 (kip/(in/s)$^{\alpha}$) < C12-C3 Effective Ranges: 68.9655 < C12 < 206.8966 & 206.8966 < C3 < 17.2414 (kip/(in/s)$^{\alpha}$)			
Drift: Inter-Story Drift; Acc: Floor Acceleration; Res Drift: Residual Inter-Story Drift; g: Gravitational Acceleration			

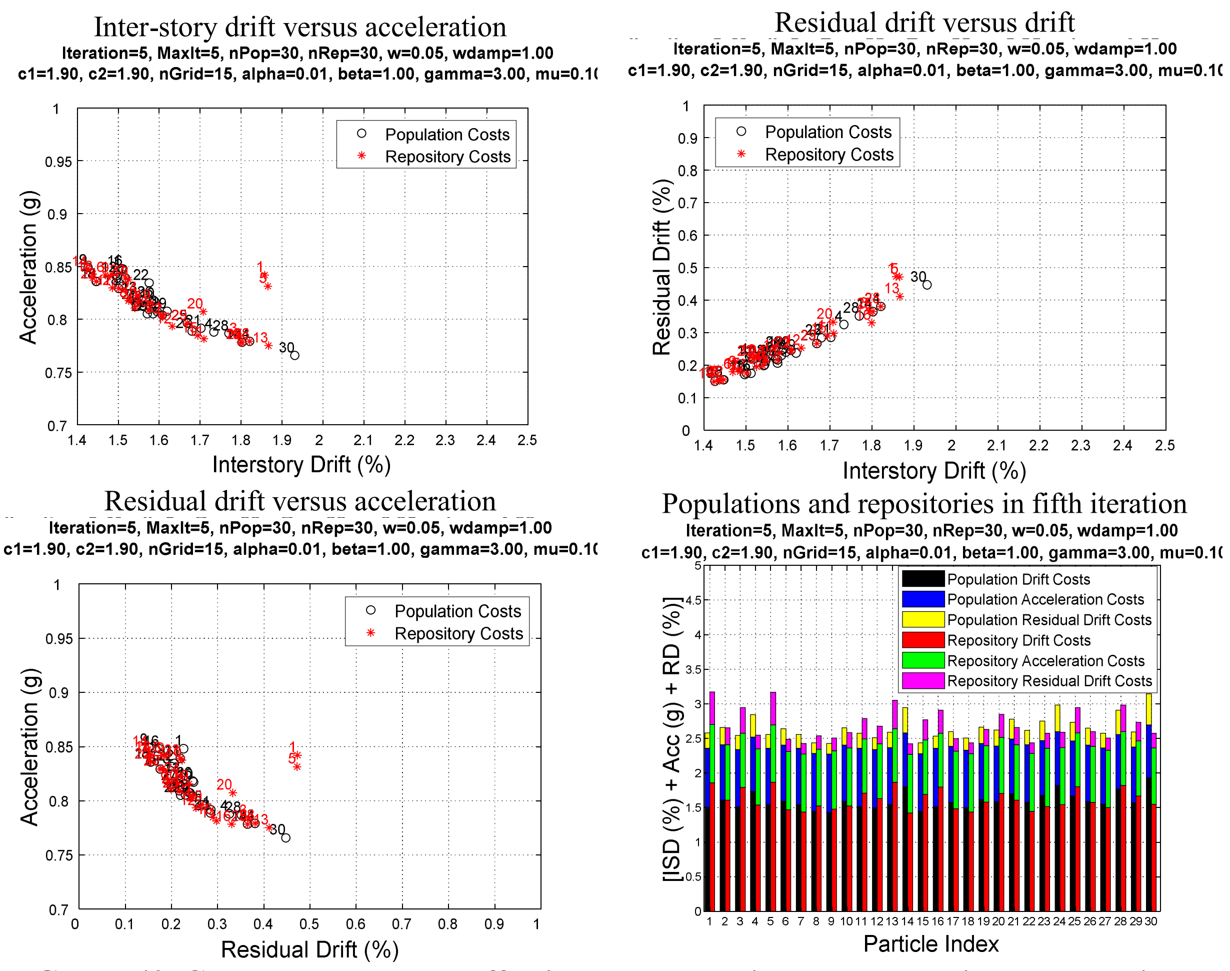


FIGURE 40. Cost outcomes when effective ranges and interdependencies were considered.

The runtime of CF7-S15-IR was 5.82 hours, which is significantly shorter than runtimes of models CF7-S15-R1 and CF7-S15-R2 that considered wide ranges of damping coefficients and took 10.93 and 9.69 hours, respectively.

Figure 41 further shows outcomes of decision variables. Note that repositories one and five meet constrained requirements for smallest values of damping coefficients. Moreover, the close-up views of these two solutions show similar solutions to those of CF7-S15-R1 and CF7-S15-R2 models, where the initial ranges of damping coefficients were broad.

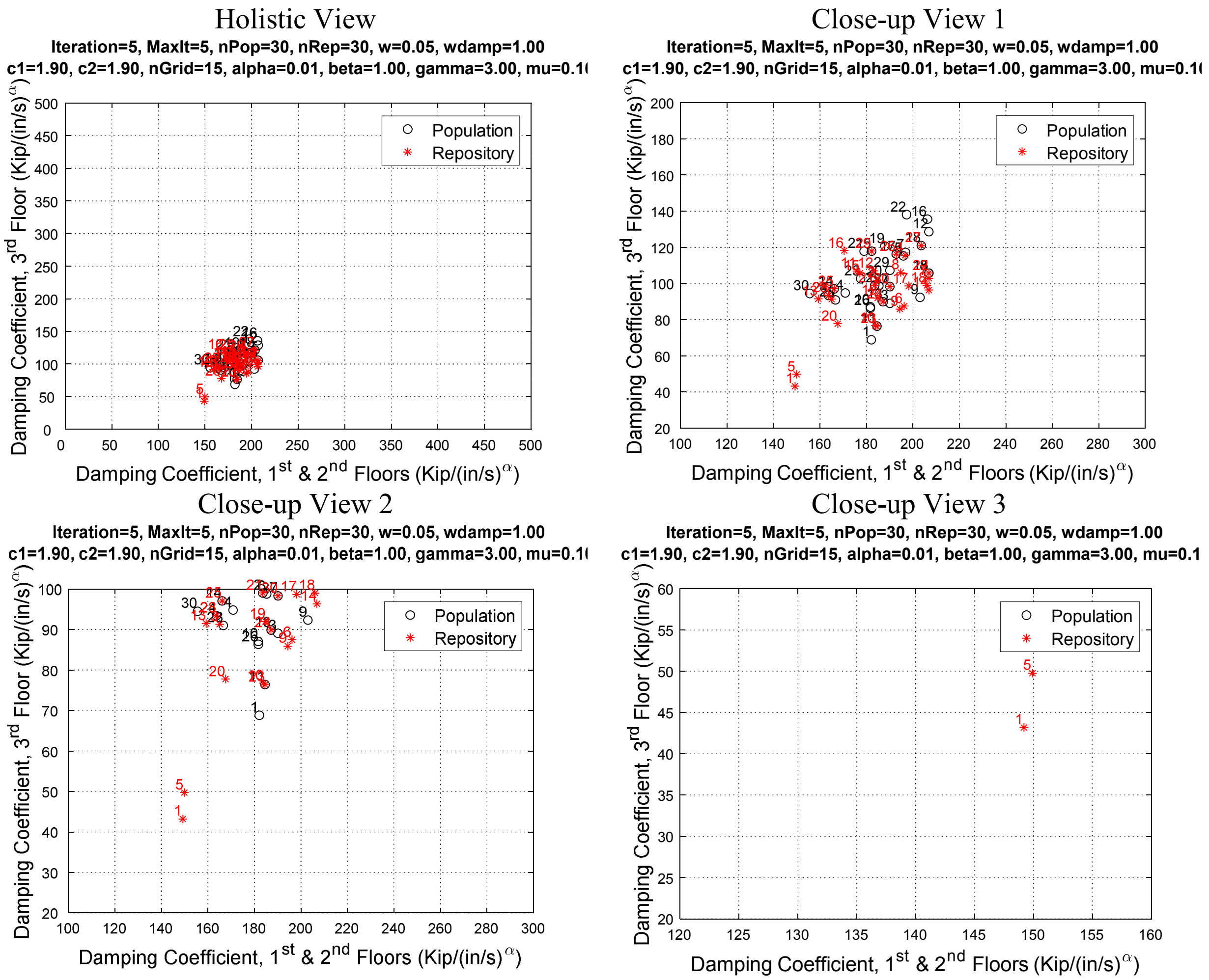


FIGURE 41. Solution outcomes when effective ranges and interdependencies were considered.

Figure 42 shows the profiles of inter-story drift, acceleration, and residual drift of the solution outcomes. The profiles of the EDPs indicate that solutions one (blue) and five (magenta) satisfy the constraints on all EDPs.

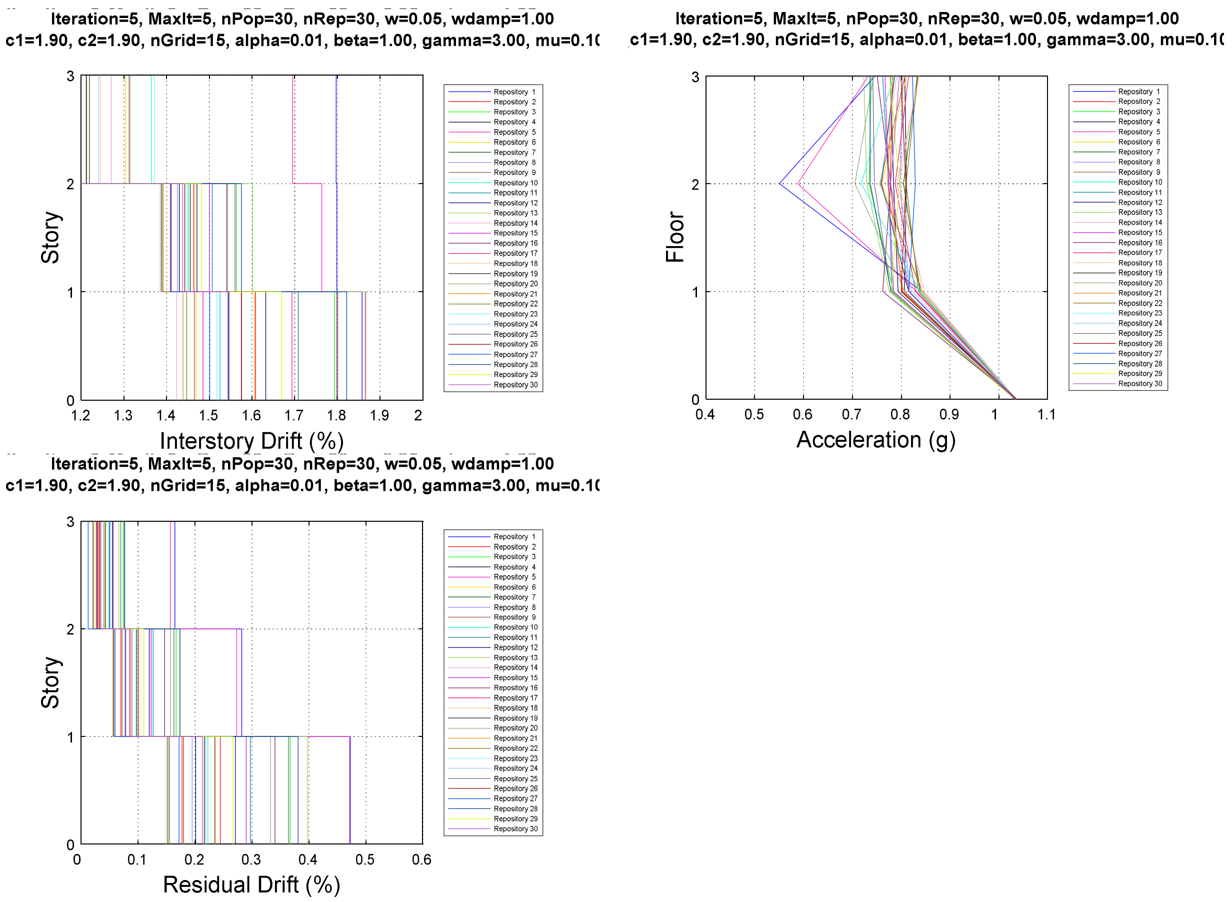


FIGURE 42. Outcomes of adding effective ranges and interdependencies, profiles of EDPs.

The case study presented in this chapter provides recommendations on how to implement MOPSO for design optimization of a VDMF. Moreover, it was shown that the runtime of the optimization can be significantly reduced by utilizing the approach introduced in Section 2.7, which considers the interdependencies between dampers of adjacent stories and finds effective ranges of damping coefficients for every story (starting from a very broad range).

In Chapter 4, the solutions of optimization are compared to the VDMF previously designed by Miyamoto International. Even though the main focus of this thesis is on optimization of VDMF utilizing MOPSO, the additional observations regarding the performance of VDMF are presented in Section 3.8.

3.8 Observation Regarding Placement of Dampers in VDMF

Figure 7 of Chapter 3 shows that the dampers of the VDMF were placed in the first and the sixth bay. While the first bay constitutes part of the moment frame, the sixth bay is the gravity frame. Figures 43-44 show the dampers' forces versus their elongations for all floors. It is apparent from the figures that the dampers in the first bay generate significantly larger forces that those in the sixth bay.

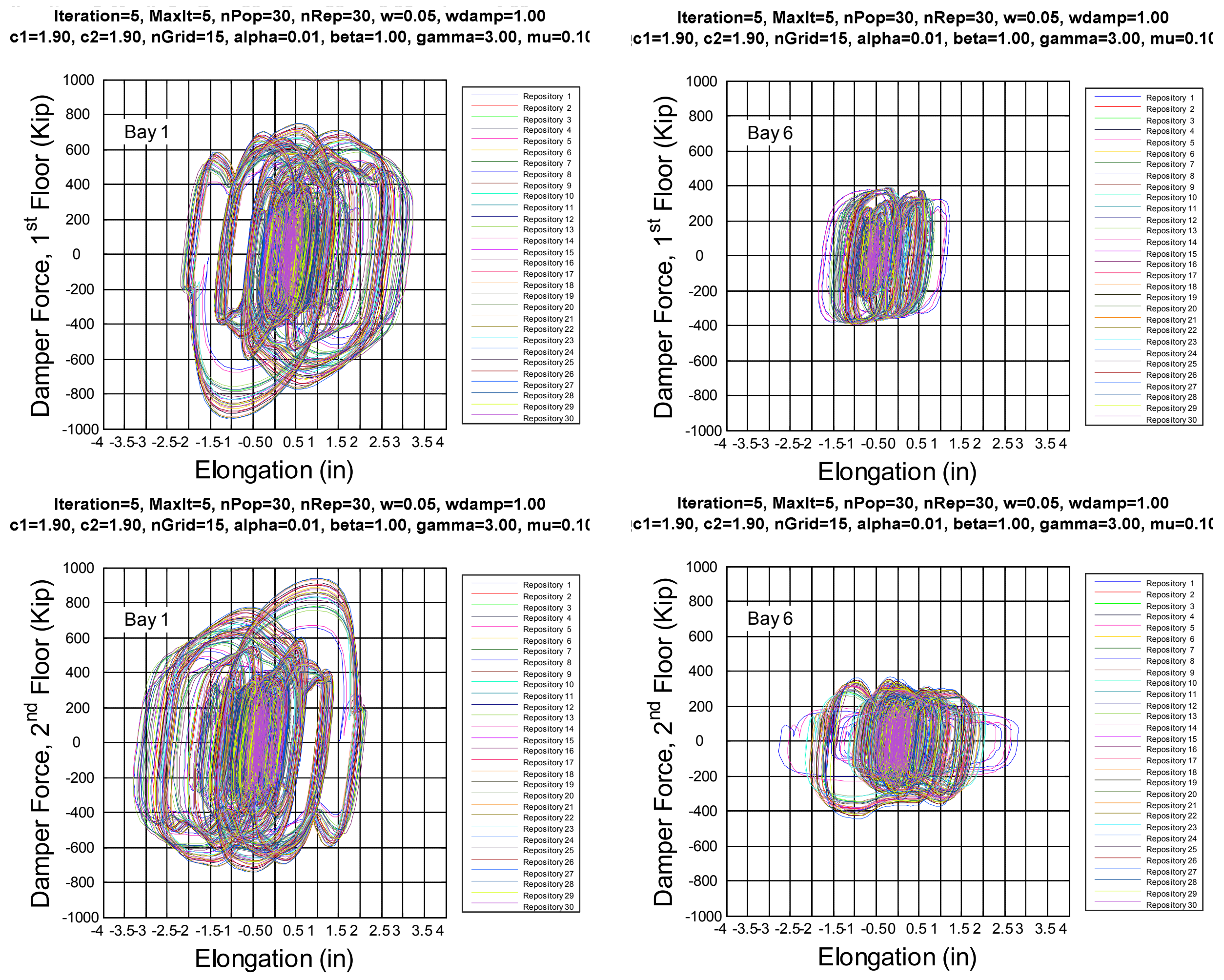


FIGURE 43. The dampers' forces versus elongations, in bays one and six, first and second floors.

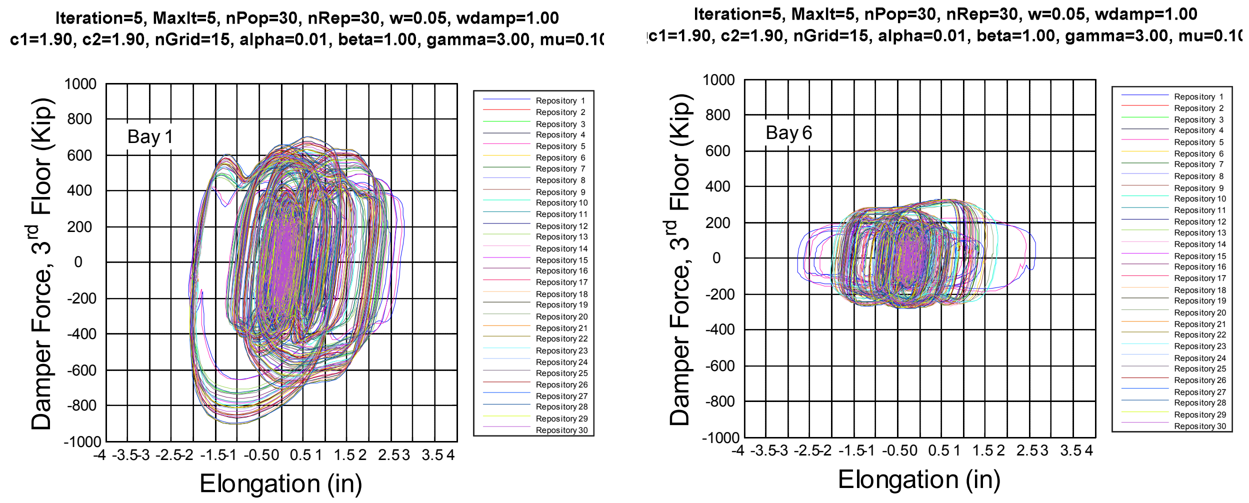


FIGURE 44. The dampers' forces versus elongations, in bays one and six, third floor.

Furthermore, the forces versus the elongations of dampers for repositories one and five are shown in Figures 45-46.

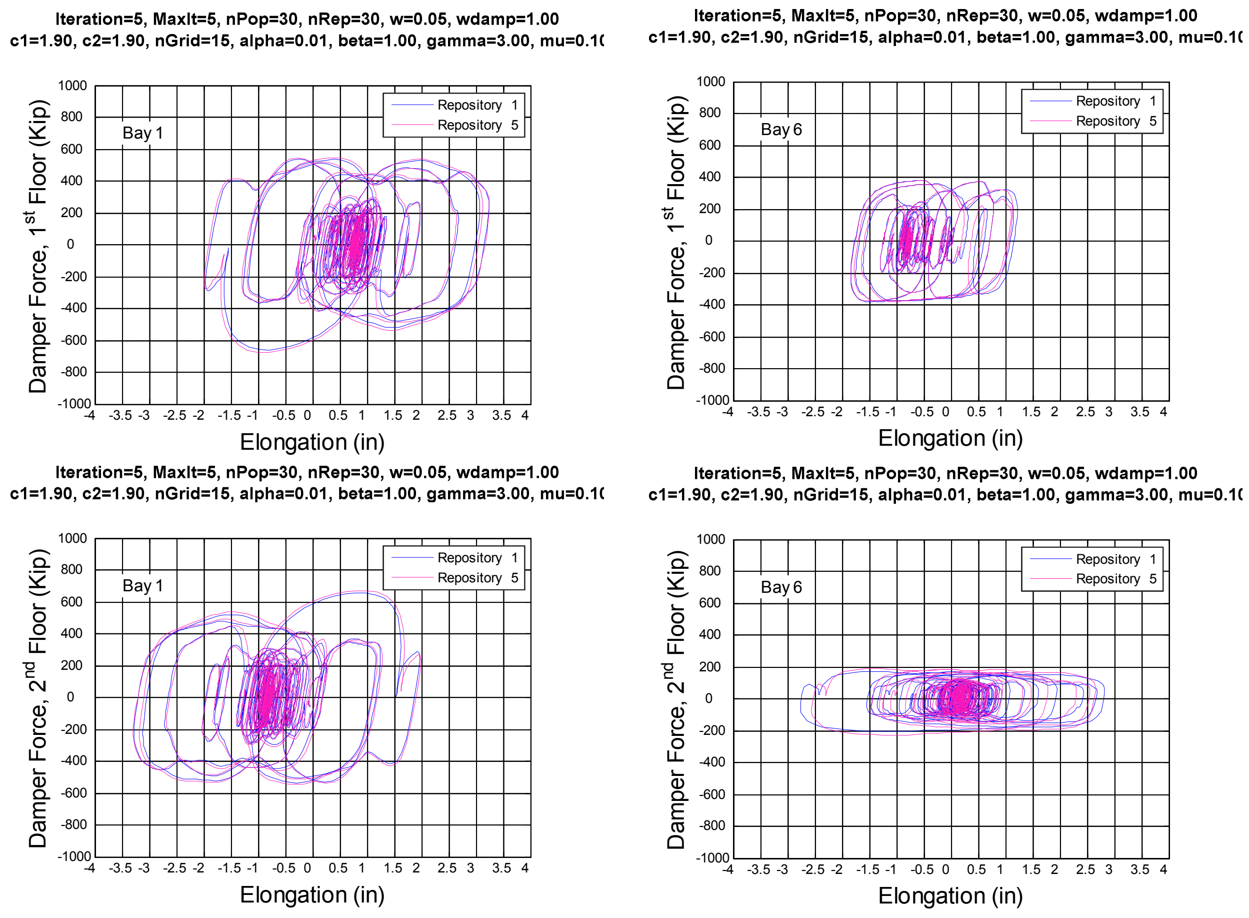
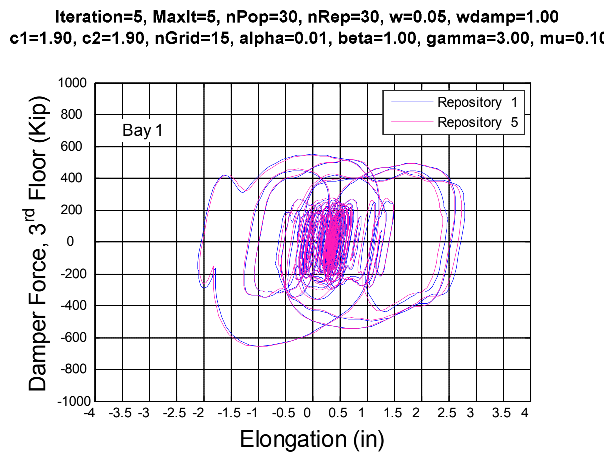


FIGURE 45. The dampers' forces versus elongations, in bays one and six (repositories one and five), first and second floors.



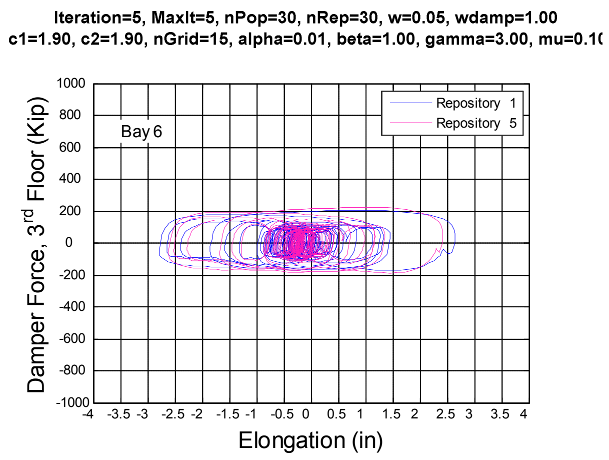


FIGURE 46. The dampers' forces versus elongations, in bays one and six (repositories one and five), third floor.

For these two repositories, dampers in the sixth bay of the second and third story generate smaller forces but slightly higher elongations than the dampers of the first bay. The higher elongations result in higher strokes of dampers for these two repositories (stroke is the maximum elongation of damper). The strokes of the dampers are indirectly controlled by the constraint associated with the maximum inter-story drift and the parameters of maximum inter-story drift of the cost function.

CHAPTER 4

VERIFICATION OF THE OPTIMIZATION FRAMEWORK

The VDMF described in Section 3.1 was designed by structural engineering company. Dampers of this VDMF utilized velocity exponent of 0.5 and damping coefficients of 135 and 35 (kip/(in/s)$^{\alpha}$) for the bottom two stories and the third story, respectively (see Table 6). Three ground motions were used for design of VDMF. The pseudo-acceleration response spectra of the considered ground motions along with the designed spectrum are shown in Figure 47. The main goal of this chapter is verification of the proposed framework through the comparison of an engineered solution with the MOPSO solutions generated under the set of the design constraints.

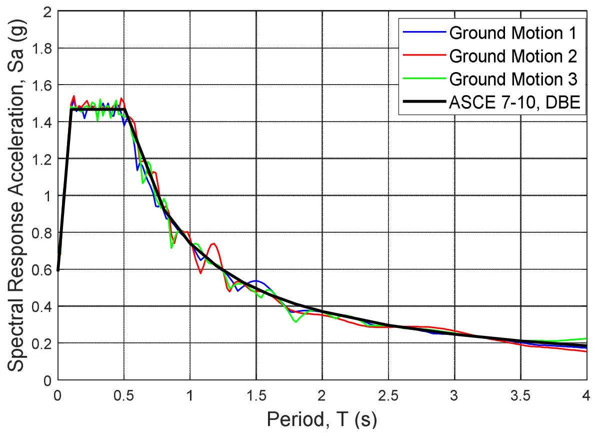


FIGURE 47. The design spectra, and ground motions selected for designed VDMF.

The results of designed VDMF subjected to ground motions showed the maximum inter-story drift, floor acceleration, and residual drift of 0.9908(%), 0.4792(g), and 0.018(%), respectively. The constraints utilized in the design were set to 1.2(%) for inter-story drift, 0.5(g) for floor acceleration, and 0.2(%) for inter-story residual drift. To compare designed VDMF with the MOPSO solutions, the optimization analysis is conducted considering the same set of constraints and broad range of damping coefficients. The MOPSO parameters utilized for finding the set of optimal solutions stem from studies presented in Chapter 3 and are summarized in Table 19. Furthermore, two sets of cost functions are considered and represented by: 1) combination of inter-story drift, floor acceleration, residual drift, damping coefficients at different stories and their linear combinations (designated as Optimal Solution 1) and 2) combination of damper stroke, floor acceleration, residual drift, damper forces at different stories and their linear combinations (designated as Optimal Solution 2).

TABLE 19. Optimal Parameters of MOPSO for VDMF Subjected to Earthquakes

Parameter	Value
MaxIt	2~5
nPop	30
nRep	30
w	0.05
wdamp	1
c1	1.9
c2	1.9
nGrid	15
alpha	0.01
beta	1
gamma	3
mu	0.1

4.1 MOPSO Optimal Solution 1

To generate the MOPSO Optimal Solution 1, the model has utilized the set of design constraints, the cost function that considers damping coefficients, very broad initial range of damping coefficients (assuming no engineering knowledge), all of which are summarized in Table 20.

TABLE 20. MOPSO model for generation of Optimal Solution 1

Model	Constraints	Cost Function	Damping Coefficient (kip/(in/s)$^{\alpha}$)
Optimal Solution 1	Drift 1.2% Acc 0.5g Res Drift 0.2%	Max Drift – Max Acc – Max Res Drift – C12 – C3 – C12+C3	C12 [0 300] C3 [0 300]
Results of Interdependencies: C 3 < C12 & 20.68 (kip/(in/s)$^{\alpha}$) < C12-C3 Effective Ranges: 51.72 < C12 < 134.48 & 31.03 < C3 < 175.86 (kip/(in/s)$^{\alpha}$)			
Drift: Inter-Story Drift; Acc: Floor Acceleration; Res Drift: Residual Inter-Story Drift; g: Gravitational Acceleration			

The Table 20 also shows the effective ranges of damping coefficients and interdependence relationships between dampers of adjacent stories generated using the proposed strategy described in Section 2.7.

The runtime of MOPSO for five iterations was 3.527497501297082e+04 (s) (9.80 hours) and the runtime for finding interdependence relationships of adjacent dampers and the effective ranges of damping coefficients was 3.1431e+03 (s) (0.87 hours), resulting in the total runtime of 10.67 hours. The model was run utilizing the Computation Platform II, introduced in Chapter 2.

Figure 48 shows the Pareto fronts considering different pairs of EDPs: 1) inter-story drift versus floor acceleration, 2) inter-story drift versus residual drift, and 3) residual drift versus floor acceleration.

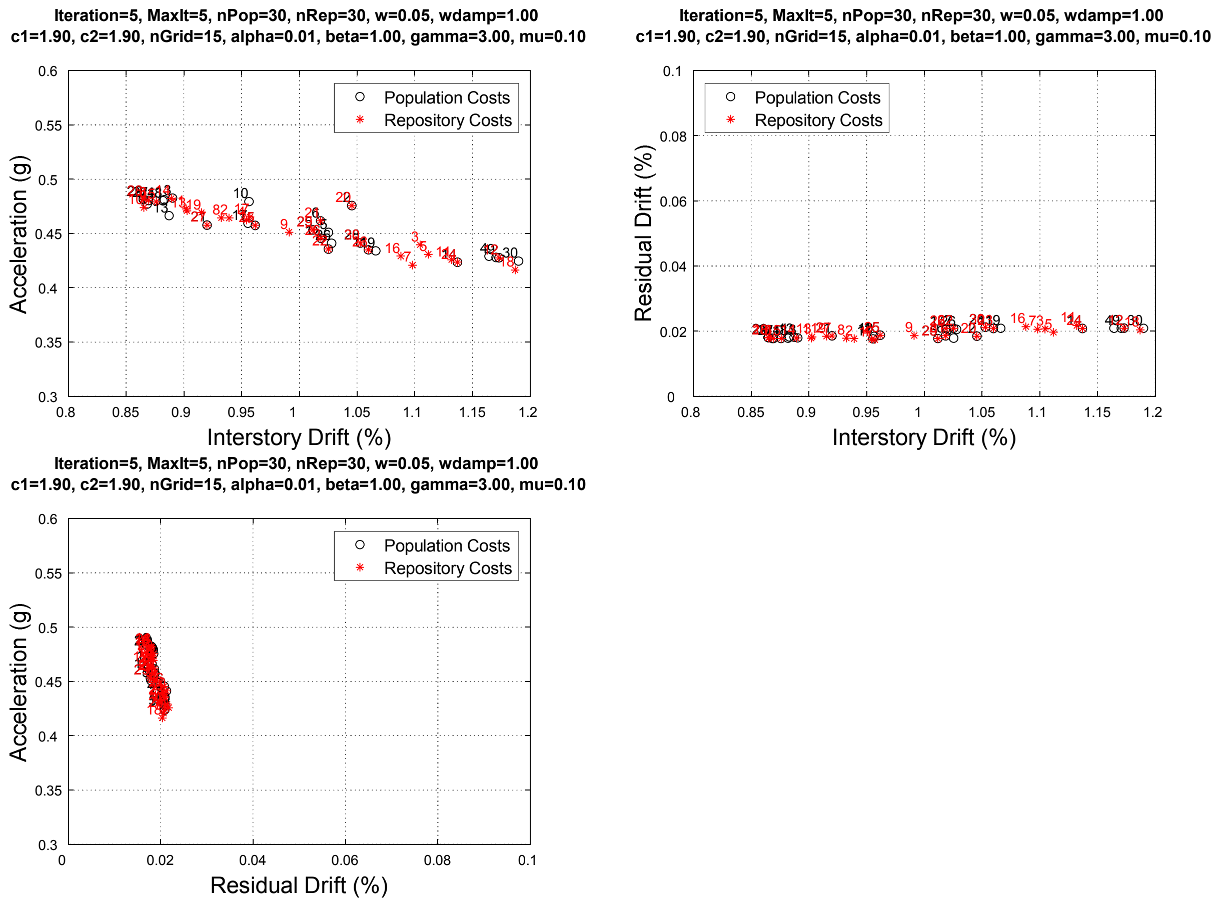


FIGURE 48. Pareto fronts of EDPs for MOPSO Optimal Solution 1.

Under the given set of constraints, very reach set of solution outcomes was generated. Furthermore, the positions of the main decision variables, damping coefficients of the bottom two stories and the third story are shown in Figure 49.

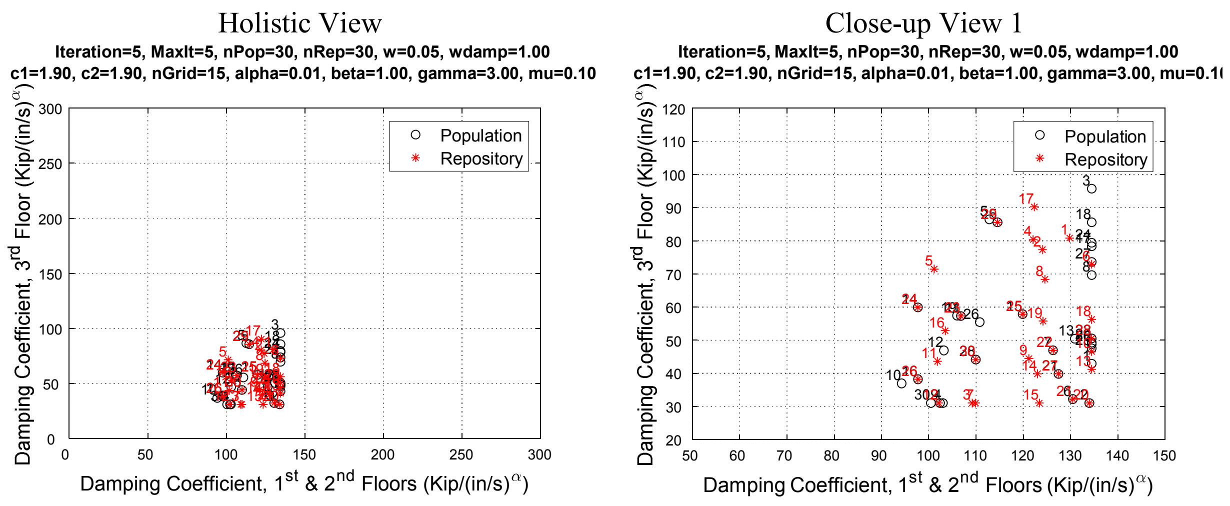


FIGURE 49. Damping Coefficients for MOPSO Optimal Solution 1.

Moreover, the repositories of MOPSO (solutions) and their produced costs are shown in Tables 21-22. The user can use this information to choose the design variables.

TABLE 21. Optimal Solution 1: Design Parameters of Dampers and Associated Costs (Part A: Repositories 1 to 15)

Solution: [axial stiffness of dampers; C12; C3; α], for units refer to Table 6					Cost Function: [max drift; max acceleration; max residual drift; C12; C3; C12+C3]					
1	2000	129.8	80.8	0.5	0.90	0.47	0.018	129.8	80.8	210.6
2	2000	124.1	77.3	0.5	0.94	0.46	0.018	124.1	77.3	201.4
3	2000	109.2	31.0	0.5	1.10	0.44	0.021	109.2	31.0	140.2
4	2000	122.1	80.4	0.5	0.95	0.46	0.018	122.1	80.4	202.5
5	2000	101.2	71.5	0.5	1.11	0.43	0.020	101.2	71.5	172.6
6	2000	134.5	72.9	0.5	0.87	0.48	0.018	134.5	72.9	207.4
7	2000	109.9	31.0	0.5	1.10	0.42	0.021	109.9	31.0	140.9
8	2000	124.6	68.4	0.5	0.93	0.46	0.018	124.6	68.4	193.0
9	2000	121.2	44.5	0.5	0.99	0.45	0.019	121.2	44.5	165.6
10	2000	134.5	46.6	0.5	0.87	0.47	0.018	134.5	46.6	181.0
11	2000	101.8	43.6	0.5	1.13	0.43	0.022	101.8	43.6	145.5
12	2000	102.1	31.0	0.5	1.17	0.43	0.021	102.1	31.0	133.2
13	2000	134.5	41.2	0.5	0.90	0.47	0.018	134.5	41.2	175.7
14	2000	123.0	39.9	0.5	0.89	0.48	0.018	123.0	39.9	162.9
15	2000	123.4	31.0	0.5	0.88	0.48	0.018	123.4	31.0	154.4

TABLE 22. Optimal Solution 1: Design Parameters of Dampers and Associated Costs (Part B: Repositories 16 to 30)

Solution: [axial stiffness of dampers; C12; C3; α], for units refer to Table 6					Cost Function: [max drift; max acceleration; max residual drift; C12; C3; C12+C3]					
16	2000	103.5	52.9	0.5	1.09	0.43	0.021	103.5	52.9	156.4
17	2000	122.3	90.3	0.5	0.96	0.47	0.017	122.3	90.3	212.6
18	2000	134.5	56.3	0.5	1.19	0.42	0.020	134.5	56.3	190.8
19	2000	124.2	55.8	0.5	0.92	0.47	0.018	124.2	55.8	179.9
20	2000	134.0	31.0	0.5	1.05	0.48	0.018	134.0	31.0	165.0
21	2000	130.5	32.2	0.5	1.02	0.46	0.019	130.5	32.2	162.7
22	2000	126.3	47.0	0.5	1.03	0.44	0.021	126.3	47.0	173.3
23	2000	106.8	57.3	0.5	1.06	0.43	0.021	106.8	57.3	164.1
24	2000	97.7	59.9	0.5	1.14	0.42	0.021	97.7	59.9	157.6
25	2000	119.9	57.9	0.5	0.96	0.46	0.019	119.9	57.9	177.7
26	2000	97.7	38.2	0.5	1.02	0.45	0.021	97.7	38.2	135.9
27	2000	127.5	39.9	0.5	0.92	0.46	0.019	127.5	39.9	167.3
28	2000	134.5	50.5	0.5	0.86	0.48	0.018	134.5	50.5	185.0
29	2000	114.5	85.6	0.5	1.01	0.45	0.018	114.5	85.6	200.1
30	2000	109.9	44.1	0.5	1.05	0.44	0.021	109.9	44.1	154.1

Moreover, the profiles of inter-story drifts, floor accelerations, and residual drifts for both Optimal Solution 1 and original design of VDMF are illustrated in Figure 50. The profiles of EDPs for designed VDMF are shown in black dashed line. The presented results show that MOPSO solutions encompass the design solution while providing significantly larger set of options, some of which can reduce the manufacturing cost.

The manufacturing cost of a damper is directly related to the damper force capacity and its stroke (maximum elongations). Figure 51 depicts the profile of maximum dampers' forces and their maximum deformations (strokes) for the MOPSO solutions and the original VDMF design. This information could be used to select the dampers that would yield cost savings. Further, Table 23 shows the MOPSO repositories (solutions) and associated values of dampers' forces and their maximum strokes. Dampers forces are shown in units of kips and strokes are in units of inches.

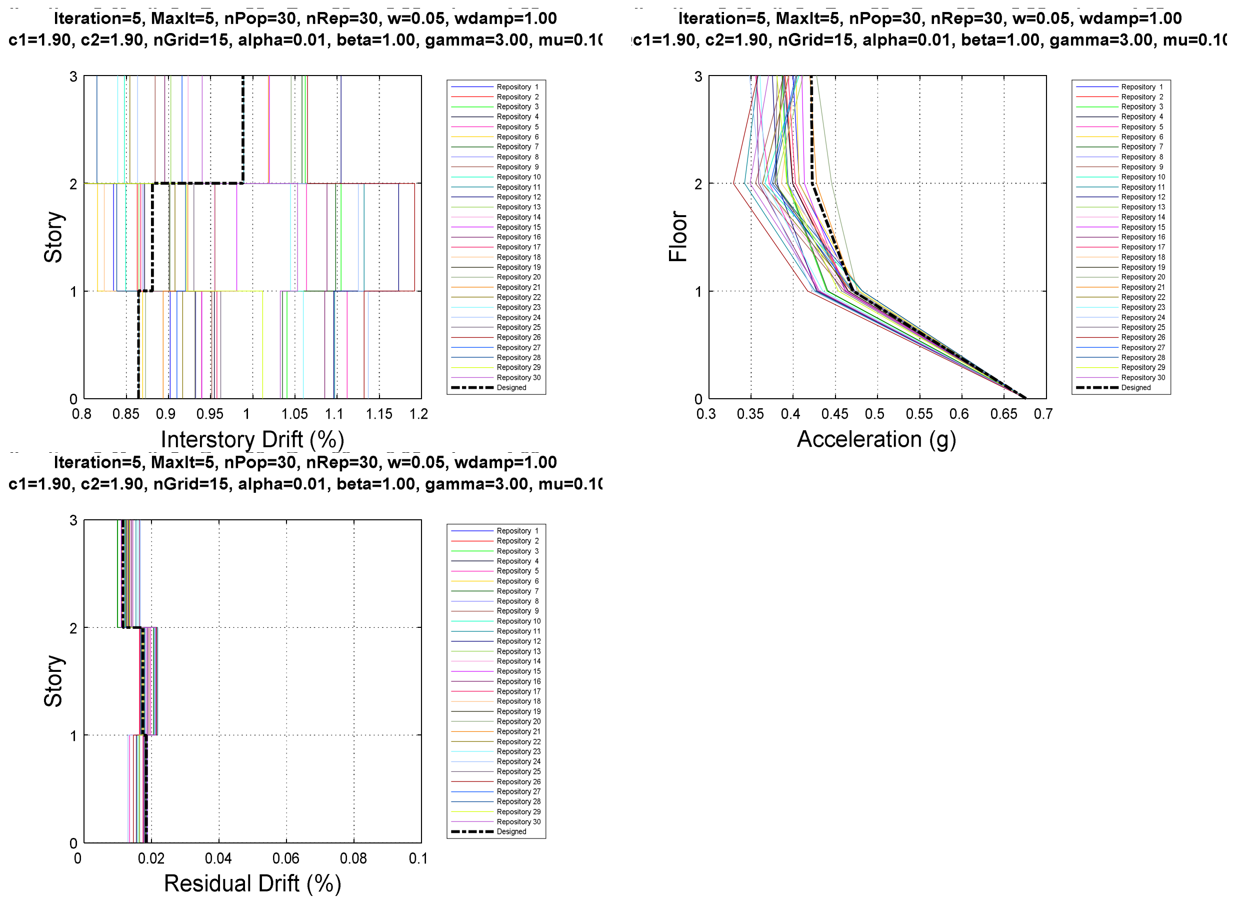


FIGURE 50. Profiles of EDPs for Optimal Solution 1 versus the designed VDMF.

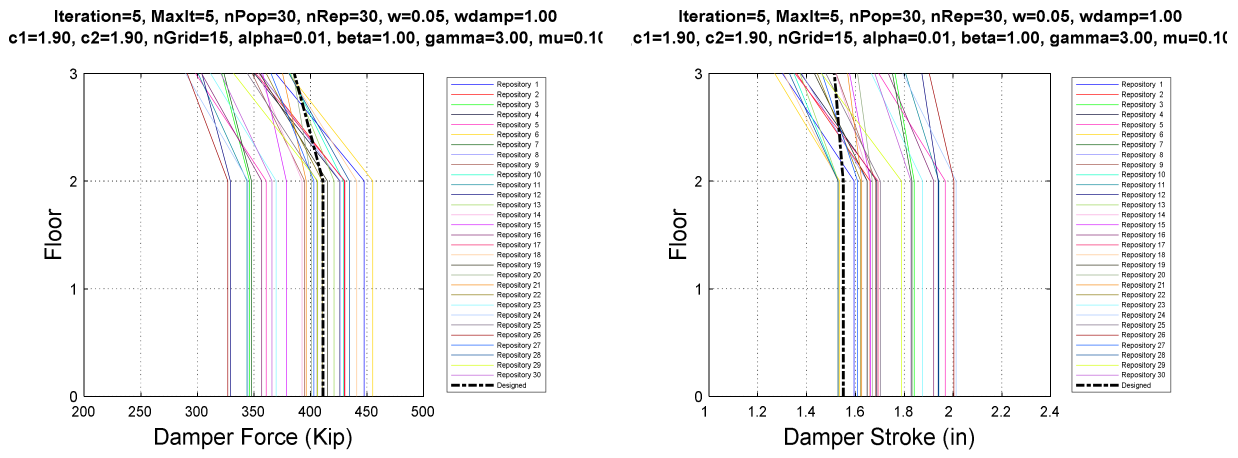


FIGURE 51. Dampers' forces and strokes for MOPSO Optimal Solution 1 and original VDMF design.

TABLE 23. Values of Dampers' Forces and Strokes for MOPSO Optimal Solution 1

Solution: [axial stiffness of dampers; C12; C3; α], for units refer to Table 6					Stroke 12, max force of dampers (first and second floors); stroke 3, max force of dampers (third floor)			
1	2000	129.8	80.8	0.5	1.59	447.36	1.30	369.42
2	2000	124.1	77.3	0.5	1.66	430.00	1.36	355.07
3	2000	109.2	31.0	0.5	1.84	346.13	1.76	321.64
4	2000	122.1	80.4	0.5	1.69	425.92	1.38	350.35
5	2000	101.2	71.5	0.5	1.97	360.85	1.69	298.47
6	2000	134.5	72.9	0.5	1.54	455.18	1.27	380.53
7	2000	109.9	31.0	0.5	1.83	347.79	1.75	323.41
8	2000	124.6	68.4	0.5	1.65	426.15	1.38	355.83
9	2000	121.2	44.5	0.5	1.68	394.71	1.52	348.49
10	2000	134.5	46.6	0.5	1.53	428.96	1.35	381.46
11	2000	101.8	43.6	0.5	1.94	344.01	1.81	300.58
12	2000	102.1	31.0	0.5	1.94	329.03	1.87	303.85
13	2000	134.5	41.2	0.5	1.53	420.88	1.37	382.47
14	2000	123.0	39.9	0.5	1.66	392.66	1.52	354.03
15	2000	123.4	31.0	0.5	1.66	378.56	1.58	357.68
16	2000	103.5	52.9	0.5	1.92	356.88	1.73	303.83
17	2000	122.3	90.3	0.5	1.69	430.49	1.35	351.66
18	2000	134.5	56.3	0.5	1.53	441.02	1.31	380.52
19	2000	124.2	55.8	0.5	1.65	414.94	1.43	354.82
20	2000	134.0	31.0	0.5	1.67	401.03	1.61	384.72
21	2000	130.5	32.2	0.5	1.63	396.11	1.57	375.27
22	2000	126.3	47.0	0.5	1.62	410.35	1.44	360.86
23	2000	106.8	57.3	0.5	1.88	369.50	1.67	311.77
24	2000	97.7	59.9	0.5	2.01	344.79	1.79	289.56
25	2000	119.9	57.9	0.5	1.70	405.71	1.48	344.02
26	2000	97.7	38.2	0.5	2.00	327.02	1.90	291.11
27	2000	127.5	39.9	0.5	1.61	402.90	1.46	365.21
28	2000	134.5	50.5	0.5	1.53	434.26	1.33	380.97
29	2000	114.5	85.6	0.5	1.79	406.26	1.46	332.08
30	2000	109.9	44.1	0.5	1.83	366.08	1.68	320.63
Designed VDMF					Designed VDMF			
	2000	135	135	0.5	1.55	411.07	1.51	385.76

4.2 MOPSO Optimal Solution 2

To generate the MOPSO Optimal Solution 2, the model has utilized the set of design constraints, the cost function that considers damping force and stroke, the very broad initial range of damping coefficients (assuming no engineering knowledge) as shown in Table 24. The table also shows the effective ranges of damping coefficients and interdependence relationships between dampers of adjacent stories generated using the proposed strategy described in Section 2.7.

TABLE 24. MOPSO Model for Generation of Optimal Solution 2

Model	**Constraints**	**Cost Function**	**Damping Coefficient (kip/in/s)$^{\alpha}$)**
Optimal Solution 2	Drift 1.2% Acc 0.5g Res Drift 0.2%	Max Stroke – Max Acc – Max Res Drift – F12 – F3 – F12+F3	C12 [0 300] C3 [0 300]
Results of Interdependencies: C 3 < C12 & 20.68 (kip/(in/s)$^{\alpha}$) < C12-C3 Effective Ranges: 51.72 < C12 < 134.48 & 31.03 < C3 < 175.86 (kip/(in/s)$^{\alpha}$)			
Drift: Inter-Story Drift; Acc: Floor Acceleration; Res Drift: Residual Inter-Story Drift; g: Gravitational Acceleration; F12: Maximum Force of Dampers of First and Second Floors; F3: Maximum Force of Dampers of Third Floor			

The runtime of MOPSO for five iterations for Optimal Solution 2 was 1.741430219869433e+04 (s) (4.84 hours) and the runtime for finding interdependencies of adjacent dampers and the ranges was 3.1431e+03 (s) (0.87 hours), the total run was approximately 5.71 hours. The model was run in Platform II. Pareto fronts of stroke versus acceleration, stroke versus residual drift, residual drift versus acceleration, and forces of dampers are depicted in Figure 52.

The results of Optimal Solution 2 are similar to the results of Optimal Solution 1. The results indicate that considering either damping coefficients or forces of dampers can reach to the similar solutions. Further, maximum stroke can be used in cost function instead of maximum drift.

Tables 25-26 Show the solutions of Optimal Solution 2. The units for dampers forces are kips and for strokes are inches.

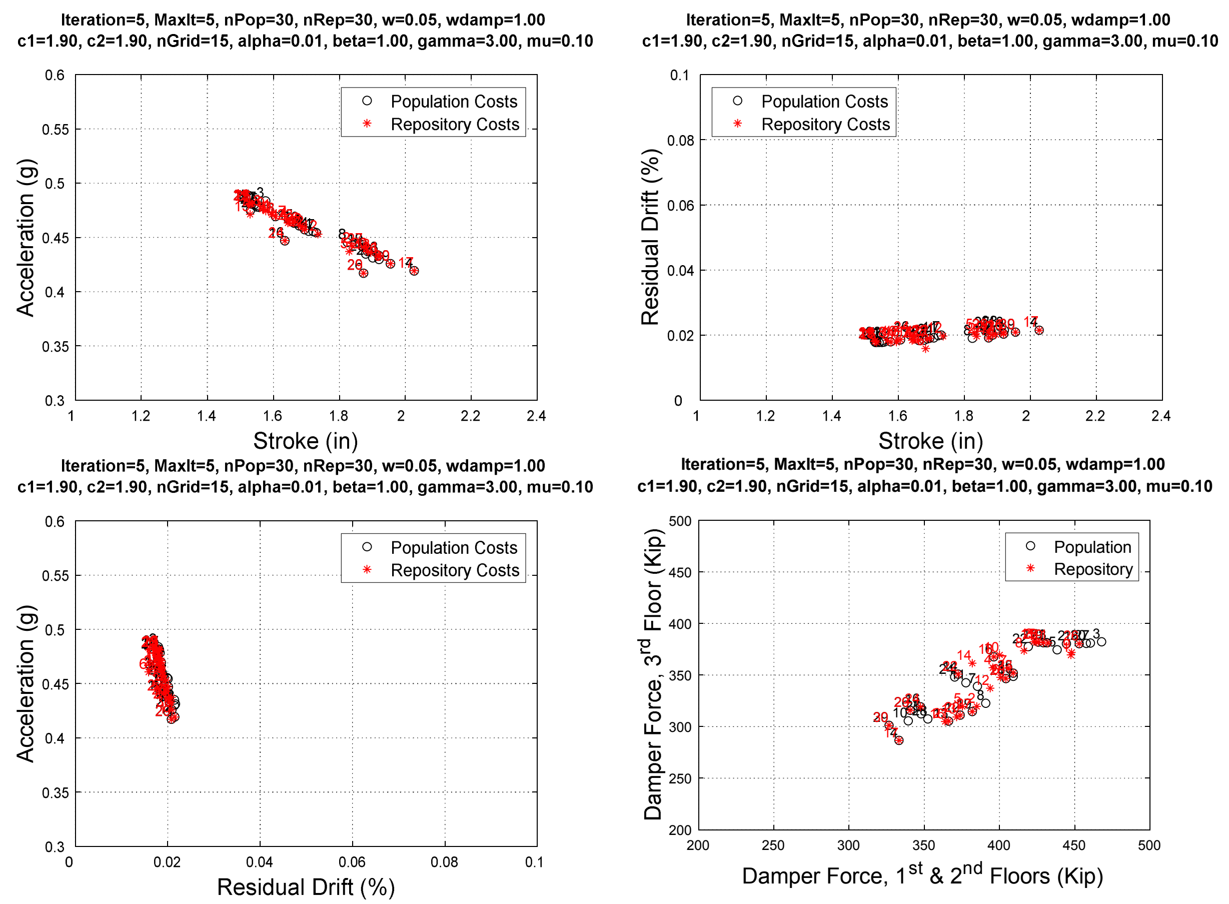


FIGURE 52. Pareto fronts of EDPs and forces for MOPSO Optimal Solution 2.

TABLE 25. Values of Dampers' Forces and Strokes for MOPSO Optimal Solution 2 (Part A: Repositories 1 to 13)

Solution: [axial stiffness of dampers; C12; C3; α], for units refer to Table 6					Stroke 12, max force of dampers (first and second floors), stroke 3, max force of dampers (third floor)			
1	2000	129.80	80.83	0.5	1.59	447.36	1.30	369.42
2	2000	109.94	67.23	0.5	1.84	384.79	1.58	319.58
3	2000	134.48	42.39	0.5	1.47	353.80	1.35	304.97
4	2000	124.21	39.94	0.5	1.65	395.56	1.50	357.03
5	2000	109.94	53.52	0.5	1.83	375.30	1.64	319.65
6	2000	121.22	49.31	0.5	1.68	400.68	1.50	347.84
7	2000	124.60	46.54	0.5	1.64	405.68	1.47	356.64
8	2000	131.11	43.09	0.5	1.57	416.21	1.40	373.57
9	2000	131.00	75.84	0.5	1.58	447.89	1.29	372.08
10	2000	128.75	36.36	0.5	1.60	400.02	1.47	369.46
11	2000	134.48	48.96	0.5	1.53	432.26	1.34	381.14
12	2000	117.08	52.82	0.5	1.74	393.74	1.54	337.25
13	2000	105.74	65.19	0.5	1.90	371.50	1.65	309.31

TABLE 26. Values of Dampers' Forces and Strokes for MOPSO Optimal Solution 2 (Part B: Repositories 14 to 30)

	Solution: [axial stiffness of dampers; C12; C3; α], for units refer to Table 6				Stroke 12, max force of dampers (first and second floors), stroke 3, max force of dampers (third floor)			
14	2000	124.88	31.03	0.5	1.64	381.81	1.58	361.47
15	2000	134.48	43.45	0.5	1.47	354.42	1.33	306.26
16	2000	103.97	60.79	0.5	1.92	363.79	1.69	304.90
17	2000	96.39	48.48	0.5	2.03	333.23	1.87	286.60
18	2000	127.89	35.25	0.5	1.61	396.17	1.49	367.64
19	2000	134.48	46.69	0.5	1.53	429.15	1.35	381.44
20	2000	106.43	65.82	0.5	1.89	373.85	1.64	311.01
21	2000	104.07	64.29	0.5	1.92	366.12	1.68	305.22
22	2000	120.66	31.03	0.5	1.69	372.53	1.60	350.74
23	2000	122.98	52.94	0.5	1.66	409.04	1.46	351.93
24	2000	113.82	31.03	0.5	1.78	357.03	1.69	333.39
25	2000	134.48	44.31	0.5	1.53	425.69	1.36	381.81
26	2000	134.48	69.70	0.5	1.53	452.95	1.27	380.42
27	2000	107.72	74.37	0.5	1.87	381.79	1.59	314.56
28	2000	120.89	53.37	0.5	1.69	404.15	1.48	346.67
29	2000	106.83	31.03	0.5	1.87	340.53	1.80	315.72
30	2000	101.11	31.03	0.5	1.95	326.49	1.89	301.28
	Designed VDMF				Designed VDMF			
	2000	135	135	0.5	1.55	411.07	1.51	385.76

The profile of dampers forces and their strokes for Optimal Solution 2 are shown in Figure 53.

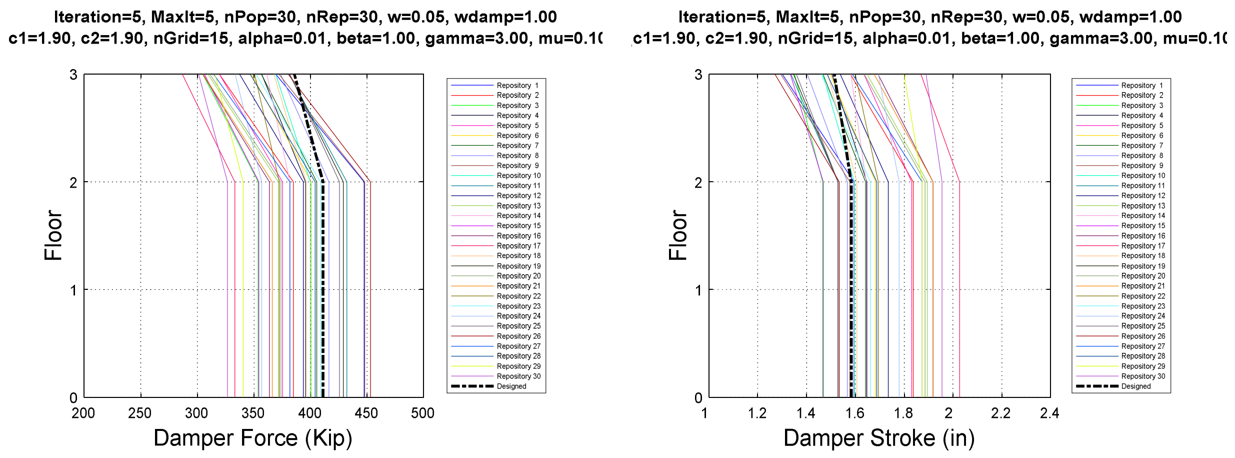


FIGURE 53. Dampers' forces and strokes for MOPSO Optimal Solution 2 and original VDMF design.

The solutions show that the profile of dampers forces and their strokes for Optimal Solution 2 are similar to those for Optimal Solution 1.

CHAPTER 5

SUMMARY AND CONCLUSIONS

This study focuses on optimization of viscous dampers for seismic applications utilizing multi-objective optimization algorithm MOPSO. While Genetic Algorithms can also be used for multi-objective optimization, they typically require larger population size than MOPSO to find the global minimum, requiring longer computation time. MOPSO with its inherently metaheuristic approach and grid-based population selection avoids getting trapped in the local optimal solutions and effectively discovers non-convex solutions.

The thesis provides recommendations on how to adequately use MOPSO to generate reliable solution set. Special emphasis is given to formulation of constraints, set up of MOPSO input parameters, preferred definition of cost function, and efficient parallel computation strategies utilizing high-performing server. The presented study reveals that cost functions that only contain EDPs generate locally optimized solutions, while consideration of EDPs along with dampers' properties improves the solution set. The study proposes two equally viable options for the cost functions if optimum seismic EDPs are of interest:

1. Combination of maximum inter-story drift, maximum floor acceleration, maximum inter-story residual drift, damping coefficients associated with each story, and linear combination of damping coefficients at all stories.
2. Combination of maximum damper stroke associated with each story, maximum floor acceleration, maximum inter-story residual drift, maximum damper force associated with each story, and linear combination of maximum damper forces at all stories.

Furthermore, this research study proposes efficinet optimiziation strategy for finding optimal design solutions in the absence of prior engineering knowladge of suitable damper properties that would meet design requirements. The proposed optimization strategy allows utilization of broad ranges of damping coefficients, the primary damper property that governs the design and therefore the primary decision variable, to efficiently find the optimal solutions (i.e., Pareto front) under the set of design objectives and constraints. When the ranges of damping coefficients are broadly selected, the runtime of the original MOPSO optimization is excessive because of randomly generated populations within the broad bounds of the decision variables.

The efficiency (i.e., speed) of the optimization is greatly improved when using the proposed optimization strategy that defines new narower bounds for descision variables within the original bounds by solving for the new minimum and maximum values of damping coefficients that meet constraints imposed on different EDPs. Additionally, interdepencies of dampers in adjecent stories are established and used to improve the qualidty of the solution. The proposed strutegy is verfied through the comparison of an engineered solution with the MOPSO solutions generated considering the set of constraints utilized in the design.

APPENDICES

APPENDIX A

MAIN FUNCTION AND COST FUNCTION FOR DIFFERENT STRATEGIES

Main Function for All Platforms.

```
% Copyright (c) 2015, Yarpiz (www.yarpiz.com)
% All rights reserved. Please read the "license.txt" for license terms.
% Project Code: YPEA121
% Project Title: Multi-Objective Particle Swarm Optimization (MOPSO)
% Publisher: Yarpiz (www.yarpiz.com)
% Developer: S. Mostapha Kalami Heris (Member of Yarpiz Team)
% Contact Info: sm.kalami@gmail.com, info@yarpiz.com
%
clc;
clear;
close all;
diary MATLAB&OPENSEES.out;
tic
%%
% This program capable to run optimization in following circumstances
% 1- Only OpenSees 2- Parallel Processing in Matlab 3- Parallel
% Processing in OpenSeesMP 4- Parallel Processing in (Matlab and
% OpenSeesMP.
%% Important message for running parallel processing in Matlab
% 1- Our local server has the capability to run the total
% set of 30 workers for parallel processing in MATLAB! For each run, consider
% the minimum workers of "number of population".
%% 2- Important message for running OpenSeesMP
% 1- the system capable to run 20 paired earthquakes. 40 single direction.
% 2- It is recommended to use maximum of 12 paired earthquakes or 24 single
% direction.
%% 3- Important message for running parallel processing in Matlab +
OpenSeesMP
% 1- the system capable to run maximum of 6 populations and assiging 2
% cores for each of the populations in one or two directions
%% Platform
Platform=2;
% 1  Only OpenSees                        any models
% 2  Matlab parfor                        for fast run-time models, with large
initial populations
% 3  OpenSeesMP                           for slow run-time models, with any
initial populations
% 4  Matlab parfor + OpenSeesMP           for small set of populations
%% Problem Definition
nVar=4;             % Number of Decision Variables
VarSize=[1 nVar];   % Size of Decision Variables Matrix

VarMin=[2000.0 120.0 20.0 0.5]; % Lower Bound of Input Variables [axial
stiffness, damping coefficient of first and second floors, damping
coefficient of third floor, velocity exponent]
VarMax=[2000.0 160.0 60.0 0.5]; % Upper Bound of Input Variables [axial
stiffness, damping coefficient of first and second floors, damping
coefficient of third floor, velocity exponent]

% for final figures set up
varmin= [1800 VarMin(2) VarMin(3) 0.3];     % graphs lower boundaries
varmax= [2200 VarMax(2) VarMax(3) 0.6];     % graphs upper boundaries
varspac=[50.0 5.0 5.0 0.05];                % graphs spacing between lower
and upper boundaries
```

```
dmin=0;dmax=2.4;dspac=0.2;                    % graphs set up for drift/ min, 
max, spacing
amin=0;amax=1.4;aspac=0.2;                    % graphs set up for acceleration/ 
min, max, spacing
rmin=0;rmax=0.6;rspac=0.05;                   % graphs set up for residual 
drift/ min, max, spacing

%% OpenSees Parameters
intensity=[2];                       % intensity=[50] or [10] or [2]
eqNumI=1;                            % eqNumI is the starting earthquake number
eqNumJ=1;                            % eqNumJ is the final earthquake number
Dir={'FP'};                         % Dir={'FN','FP'} or Dir={'FN'} or 
Dir={'FP'}

%% Constraint tag
Drift_Constraint=2;             % set drift threshold  (unit Percentage in/in) 
or/ 'None'.          'None' means no drift constraint
Acc_Constraint=0.85;                % set acceleration threshold (unit g) or/ 
'None'.                    'None' means no acceleration constraint (Unit g)
Res_Drift_Constraint=0.5;             % set residual drift threshold (unit 
Percentage in/in) or/ 'None'.  'None' means no residual drift constraint 
(Unit Percentage in/in)

% Methods for adding constraints
Drift_Constraint_method=2;           % "1" Average,  "2" 50th percentile,  "3" 
90th percentile
Acc_Constraint_method=2;
Res_Drift_Constraint_method=2;

%% Results tag for calculating the cost
% Result = 1;                % "1" generates the average of drift and 
acceleration profiles
Result = 2;                  % "2" generates the median (50th percentile) of 
drift and acceleration profiles
% Result = 3;                % "3" generates the 90th percentile of drift and 
acceleration profiles
%% MOPSO Parameters

MaxIt=5;                    % Maximum Number of Iterations
nPop=30;                    % Population Size
nRep=30;                    % Repository Size
w=0.05;                     % Inertia Weight
wdamp=1;                    % Inertia Weight Damping Rate
c1=1.9;                     % Personal Learning Coefficient
c2=1.9;                     % Global Learning Coefficient
nGrid=15;                   % Number of Grids per Dimension
alpha=0.01;                 % Inflation Rate
beta=1;                     % Leader Selection Pressure
gamma=3;                    % Deletion Selection Pressure
mu=0.1;                     % Mutation Rate

%% Initialization
empty_particle.Position=[];
empty_particle.Velocity=[];
empty_particle.Cost=[];
empty_particle.Best.Position=[];
empty_particle.Best.Cost=[];
empty_particle.IsDominated=[];
empty_particle.GridIndex=[];
empty_particle.GridSubIndex=[];
pop=repmat(empty_particle,nPop,1);

NewSol=repmat(empty_particle,nPop,1); % this matrix is assigned to the NewSol
(parfor loop)

if Platform==1
    CostFunction=@cost61;       % Cost Function

    for  i=1:nPop

        pop(i).Position=unifrnd(VarMin,VarMax,VarSize);

        pop(i).Velocity=zeros(VarSize);


pop(i).Cost=CostFunction(pop(i).Position,i,intensity,Dir,eqNumI,eqNumJ,Drift_
Constraint,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,Acc_Co
nstraint_method,Res_Drift_Constraint_method,Result,nPop);

        if  pop(i).Cost(1)<=0.02 % To rerune the uncompleted runs

pop(i).Cost=CostFunction(pop(i).Position,i,intensity,Dir,eqNumI,eqNumJ,Drift_
Constraint,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,Acc_Co
nstraint_method,Res_Drift_Constraint_method,Result,nPop);
        end

        while pop(i).Cost(1)>=10 || pop(i).Cost(2)>=10 || pop(i).Cost(3)>=10
% to rerune when the constraints are not met
            pop(i).Position=unifrnd(VarMin,VarMax,VarSize);

pop(i).Cost=CostFunction(pop(i).Position,i,intensity,Dir,eqNumI,eqNumJ,Drift_
Constraint,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,Acc_Co
nstraint_method,Res_Drift_Constraint_method,Result,nPop);

            if  pop(i).Cost(1)<=0.02

pop(i).Cost=CostFunction(pop(i).Position,i,intensity,Dir,eqNumI,eqNumJ,Drift_
Constraint,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,Acc_Co
nstraint_method,Res_Drift_Constraint_method,Result,nPop);
            end
        end

        % Update Personal Best
        pop(i).Best.Position=pop(i).Position;
        pop(i).Best.Cost=pop(i).Cost;
    end

    % Determine Domination
    pop=DetermineDomination(pop);
    rep=pop(~[pop.IsDominated]);

    Grid=CreateGrid(rep,nGrid,alpha);

    for i=1:numel(rep)
        rep(i)=FindGridIndex(rep(i),Grid);
    end

    %% MOPSO Main Loop
    for it=1:MaxIt
        for i=1:nPop

            leader=SelectLeader(rep,beta);

            pop(i).Velocity = w*pop(i).Velocity ...
                +c1*rand(VarSize).*(pop(i).Best.Position-pop(i).Position) ...
                +c2*rand(VarSize).*(leader.Position-pop(i).Position);

            pop(i).Position = pop(i).Position + pop(i).Velocity;

            pop(i).Position = max(pop(i).Position, VarMin);
            pop(i).Position = min(pop(i).Position, VarMax);

            pop(i).Cost =
CostFunction(pop(i).Position,i,intensity,Dir,eqNumI,eqNumJ,Drift_Constraint,A
cc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,Acc_Constraint_met
hod,Res_Drift_Constraint_method,Result,nPop);

            if   pop(i).Cost(1)<=0.02
                pop(i).Cost =
CostFunction(pop(i).Position,i,intensity,Dir,eqNumI,eqNumJ,Drift_Constraint,A
cc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,Acc_Constraint_met
hod,Res_Drift_Constraint_method,Result,nPop);
            end
            while pop(i).Cost(1)>=10 || pop(i).Cost(2)>=10 ||
pop(i).Cost(3)>=10

                leader=SelectLeader(rep,beta);

                pop(i).Velocity = w*pop(i).Velocity ...
                    +c1*rand(VarSize).*(pop(i).Best.Position-pop(i).Position)
...

                    +c2*rand(VarSize).*(leader.Position-pop(i).Position);

                pop(i).Position = pop(i).Position + pop(i).Velocity;

                pop(i).Position = max(pop(i).Position, VarMin);
                pop(i).Position = min(pop(i).Position, VarMax);

                pop(i).Cost =
CostFunction(pop(i).Position,i,intensity,Dir,eqNumI,eqNumJ,Drift_Constraint,A
cc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,Acc_Constraint_met
hod,Res_Drift_Constraint_method,Result,nPop);
```

```
                if  pop(i).Cost(1)<=0.02
                    pop(i).Cost =
CostFunction(pop(i).Position,i,intensity,Dir,eqNumI,eqNumJ,Drift_Constraint,A
cc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,Acc_Constraint_met
hod,Res_Drift_Constraint_method,Result,nPop);
                end
            end

            % Apply Mutation
            pm=(1-(it-1)/(MaxIt-1))^(1/mu);
            if rand<pm

                NewSol(i).Position=Mutate(pop(i).Position,pm,VarMin,VarMax);

                % this part is added to code!
                NewSol(i).Position = max(NewSol(i).Position, VarMin);
                NewSol(i).Position = min(NewSol(i).Position, VarMax);


NewSol(i).Cost=CostFunction(NewSol(i).Position,i,intensity,Dir,eqNumI,eqNumJ,
Drift_Constraint,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,
Acc_Constraint_method,Res_Drift_Constraint_method,Result,nPop);

                if NewSol(i).Cost(1)<=0.02

NewSol(i).Cost=CostFunction(NewSol(i).Position,i,intensity,Dir,eqNumI,eqNumJ,
Drift_Constraint,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,
Acc_Constraint_method,Res_Drift_Constraint_method,Result,nPop);

                end

                while NewSol(i).Cost(1) >=10 || NewSol(i).Cost(2) >=10 ||
NewSol(i).Cost(3) >=10

NewSol(i).Position=Mutate(pop(i).Position,pm,VarMin,VarMax);
                    NewSol(i).Position= max(NewSol(i).Position, VarMin);
                    NewSol(i).Position= min(NewSol(i).Position, VarMax);


NewSol(i).Cost=CostFunction(NewSol(i).Position,i,intensity,Dir,eqNumI,eqNumJ,
Drift_Constraint,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,
Acc_Constraint_method,Res_Drift_Constraint_method,Result,nPop);
                    if NewSol(i).Cost(1)<=0.02

NewSol(i).Cost=CostFunction(NewSol(i).Position,i,intensity,Dir,eqNumI,eqNumJ,
Drift_Constraint,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,
Acc_Constraint_method,Res_Drift_Constraint_method,Result,nPop);

                    end
                end

                if Dominates(NewSol(i),pop(i))
                    pop(i).Position=NewSol(i).Position;
                    pop(i).Cost=NewSol(i).Cost;
                elseif Dominates(pop(i),NewSol(i))
                    % Do Nothing

                else
                    if rand<0.5
                        pop(i).Position=NewSol(i).Position;
                        pop(i).Cost=NewSol(i).Cost;
                    end
                end
            end

            if Dominates(pop(i),pop(i).Best)
                pop(i).Best.Position=pop(i).Position;
                pop(i).Best.Cost=pop(i).Cost;

            elseif Dominates(pop(i).Best,pop(i))
                % Do Nothing

            else
                if rand<0.5
                    pop(i).Best.Position=pop(i).Position;
                    pop(i).Best.Cost=pop(i).Cost;
                end
            end

        end

        % Add Non-Dominated Particles to REPOSITORY
        rep=[rep
            pop(~[pop.IsDominated])]; %#ok

        % Determine Domination of New Resository Members
        rep=DetermineDomination(rep);

        % Keep only Non-Dminated Memebrs in the Repository
        rep=rep(~[rep.IsDominated]);

        % Update Grid
        Grid=CreateGrid(rep,nGrid,alpha);

        % Update Grid Indices
        for i=1:numel(rep)
            rep(i)=FindGridIndex(rep(i),Grid);
        end

        % Check if Repository is Full
        if numel(rep)>nRep

            Extra=numel(rep)-nRep;
            for e=1:Extra
                rep=DeleteOneRepMemebr(rep,gamma);
            end

        end
```

```
        % Plot Costs
        if Result==1
            Intitle='Average';
        elseif Result==2
            Intitle= '50^t^h Percentile';
        else
            Intitle= '90^t^h Percentile';
        end

        if length(Dir)==1
            dir = sprintf('%s',Dir{:});
            if  dir=='FN'
                dir= 'FN DIR';
            else
                dir= 'FP DIR';
            end
        else
            dir = sprintf(('%s & %s'),Dir{:});
            if  dir== 'FN & FP'
                dir= 'FN & FP DIRs';
            else
                dir= 'FP & FN DIRs';
            end
        end
        tytle=sprintf('%s of [Maximum EDP in All Stories/Floors in %s]\n %d%%
in 50 years, Iteration=%d, MaxIt=%d, nPop=%d, nRep=%d, w=%.2f, wdamp=%.2f\n
c1=%.2f, c2=%.2f, nGrid=%d, alpha=%.2f, beta=%.2f, gamma=%.2f,
mu=%.2f\n',Intitle,dir,intensity,it,MaxIt,nPop,nRep,w,wdamp,c1,c2,nGrid,alpha
,beta,gamma,mu);
        tytle2=sprintf('%s of [Maximum EDP in All Stories/Floors in %s]\n
%d%% in 50 years, MaxIt=%d, nPop=%d, nRep=%d, w=%.2f, wdamp=%.2f\n c1=%.2f,
c2=%.2f, nGrid=%d, alpha=%.2f, beta=%.2f, gamma=%.2f,
mu=%.2f\n',Intitle,dir,intensity,MaxIt,nPop,nRep,w,wdamp,c1,c2,nGrid,alpha,be
ta,gamma,mu);


PlotCosts6(pop,rep,nPop,tytle,tytle2,it,varmin,varmax,varspac,dmin,dmax,dspac
,amin,amax,aspac,rmin,rmax,rspac);

        % Show Iteration Information
        disp(['Iteration ' num2str(it) ': Number of Rep Members = '
num2str(numel(rep))]);

        % Damping Inertia Weight
        w=w*wdamp;
    end
    CalcTime=toc;
    save matlab.mat;
    diary off;
    %% Results
    Directory = mkdir(sprintf('Figures1'));
    FolderName = sprintf('Figures1');
    FigList = findobj(allchild(0), 'flat', 'Type', 'figure');
    FigList =sort(FigList);
    for iFig = 1:MaxIt
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end

    for iFig = 51:50+MaxIt
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end

    for iFig = 101:100+MaxIt
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end

    for iFig = 151:150+MaxIt
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end

    for iFig = 201:200+MaxIt
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end

    for iFig = 251:250+MaxIt
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end

    for iFig = 301:300+MaxIt
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end

    for iFig = 350:350
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end

    for iFig =400:400
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end

    for iFig =450:450
        gca=figure(iFig);
```

```
            savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
            saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
        end

        for iFig =500:500
            gca=figure(iFig);
            savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
            saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
        end

        for iFig =550:550
            gca=figure(iFig);
            savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
            saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
        end

        for iFig =600:600
            gca=figure(iFig);
            savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
            saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
        end

        for iFig =651:600+MaxIt
            gca=figure(iFig);
            savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
            saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
        end

        for iFig =701:700+MaxIt
            gca=figure(iFig);
            savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
            saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
        end

        for iFig =751:750+MaxIt
            gca=figure(iFig);
            savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
            saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
        end
        for iFig =901:900+MaxIt
            gca=figure(iFig);
            savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
            saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
        end
    end

    if Platform==2
        CostFunction=@cost62;        % Cost Function
        mypool= parpool('local',nPop);
        mypool.IdleTimeout=Inf;

        parfor  i=1:nPop
            %     PRT=i
            pop(i).Position=unifrnd(VarMin,VarMax,VarSize);
            pop(i).Velocity=zeros(VarSize);


    pop(i).Cost=CostFunction(pop(i).Position,i,intensity,Dir,eqNumI,eqNumJ,Drift_
    Constraint,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,Acc_Co
    nstraint_method,Res_Drift_Constraint_method,Result,nPop);

            %  while  pop(i).Cost(1)<=0.02 || pop(i).Cost(2)<=0.02
            while  pop(i).Cost(1)<=0.02

    pop(i).Cost=CostFunction(pop(i).Position,i,intensity,Dir,eqNumI,eqNumJ,Drift_
    Constraint,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,Acc_Co
    nstraint_method,Res_Drift_Constraint_method,Result,nPop);
            end

            while pop(i).Cost(1)>=10 || pop(i).Cost(2)>=10 || pop(i).Cost(3)>=10
                pop(i).Position=unifrnd(VarMin,VarMax,VarSize);

    pop(i).Cost=CostFunction(pop(i).Position,i,intensity,Dir,eqNumI,eqNumJ,Drift_
    Constraint,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,Acc_Co
    nstraint_method,Res_Drift_Constraint_method,Result,nPop);

                %              while  pop(i).Cost(1)<=0.02 || pop(i).Cost(2)<=0.02
                while  pop(i).Cost(1)<=0.02

    pop(i).Cost=CostFunction(pop(i).Position,i,intensity,Dir,eqNumI,eqNumJ,Drift_
    Constraint,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,Acc_Co
    nstraint_method,Res_Drift_Constraint_method,Result,nPop);
                end
            end

            % Update Personal Best
            pop(i).Best.Position=pop(i).Position;
            pop(i).Best.Cost=pop(i).Cost;
        end

        % Determine Domination
        pop=DetermineDomination(pop);

        rep=pop(~[pop.IsDominated]);

        Grid=CreateGrid(rep,nGrid,alpha);

        for i=1:numel(rep)
            rep(i)=FindGridIndex(rep(i),Grid);
        end

        %% MOPSO Main Loop
        for it=1:MaxIt
            parfor i=1:nPop
                % PRT2=i
                leader=SelectLeader(rep,beta);

                pop(i).Velocity = w*pop(i).Velocity ...
                    +c1*rand(VarSize).*(pop(i).Best.Position-pop(i).Position) ...
                    +c2*rand(VarSize).*(leader.Position-pop(i).Position);

                pop(i).Position = pop(i).Position + pop(i).Velocity;

                pop(i).Position = max(pop(i).Position, VarMin);
                pop(i).Position = min(pop(i).Position, VarMax);

                pop(i).Cost = 
    CostFunction(pop(i).Position,i,intensity,Dir,eqNumI,eqNumJ,Drift_Constraint,A
    cc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,Acc_Constraint_met
    hod,Res_Drift_Constraint_method,Result,nPop);

                %              while  pop(i).Cost(1)<=0.02 || pop(i).Cost(2)<=0.02
                while  pop(i).Cost(1)<=0.02

                    pop(i).Cost = 
    CostFunction(pop(i).Position,i,intensity,Dir,eqNumI,eqNumJ,Drift_Constraint,A
    cc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,Acc_Constraint_met
    hod,Res_Drift_Constraint_method,Result,nPop);
                end
                while pop(i).Cost(1)>=10 || pop(i).Cost(2)>=10 ||
    pop(i).Cost(3)>=10

                    leader=SelectLeader(rep,beta);

                    pop(i).Velocity = w*pop(i).Velocity ...
                        +c1*rand(VarSize).*(pop(i).Best.Position-pop(i).Position)
    ...
                        +c2*rand(VarSize).*(leader.Position-pop(i).Position);

                    pop(i).Position = pop(i).Position + pop(i).Velocity;

                    pop(i).Position = max(pop(i).Position, VarMin);
                    pop(i).Position = min(pop(i).Position, VarMax);

                    pop(i).Cost = 
    CostFunction(pop(i).Position,i,intensity,Dir,eqNumI,eqNumJ,Drift_Constraint,A
    cc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,Acc_Constraint_met
    hod,Res_Drift_Constraint_method,Result,nPop);

                    %                  while  pop(i).Cost(1)<=0.02 ||
    pop(i).Cost(2)<=0.02
                    while  pop(i).Cost(1)<=0.02

                        pop(i).Cost = 
    CostFunction(pop(i).Position,i,intensity,Dir,eqNumI,eqNumJ,Drift_Constraint,A
    cc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,Acc_Constraint_met
    hod,Res_Drift_Constraint_method,Result,nPop);
                    end
                end

                % Apply Mutation
                pm=(1-(it-1)/(MaxIt-1))^(1/mu);
                if rand<pm
```

```
                NewSol(i).Position=Mutate(pop(i).Position,pm,VarMin,VarMax);

                % this part is added to code!
                NewSol(i).Position = max(NewSol(i).Position, VarMin);
                NewSol(i).Position = min(NewSol(i).Position, VarMax);


NewSol(i).Cost=CostFunction(NewSol(i).Position,i,intensity,Dir,eqNumI,eqNumJ,
Drift_Constraint,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,
Acc_Constraint_method,Res_Drift_Constraint_method,Result,nPop);

                %                  while  NewSol(i).Cost(1)<=0.02 ||
NewSol(i).Cost(2)<=0.02

                while  NewSol(i).Cost(1)<=0.02

NewSol(i).Cost=CostFunction(NewSol(i).Position,i,intensity,Dir,eqNumI,eqNumJ,
Drift_Constraint,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,
Acc_Constraint_method,Res_Drift_Constraint_method,Result,nPop);
                end

                while NewSol(i).Cost(1) >=10 || NewSol(i).Cost(2) >=10 ||
NewSol(i).Cost(3) >=10

NewSol(i).Position=Mutate(pop(i).Position,pm,VarMin,VarMax);
                    NewSol(i).Position= max(NewSol(i).Position, VarMin);
                    NewSol(i).Position= min(NewSol(i).Position, VarMax);


NewSol(i).Cost=CostFunction(NewSol(i).Position,i,intensity,Dir,eqNumI,eqNumJ,
Drift_Constraint,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,
Acc_Constraint_method,Res_Drift_Constraint_method,Result,nPop);
                    %                    while  NewSol(i).Cost(1)<=0.02 ||
NewSol(i).Cost(2)<=0.02

                    while  NewSol(i).Cost(1)<=0.02


NewSol(i).Cost=CostFunction(NewSol(i).Position,i,intensity,Dir,eqNumI,eqNumJ,
Drift_Constraint,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,
Acc_Constraint_method,Res_Drift_Constraint_method,Result,nPop);
                    end
                end

                if Dominates(NewSol(i),pop(i))
                    pop(i).Position=NewSol(i).Position;
                    pop(i).Cost=NewSol(i).Cost;

                elseif Dominates(pop(i),NewSol(i))
                    % Do Nothing

                else
                    if rand<0.5
                pop(i).Position=NewSol(i).Position;
                pop(i).Cost=NewSol(i).Cost;
            end
        end
    end

    if Dominates(pop(i),pop(i).Best)
        pop(i).Best.Position=pop(i).Position;
        pop(i).Best.Cost=pop(i).Cost;

    elseif Dominates(pop(i).Best,pop(i))
        % Do Nothing

    else
        if rand<0.5
            pop(i).Best.Position=pop(i).Position;
            pop(i).Best.Cost=pop(i).Cost;
        end
    end

end

% Add Non-Dominated Particles to REPOSITORY
rep=[rep
    pop(~[pop.IsDominated])]; %#ok

% Determine Domination of New Resository Members
rep=DetermineDomination(rep);

% Keep only Non-Dminated Memebrs in the Repository
rep=rep(~[rep.IsDominated]);

% Update Grid
Grid=CreateGrid(rep,nGrid,alpha);

% Update Grid Indices
for i=1:numel(rep)
    rep(i)=FindGridIndex(rep(i),Grid);
end

% Check if Repository is Full
if numel(rep)>nRep

    Extra=numel(rep)-nRep;
    for e=1:Extra
        rep=DeleteOneRepMemebr(rep,gamma);
    end

end

% Plot Costs
if Result==1
    Intitle='Average';
elseif Result==2
```

```
            Intitle= '50^t^h Percentile';
        else
            Intitle= '90^t^h Percentile';
        end

        if length(Dir)==1
            dir = sprintf('%s',Dir{:});
            if dir=='FN'
                dir= 'FN DIR';
            else
                dir= 'FP DIR';
            end
        else
            dir = sprintf(('%s & %s'),Dir{:});
            if  dir== 'FN & FP'
                dir= 'FN & FP DIRs';
            else
                dir= 'FP & FN DIRs';
            end
        end
        tytle=sprintf('%s of [Maximum EDP in All Stories/Floors in %s]\n %d%%
in 50 years, Iteration=%d, MaxIt=%d, nPop=%d, nRep=%d, w=%.2f, wdamp=%.2f\n
c1=%.2f, c2=%.2f, nGrid=%d, alpha=%.2f, beta=%.2f, gamma=%.2f,
mu=%.2f\n',Intitle,dir,intensity,it,MaxIt,nPop,nRep,w,wdamp,c1,c2,nGrid,alpha
,beta,gamma,mu);
        tytle2=sprintf('%s of [Maximum EDP in All Stories/Floors in %s]\n
%d%% in 50 years, MaxIt=%d, nPop=%d, nRep=%d, w=%.2f, wdamp=%.2f\n c1=%.2f,
c2=%.2f, nGrid=%d, alpha=%.2f, beta=%.2f, gamma=%.2f,
mu=%.2f\n',Intitle,dir,intensity,MaxIt,nPop,nRep,w,wdamp,c1,c2,nGrid,alpha,be
ta,gamma,mu);


PlotCosts6(pop,rep,nPop,tytle,tytle2,it,varmin,varmax,varspac,dmin,dmax,dspac
,amin,amax,aspac,rmin,rmax,rspac);

        % Show Iteration Information
        disp(['Iteration ' num2str(it) ': Number of Rep Members = '
num2str(numel(rep))]);

        % Damping Inertia Weight
        w=w*wdamp;
    end
    CalcTime=toc;
    save matlab.mat;
    delete(mypool);
    diary off;
    %% Results
    Directory = mkdir(sprintf('Figures1'));
    FolderName = sprintf('Figures1');
    FigList = findobj(allchild(0), 'flat', 'Type', 'figure');
    FigList =sort(FigList);
    for iFig = 1:MaxIt
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end
```

```
for iFig = 51:50+MaxIt
    gca=figure(iFig);
    savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
    saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
end

for iFig = 101:100+MaxIt
    gca=figure(iFig);
    savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
    saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
end

for iFig = 151:150+MaxIt
    gca=figure(iFig);
    savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
    saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
end

for iFig = 201:200+MaxIt
    gca=figure(iFig);
    savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
    saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
end

for iFig = 251:250+MaxIt
    gca=figure(iFig);
    savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
    saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
end

for iFig = 301:300+MaxIt
    gca=figure(iFig);
    savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
    saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
end

for iFig = 350:350
    gca=figure(iFig);
    savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
    saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
end

for iFig =400:400
    gca=figure(iFig);
    savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
    saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
end

for iFig =450:450
    gca=figure(iFig);
    savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
    saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
end
```

```
    for iFig =500:500
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end

    for iFig =550:550
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end

    for iFig =600:600
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end

    for iFig =651:600+MaxIt
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end

    for iFig =701:700+MaxIt
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end

    for iFig =751:750+MaxIt
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end
    for iFig =901:900+MaxIt
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end
end

if Platform==3
    CostFunction=@cost63;       % Cost Function
    np=(eqNumJ-eqNumI+1)*numel(Dir)-2;

    for  i=1:nPop
        PRT=i
        pop(i).Position=unifrnd(VarMin,VarMax,VarSize);

        pop(i).Velocity=zeros(VarSize);


pop(i).Cost=CostFunction(pop(i).Position,i,np,intensity,Dir,eqNumI,eqNumJ,Dri
ft_Constraint,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,Acc
_Constraint_method,Res_Drift_Constraint_method,Result,nPop);

        if  pop(i).Cost(1)<=0.02

pop(i).Cost=CostFunction(pop(i).Position,i,np,intensity,Dir,eqNumI,eqNumJ,Dri
ft_Constraint,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,Acc
_Constraint_method,Res_Drift_Constraint_method,Result,nPop);
        end

        while pop(i).Cost(1)>=10 || pop(i).Cost(2)>=10 || pop(i).Cost(3)>=10
            pop(i).Position=unifrnd(VarMin,VarMax,VarSize);

pop(i).Cost=CostFunction(pop(i).Position,i,np,intensity,Dir,eqNumI,eqNumJ,Dri
ft_Constraint,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,Acc
_Constraint_method,Res_Drift_Constraint_method,Result,nPop);

            if  pop(i).Cost(1)<=0.02

pop(i).Cost=CostFunction(pop(i).Position,i,np,intensity,Dir,eqNumI,eqNumJ,Dri
ft_Constraint,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,Acc
_Constraint_method,Res_Drift_Constraint_method,Result,nPop);
            end
        end
        % Update Personal Best
        pop(i).Best.Position=pop(i).Position;
        pop(i).Best.Cost=pop(i).Cost;
    end

    % Determine Domination
    pop=DetermineDomination(pop);

    rep=pop(~[pop.IsDominated]);

    Grid=CreateGrid(rep,nGrid,alpha);

    for i=1:numel(rep)
        rep(i)=FindGridIndex(rep(i),Grid);
    end

    %% MOPSO Main Loop
    for it=1:MaxIt
        for i=1:nPop
            PRT2=i
            leader=SelectLeader(rep,beta);

            pop(i).Velocity = w*pop(i).Velocity ...
                +c1*rand(VarSize).*(pop(i).Best.Position-pop(i).Position) ...
                +c2*rand(VarSize).*(leader.Position-pop(i).Position);

            pop(i).Position = pop(i).Position + pop(i).Velocity;

            pop(i).Position = max(pop(i).Position, VarMin);
            pop(i).Position = min(pop(i).Position, VarMax);
            pop(i).Cost = 
CostFunction(pop(i).Position,i,np,intensity,Dir,eqNumI,eqNumJ,Drift_Constrain
t,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,Acc_Constraint_
method,Res_Drift_Constraint_method,Result,nPop);

            if   pop(i).Cost(1)<=0.02
                pop(i).Cost = 
CostFunction(pop(i).Position,i,np,intensity,Dir,eqNumI,eqNumJ,Drift_Constrain
t,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,Acc_Constraint_
method,Res_Drift_Constraint_method,Result,nPop);
            end
            while pop(i).Cost(1)>=10 || pop(i).Cost(2)>=10 || 
pop(i).Cost(3)>=10

                leader=SelectLeader(rep,beta);

                pop(i).Velocity = w*pop(i).Velocity ...
                    +c1*rand(VarSize).*(pop(i).Best.Position-pop(i).Position)
...
                    +c2*rand(VarSize).*(leader.Position-pop(i).Position);

                pop(i).Position = pop(i).Position + pop(i).Velocity;

                pop(i).Position = max(pop(i).Position, VarMin);
                pop(i).Position = min(pop(i).Position, VarMax);

                pop(i).Cost = 
CostFunction(pop(i).Position,i,np,intensity,Dir,eqNumI,eqNumJ,Drift_Constrain
t,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,Acc_Constraint_
method,Res_Drift_Constraint_method,Result,nPop);

                if  pop(i).Cost(1)<=0.02
                    pop(i).Cost = 
CostFunction(pop(i).Position,i,np,intensity,Dir,eqNumI,eqNumJ,Drift_Constrain
t,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,Acc_Constraint_
method,Res_Drift_Constraint_method,Result,nPop);
                end
            end

            % Apply Mutation
            pm=(1-(it-1)/(MaxIt-1))^(1/mu);
            if rand<pm

                NewSol(i).Position=Mutate(pop(i).Position,pm,VarMin,VarMax);

                % this part is added to code!
                NewSol(i).Position = max(NewSol(i).Position, VarMin);
                NewSol(i).Position = min(NewSol(i).Position, VarMax);


NewSol(i).Cost=CostFunction(NewSol(i).Position,i,np,intensity,Dir,eqNumI,eqNu
mJ,Drift_Constraint,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_meth
od,Acc_Constraint_method,Res_Drift_Constraint_method,Result,nPop);
```

```
                if NewSol(i).Cost(1)<=0.02

NewSol(i).Cost=CostFunction(NewSol(i).Position,i,np,intensity,Dir,eqNumI,eqNu
mJ,Drift_Constraint,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_meth
od,Acc_Constraint_method,Res_Drift_Constraint_method,Result,nPop);

                end

                while NewSol(i).Cost(1) >=10 || NewSol(i).Cost(2) >=10 ||
NewSol(i).Cost(3) >=10

NewSol(i).Position=Mutate(pop(i).Position,pm,VarMin,VarMax);
                    NewSol(i).Position= max(NewSol(i).Position, VarMin);
                    NewSol(i).Position= min(NewSol(i).Position, VarMax);


NewSol(i).Cost=CostFunction(NewSol(i).Position,i,np,intensity,Dir,eqNumI,eqNu
mJ,Drift_Constraint,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_meth
od,Acc_Constraint_method,Res_Drift_Constraint_method,Result,nPop);
                    if NewSol(i).Cost(1)<=0.02

NewSol(i).Cost=CostFunction(NewSol(i).Position,i,np,intensity,Dir,eqNumI,eqNu
mJ,Drift_Constraint,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_meth
od,Acc_Constraint_method,Res_Drift_Constraint_method,Result,nPop);

                    end
                end

                if Dominates(NewSol(i),pop(i))
                    pop(i).Position=NewSol(i).Position;
                    pop(i).Cost=NewSol(i).Cost;

                elseif Dominates(pop(i),NewSol(i))
                    % Do Nothing

                else
                    if rand<0.5
                        pop(i).Position=NewSol(i).Position;
                        pop(i).Cost=NewSol(i).Cost;
                    end
                end
            end

            if Dominates(pop(i),pop(i).Best)
                pop(i).Best.Position=pop(i).Position;
                pop(i).Best.Cost=pop(i).Cost;

            elseif Dominates(pop(i).Best,pop(i))
                % Do Nothing

            else
                if rand<0.5
                    pop(i).Best.Position=pop(i).Position;
                    pop(i).Best.Cost=pop(i).Cost;
```

```
        end
    end

end

% Add Non-Dominated Particles to REPOSITORY
rep=[rep
    pop(~[pop.IsDominated])]; %#ok

% Determine Domination of New Resository Members
rep=DetermineDomination(rep);

% Keep only Non-Dminated Memebrs in the Repository
rep=rep(~[rep.IsDominated]);

% Update Grid
Grid=CreateGrid(rep,nGrid,alpha);

% Update Grid Indices
for i=1:numel(rep)
    rep(i)=FindGridIndex(rep(i),Grid);
end

% Check if Repository is Full
if numel(rep)>nRep

    Extra=numel(rep)-nRep;
    for e=1:Extra
        rep=DeleteOneRepMemebr(rep,gamma);
    end

end

% Plot Costs
if Result==1
    Intitle='Average';
elseif Result==2
    Intitle= '50^t^h Percentile';
else
    Intitle= '90^t^h Percentile';
end

if length(Dir)==1
    dir = sprintf('%s',Dir{:});
    if dir=='FN'
        dir= 'FN DIR';
    else
        dir= 'FP DIR';
    end
else
    dir = sprintf(('%s & %s'),Dir{:});
    if  dir== 'FN & FP'
        dir= 'FN & FP DIRs';
    else
        dir= 'FP & FN DIRs';
```

```
            end
        end
        tytle=sprintf('%s of [Maximum EDP in All Stories/Floors in %s]\n %d%%
in 50 years, Iteration=%d, MaxIt=%d, nPop=%d, nRep=%d, w=%.2f, wdamp=%.2f\n
c1=%.2f, c2=%.2f, nGrid=%d, alpha=%.2f, beta=%.2f, gamma=%.2f,
mu=%.2f\n',Intitle,dir,intensity,it,MaxIt,nPop,nRep,w,wdamp,c1,c2,nGrid,alpha
,beta,gamma,mu);
        tytle2=sprintf('%s of [Maximum EDP in All Stories/Floors in %s]\n
%d%% in 50 years, MaxIt=%d, nPop=%d, nRep=%d, w=%.2f, wdamp=%.2f\n c1=%.2f,
c2=%.2f, nGrid=%d, alpha=%.2f, beta=%.2f, gamma=%.2f,
mu=%.2f\n',Intitle,dir,intensity,MaxIt,nPop,nRep,w,wdamp,c1,c2,nGrid,alpha,be
ta,gamma,mu);


PlotCosts6(pop,rep,nPop,tytle,tytle2,it,varmin,varmax,varspac,dmin,dmax,dspac
,amin,amax,aspac,rmin,rmax,rspac);

        % Show Iteration Information
        disp(['Iteration ' num2str(it) ': Number of Rep Members = '
num2str(numel(rep))]);

        % Damping Inertia Weight
        w=w*wdamp;
    end
    CalcTime=toc;
    save matlab.mat;
    diary off;
    %% Results
    Directory = mkdir(sprintf('Figures1'));
    FolderName = sprintf('Figures1');
    FigList = findobj(allchild(0), 'flat', 'Type', 'figure');
    FigList =sort(FigList);
    for iFig = 1:MaxIt
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end

    for iFig = 51:50+MaxIt
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end

    for iFig = 101:100+MaxIt
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end

    for iFig = 151:150+MaxIt
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end
```

```
for iFig = 201:200+MaxIt
    gca=figure(iFig);
    savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
    saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
end

for iFig = 251:250+MaxIt
    gca=figure(iFig);
    savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
    saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
end

for iFig = 301:300+MaxIt
    gca=figure(iFig);
    savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
    saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
end

for iFig = 350:350
    gca=figure(iFig);
    savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
    saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
end

for iFig =400:400
    gca=figure(iFig);
    savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
    saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
end

for iFig =450:450
    gca=figure(iFig);
    savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
    saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
end

for iFig =500:500
    gca=figure(iFig);
    savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
    saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
end

for iFig =550:550
    gca=figure(iFig);
    savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
    saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
end

for iFig =600:600
    gca=figure(iFig);
    savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
    saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
end
```

```
    for iFig =651:600+MaxIt
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end

    for iFig =701:700+MaxIt
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end

    for iFig =751:750+MaxIt
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end
    for iFig =901:900+MaxIt
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end
end

if Platform==4
    CostFunction=@cost64;       % Cost Function
    mypool= parpool('local',nPop);
    mypool.IdleTimeout=Inf;
    if eqNumJ-eqNumI==0
        np=1;
    else
        np=2;
    end

    parfor  i=1:nPop
        PRT=i
        pop(i).Position=unifrnd(VarMin,VarMax,VarSize);

        pop(i).Velocity=zeros(VarSize);


pop(i).Cost=CostFunction(pop(i).Position,i,np,intensity,Dir,eqNumI,eqNumJ,Dri
ft_Constraint,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,Acc
_Constraint_method,Res_Drift_Constraint_method,Result,nPop);

        if  pop(i).Cost(1)<=0.02

pop(i).Cost=CostFunction(pop(i).Position,i,np,intensity,Dir,eqNumI,eqNumJ,Dri
ft_Constraint,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,Acc
_Constraint_method,Res_Drift_Constraint_method,Result,nPop);
        end

        while pop(i).Cost(1)>=10 || pop(i).Cost(2)>=10 || pop(i).Cost(3)>=10
            pop(i).Position=unifrnd(VarMin,VarMax,VarSize);
pop(i).Cost=CostFunction(pop(i).Position,i,np,intensity,Dir,eqNumI,eqNumJ,Dri
ft_Constraint,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,Acc
_Constraint_method,Res_Drift_Constraint_method,Result,nPop);

            if  pop(i).Cost(1)<=0.02

pop(i).Cost=CostFunction(pop(i).Position,i,np,intensity,Dir,eqNumI,eqNumJ,Dri
ft_Constraint,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,Acc
_Constraint_method,Res_Drift_Constraint_method,Result,nPop);
            end
        end
        % Update Personal Best
        pop(i).Best.Position=pop(i).Position;
        pop(i).Best.Cost=pop(i).Cost;
    end

    % Determine Domination
    pop=DetermineDomination(pop);

    rep=pop(~[pop.IsDominated]);

    Grid=CreateGrid(rep,nGrid,alpha);

    for i=1:numel(rep)
        rep(i)=FindGridIndex(rep(i),Grid);
    end

    %% MOPSO Main Loop
    for it=1:MaxIt
        parfor i=1:nPop
            PRT2=i
            leader=SelectLeader(rep,beta);

            pop(i).Velocity = w*pop(i).Velocity ...
                +c1*rand(VarSize).*(pop(i).Best.Position-pop(i).Position) ...
                +c2*rand(VarSize).*(leader.Position-pop(i).Position);

            pop(i).Position = pop(i).Position + pop(i).Velocity;

            pop(i).Position = max(pop(i).Position, VarMin);
            pop(i).Position = min(pop(i).Position, VarMax);

            pop(i).Cost =
CostFunction(pop(i).Position,i,np,intensity,Dir,eqNumI,eqNumJ,Drift_Constrain
t,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,Acc_Constraint_
method,Res_Drift_Constraint_method,Result,nPop);

            if   pop(i).Cost(1)<=0.02
                pop(i).Cost =
CostFunction(pop(i).Position,i,np,intensity,Dir,eqNumI,eqNumJ,Drift_Constrain
t,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,Acc_Constraint_
method,Res_Drift_Constraint_method,Result,nPop);
            end
            while pop(i).Cost(1)>=10 || pop(i).Cost(2)>=10 ||
pop(i).Cost(3)>=10

                leader=SelectLeader(rep,beta);

                pop(i).Velocity = w*pop(i).Velocity ...
                    +c1*rand(VarSize).*(pop(i).Best.Position-pop(i).Position)
...
                    +c2*rand(VarSize).*(leader.Position-pop(i).Position);

                pop(i).Position = pop(i).Position + pop(i).Velocity;

                pop(i).Position = max(pop(i).Position, VarMin);
                pop(i).Position = min(pop(i).Position, VarMax);

                pop(i).Cost =
CostFunction(pop(i).Position,i,np,intensity,Dir,eqNumI,eqNumJ,Drift_Constrain
t,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,Acc_Constraint_
method,Res_Drift_Constraint_method,Result,nPop);

                if  pop(i).Cost(1)<=0.02
                    pop(i).Cost =
CostFunction(pop(i).Position,i,np,intensity,Dir,eqNumI,eqNumJ,Drift_Constrain
t,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,Acc_Constraint_
method,Res_Drift_Constraint_method,Result,nPop);
                end
            end

            % Apply Mutation
            pm=(1-(it-1)/(MaxIt-1))^(1/mu);
            if rand<pm

                NewSol(i).Position=Mutate(pop(i).Position,pm,VarMin,VarMax);

                % this part is added to code!
                NewSol(i).Position = max(NewSol(i).Position, VarMin);
                NewSol(i).Position = min(NewSol(i).Position, VarMax);


NewSol(i).Cost=CostFunction(NewSol(i).Position,i,np,intensity,Dir,eqNumI,eqNu
mJ,Drift_Constraint,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_meth
od,Acc_Constraint_method,Res_Drift_Constraint_method,Result,nPop);

                if NewSol(i).Cost(1)<=0.02

NewSol(i).Cost=CostFunction(NewSol(i).Position,i,np,intensity,Dir,eqNumI,eqNu
mJ,Drift_Constraint,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_meth
od,Acc_Constraint_method,Res_Drift_Constraint_method,Result,nPop);

                end

                while NewSol(i).Cost(1) >=10 || NewSol(i).Cost(2) >=10 ||
NewSol(i).Cost(3) >=10
NewSol(i).Position=Mutate(pop(i).Position,pm,VarMin,VarMax);
                    NewSol(i).Position= max(NewSol(i).Position, VarMin);
                    NewSol(i).Position= min(NewSol(i).Position, VarMax);


NewSol(i).Cost=CostFunction(NewSol(i).Position,i,np,intensity,Dir,eqNumI,eqNu
mJ,Drift_Constraint,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_meth
od,Acc_Constraint_method,Res_Drift_Constraint_method,Result,nPop);
                    if NewSol(i).Cost(1)<=0.02

NewSol(i).Cost=CostFunction(NewSol(i).Position,i,np,intensity,Dir,eqNumI,eqNu
mJ,Drift_Constraint,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_meth
od,Acc_Constraint_method,Res_Drift_Constraint_method,Result,nPop);

                    end
                end

                if Dominates(NewSol(i),pop(i))
                    pop(i).Position=NewSol(i).Position;
                    pop(i).Cost=NewSol(i).Cost;

                elseif Dominates(pop(i),NewSol(i))
                    % Do Nothing

                else
                    if rand<0.5
                        pop(i).Position=NewSol(i).Position;
                        pop(i).Cost=NewSol(i).Cost;
                    end
                end
            end

            if Dominates(pop(i),pop(i).Best)
                pop(i).Best.Position=pop(i).Position;
                pop(i).Best.Cost=pop(i).Cost;

            elseif Dominates(pop(i).Best,pop(i))
                % Do Nothing

            else
                if rand<0.5
                    pop(i).Best.Position=pop(i).Position;
                    pop(i).Best.Cost=pop(i).Cost;
                end
            end

        end

        % Add Non-Dominated Particles to REPOSITORY
        rep=[rep
            pop(~[pop.IsDominated])]; %#ok

        % Determine Domination of New Resository Members
        rep=DetermineDomination(rep);
        % Keep only Non-Dminated Memebrs in the Repository
        rep=rep(~[rep.IsDominated]);

        % Update Grid
        Grid=CreateGrid(rep,nGrid,alpha);

        % Update Grid Indices
        for i=1:numel(rep)
            rep(i)=FindGridIndex(rep(i),Grid);
        end

        % Check if Repository is Full
        if numel(rep)>nRep

            Extra=numel(rep)-nRep;
            for e=1:Extra
                rep=DeleteOneRepMemebr(rep,gamma);
            end

        end

        % Plot Costs
        if Result==1
            Intitle='Average';
        elseif Result==2
            Intitle= '50^t^h Percentile';
        else
            Intitle= '90^t^h Percentile';
        end

        if length(Dir)==1
            dir = sprintf('%s',Dir{:});
            if dir=='FN'
                dir= 'FN DIR';
            else
                dir= 'FP DIR';
            end
        else
            dir = sprintf(('%s & %s'),Dir{:});
            if  dir== 'FN & FP'
                dir= 'FN & FP DIRs';
            else
                dir= 'FP & FN DIRs';
            end
        end
        tytle=sprintf('%s of [Maximum EDP in All Stories/Floors in %s]\n %d%%
in 50 years, Iteration=%d, MaxIt=%d, nPop=%d, nRep=%d, w=%.2f, wdamp=%.2f\n
c1=%.2f, c2=%.2f, nGrid=%d, alpha=%.2f, beta=%.2f, gamma=%.2f,
mu=%.2f\n',Intitle,dir,intensity,it,MaxIt,nPop,nRep,w,wdamp,c1,c2,nGrid,alpha
,beta,gamma,mu);
        tytle2=sprintf('%s of [Maximum EDP in All Stories/Floors in %s]\n
%d%% in 50 years, MaxIt=%d, nPop=%d, nRep=%d, w=%.2f, wdamp=%.2f\n c1=%.2f,
c2=%.2f, nGrid=%d, alpha=%.2f, beta=%.2f, gamma=%.2f,
mu=%.2f\n',Intitle,dir,intensity,MaxIt,nPop,nRep,w,wdamp,c1,c2,nGrid,alpha,be
ta,gamma,mu);


PlotCosts6(pop,rep,nPop,tytle,tytle2,it,varmin,varmax,varspac,dmin,dmax,dspac
,amin,amax,aspac,rmin,rmax,rspac);

        % Show Iteration Information
        disp(['Iteration ' num2str(it) ': Number of Rep Members = '
num2str(numel(rep))]);

        % Damping Inertia Weight
        w=w*wdamp;
    end
    CalcTime=toc;
    save matlab.mat;
    delete(mypool);
    diary off;
    %% Results
    Directory = mkdir(sprintf('Figures1'));
    FolderName = sprintf('Figures1');
    FigList = findobj(allchild(0), 'flat', 'Type', 'figure');
    FigList =sort(FigList);
    for iFig = 1:MaxIt
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end

    for iFig = 51:50+MaxIt
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end

    for iFig = 101:100+MaxIt
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end

    for iFig = 151:150+MaxIt
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end

    for iFig = 201:200+MaxIt
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end

    for iFig = 251:250+MaxIt
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end

    for iFig = 301:300+MaxIt
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end

    for iFig = 350:350
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end

    for iFig =400:400
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end

    for iFig =450:450
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end

    for iFig =500:500
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end

    for iFig =550:550
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end

    for iFig =600:600
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end

    for iFig =651:600+MaxIt
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end

    for iFig =701:700+MaxIt
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
```

```
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end

    for iFig =751:750+MaxIt
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end
    for iFig =901:900+MaxIt
        gca=figure(iFig);
        savefig(gca,sprintf('%s/%i.fig',FolderName,iFig));
        saveas(gca,sprintf('%s/%i.emf',FolderName,iFig));
    end
end
```

Cost Function, Platform I

```
function
y=cost61(x,PopNum,intensity,Dir,eqNumI,eqNumJ,Drift_Constraint,Acc_Constraint
,Res_Drift_Constraint,Drift_Constraint_method,Acc_Constraint_method,Res_Drift
_Constraint_method,Result,N)
% Inputs
K = x(1);
C12 = x(2);
C3 = x(3);
a = x(4);
% Opening .tcl file, Reading & Saving in A{}
fid=fopen('VDMF.Original1.tcl','rt');
i = 1;
tline = fgetl(fid);
A{i} = tline;
while ischar(tline)
    i = i+1;
    tline = fgetl(fid);
    A{i} = tline;
end
fclose(fid);
% Editing Cells in A{}
text1 = 'set Pop';
text2 = 'set K';
text3 = 'set C12';
text4 = 'set C3';
text5 = 'set a';
text6 = 'set HazardList';
text7 = 'set eqDirList';
text8 = 'set eqNumI';
text9 = 'set eqNumJ';

line1= sprintf('%s {%0.0f}',text1,PopNum);
line2= sprintf('%s %0.2f',text2,K);
line3 = sprintf('%s %0.2f',text3,C12);
line4 = sprintf('%s %0.2f',text4,C3);
line5 = sprintf('%s %0.2f',text5,a);

if length(intensity)==1
    line6 = sprintf('%s {%0.0f}',text6,intensity(:));
elseif length(intensity)==2
    line6 = sprintf('%s {%0.0f %0.0f}',text6,intensity(:));
else
    line6 = sprintf('%s {%0.0f %0.0f %0.0f}',text6,intensity(:));
end
if length(Dir)==1
    line7 = sprintf('%s {"%s"}',text7,Dir{:});
else
    line7 = sprintf('%s {"%s" "%s"}',text7,Dir{:});
end
line8 = sprintf('%s %0.0f',text8,eqNumI);
line9 = sprintf('%s %0.0f',text9,eqNumJ);

A{1} = line1;
A{2} = line2;
```

```
A{3} = line3;
A{4} = line4;
A{5} = line5;
A{6} = line6;
A{7} = line7;
A{8} = line8;
A{9} = line9;

% Printing A{} in Text File
fid0=sprintf('VDMFRun%d.tcl',PopNum);
fid = fopen(fid0, 'wt');
for i = 1:numel(A)
    if A{i+1} == -1
        fprintf(fid,'%s', A{i});
        break
    else
        fprintf(fid,'%s \n', A{i});
    end
end
fclose(fid);
% Executing Opensees operations & Collecting Results
% Write In File
fid = fopen(fid0, 'wt');
for i = 1:numel(A)
    if A{i+1} == -1
        fprintf(fid,'%s', A{i});
        break
    else
        fprintf(fid,'%s\n', A{i});
    end
end
fclose(fid);

% calling OpenSees
% pause(30)
system(sprintf('OpenSees.exe %s',fid0));
% % % system(sprintf('mpiexec -np %d OpenSeesMP.exe %s',np, fid0));
% pause(30)
OpenSeesRun=sprintf('%s is done!',fid0)
% pause(30)
Counter=(eqNumJ-eqNumI+1)*length(Dir);
fileID=cell(1,Counter);
fileID2=cell(1,Counter);
SaveMat=zeros(eqNumJ-eqNumI+1,length(Dir),N);
for eq=eqNumI:eqNumJ
    for dir=1:length(Dir)
        for Name=1:length(fileID)

fileID{Name}=sprintf('Final1_%d_%d_%d_%s.out',intensity,PopNum,eq,Dir{dir});

fileID2{Name}=sprintf('Final2_%d_%d_%d_%s.out',intensity,PopNum,eq,Dir{dir});
            while exist(fileID{Name})==0 || exist(fileID2{Name})==0
                % Editing Cells in A{}
                text8 = 'set eqNumI';
                text9 = 'set eqNumJ';
                line8 = sprintf('%s %0.0f',text8,eq);
                line9 = sprintf('%s %0.0f',text9,eq);
                A{8} = line8;
                A{9} = line9;
                % Printing A{} in Text File
                fid0=sprintf('VDMFRun%d.tcl',PopNum);
                fid = fopen(fid0, 'wt');
                for i = 1:numel(A)
                    if A{i+1} == -1
                        fprintf(fid,'%s', A{i});
                        break
                    else
                        fprintf(fid,'%s \n', A{i});
                    end
                end
                fclose(fid);
                % Executing Opensees operations & Collecting Results
                % Write In File
                fid = fopen(fid0, 'wt');
                for i = 1:numel(A)
                    if A{i+1} == -1
                        fprintf(fid,'%s', A{i});
                        break
                    else
                        fprintf(fid,'%s\n', A{i});
                    end
                end
                fclose(fid);
                OpenSeesRun='earthquake is missed here (loop)!'
                system(sprintf('OpenSees.exe %s',fid0));
            end
            raw=load(fileID{Name});
            raw1=load(fileID2{Name});
            if (raw(2)- raw(1))<=0.1 && (raw1(2)- raw1(1))<=0.1
                SaveMat(eq,dir,PopNum)=1;
            else
                SaveMat(eq,dir,PopNum)=0;
            end
        end
    end
end

% re-change file to its original status
text8 = 'set eqNumI';
text9 = 'set eqNumJ';
line8 = sprintf('%s %0.0f',text8,eqNumI);
line9 = sprintf('%s %0.0f',text9,eqNumJ);
A{8} = line8;
A{9} = line9;
% Printing A{} in Text File
fid0=sprintf('VDMFRun%d.tcl',PopNum);
fid = fopen(fid0, 'wt');
for i = 1:numel(A)
    if A{i+1} == -1
        fprintf(fid,'%s', A{i});
        break
    else
        fprintf(fid,'%s \n', A{i});
    end
end
fclose(fid);
% Write In File
fid = fopen(fid0, 'wt');
for i = 1:numel(A)
    if A{i+1} == -1
        fprintf(fid,'%s', A{i});
        break
    else
        fprintf(fid,'%s\n', A{i});
    end
end
fclose(fid);
%
EQKeepMat=[];
for eq=eqNumI:eqNumJ
    if  mean(SaveMat(eq,:,PopNum))==1
        EQKeepMat=[EQKeepMat eq];
    end
end
EQKeepMat=unique(EQKeepMat);
% if numel(EQKeepMat) <= (eqNumJ-eqNumI)*0.8
if numel(EQKeepMat) < (eqNumJ-eqNumI)+1
    EQKeepMat=[];
end
save(['EQKeepMat_' num2str(PopNum) '.mat'],'EQKeepMat');

% abs of drift and acceleration of the floors
Unit_Conversion_Drift=0.01;
Unit_Conversion_Acc=386.4;
Data_Drift1=[];
Data_Drift2=[];
Data_Acc1=[];
Data_Acc2=[];
Data_Res_Drift1=[];
Data_Res_Drift2=[];
MaxofMaxAbsDrifts=zeros(numel(EQKeepMat),3);
MaxofMaxAbsAccs=zeros(numel(EQKeepMat),3);
MaxofMaxAbsResDrifts=zeros(numel(EQKeepMat),3);
AvgofMaxofMaxAbsDriftstot=zeros(length(intensity),3);
AvgofMaxofMaxAbsAccstot=zeros(length(intensity),3);
AvgofMaxofMaxAbsResDriftstot=zeros(length(intensity),3);
MedianofMaxofMaxAbsDrifts=zeros(length(intensity),3);
MedianofMaxofMaxAbsAccs=zeros(length(intensity),3);
MedianofMaxofMaxAbsResDrifts=zeros(length(intensity),3);
NinetiethofMaxofMaxAbsDrifts=zeros(length(intensity),3);
NinetiethofMaxofMaxAbsAccs=zeros(length(intensity),3);
NinetiethofMaxofMaxAbsResDrifts=zeros(length(intensity),3);

for ii=1:length(intensity)
    if isempty(EQKeepMat)==1
        jj=1;
        MaxofMaxAbsDrifts(jj,:)=10+unifrnd(0,1);
        MaxofMaxAbsAccs(jj,:)=10+unifrnd(0,1);
        MaxofMaxAbsResDrifts(jj,:)=10+unifrnd(0,1);
    end
    if isempty(EQKeepMat)~=1
        for jj=1:numel(EQKeepMat)
            for kk=1:length(Dir)
                if kk==1
                    filename1=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/Drift-
Story1_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj),Dir{kk});
                    filename2=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/Drift-
Story2_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj),Dir{kk});
                    filename3=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/Drift-
Story3_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj),Dir{kk});

                    story1_res=load(filename1);
                    story2_res=load(filename2);
                    story3_res=load(filename3);
                    A=size(story1_res,1);
                    B=size(story2_res,1);
                    C=size(story3_res,1);
                    D=min([A B C]);
                    Data_Drift1=[story1_res(1:D) story2_res(1:D)
story3_res(1:D)]./Unit_Conversion_Drift;
                    Data_Res_Drift1=[abs(story1_res(end))
abs(story2_res(end)) abs(story3_res(end))]./Unit_Conversion_Drift;

                    filename7=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/FloorAccEnv_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj
),Dir{kk});
                    Data_Acc1=[load(filename7)];
                    Data_Acc1=Data_Acc1(end,2:4)./Unit_Conversion_Acc;
                end

                if kk==2
                    filename4=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/Drift-
Story1_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj),Dir{kk});
                    filename5=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/Drift-
Story2_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj),Dir{kk});
                    filename6=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/Drift-
Story3_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj),Dir{kk});

                    story11_res=load(filename4);
                    story22_res=load(filename5);
                    story33_res=load(filename6);

                    AA=size(story11_res,1);
                    BB=size(story22_res,1);
                    CC=size(story33_res,1);
                    DD=min([AA BB CC]);

                    Data_Drift2=[story11_res(1:DD) story22_res(1:DD)
story33_res(1:DD)]./Unit_Conversion_Drift;
```

```
                    Data_Res_Drift2=[abs(story11_res(end))
abs(story22_res(end)) abs(story33_res(end))]./Unit_Conversion_Drift;

                    filename8=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/FloorAccEnv_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj
),Dir{kk});
                    Data_Acc2=[load(filename8)];
                    Data_Acc2=Data_Acc2(end,2:4)./Unit_Conversion_Acc;
                end
            end

            if  length(Dir)==1
                MaxDrifts=[max(abs(Data_Drift1))];
                MaxAccs=[(abs(Data_Acc1))];
                MaxResDrifts=[(abs(Data_Res_Drift1))];
            elseif length(Dir)==2
                MaxDrifts=max([max(abs(Data_Drift1));max(abs(Data_Drift2))]);
                MaxAccs=max([(abs(Data_Acc1));(abs(Data_Acc2))]);

MaxResDrifts=max([(abs(Data_Res_Drift1));(abs(Data_Res_Drift2))]);
            end
            MaxofMaxAbsDrifts(jj,:)=MaxDrifts;
            MaxofMaxAbsAccs(jj,:)=MaxAccs;
            MaxofMaxAbsResDrifts(jj,:)=MaxResDrifts;
        end
    end
    if Result==1
        AvgofMaxofMaxAbsDriftstot(ii,:)=mean(MaxofMaxAbsDrifts);
        AvgofMaxofMaxAbsAccstot(ii,:)=mean(MaxofMaxAbsAccs);
        AvgofMaxofMaxAbsResDriftstot(ii,:)=mean(MaxofMaxAbsResDrifts);
    elseif Result==2

MedianofMaxofMaxAbsDrifts(ii,:)=[prctile_one(MaxofMaxAbsDrifts(:,1),0.5)
prctile_one(MaxofMaxAbsDrifts(:,2),0.5)
prctile_one(MaxofMaxAbsDrifts(:,3),0.5)];
        MedianofMaxofMaxAbsAccs(ii,:)=[prctile_one(MaxofMaxAbsAccs(:,1),0.5)
prctile_one(MaxofMaxAbsAccs(:,2),0.5) prctile_one(MaxofMaxAbsAccs(:,3),0.5)];

MedianofMaxofMaxAbsResDrifts(ii,:)=[prctile_one(MaxofMaxAbsResDrifts(:,1),0.5
) prctile_one(MaxofMaxAbsResDrifts(:,2),0.5)
prctile_one(MaxofMaxAbsResDrifts(:,3),0.5)];
    elseif Result==3

NinetiethofMaxofMaxAbsDrifts(ii,:)=[prctile_one(MaxofMaxAbsDrifts(:,1),1)
prctile_one(MaxofMaxAbsDrifts(:,1),1) prctile_one(MaxofMaxAbsDrifts(:,3),1)];
        NinetiethofMaxofMaxAbsAccs(ii,:)=[prctile_one(MaxofMaxAbsAccs(:,1),1)
prctile_one(MaxofMaxAbsAccs(:,2),1) prctile_one(MaxofMaxAbsAccs(:,3),1)];

NinetiethofMaxofMaxAbsResDrifts(ii,:)=[prctile_one(MaxofMaxAbsResDrifts(:,1),
1) prctile_one(MaxofMaxAbsResDrifts(:,2),1)
prctile_one(MaxofMaxAbsResDrifts(:,3),1)];
    end
end
%%
y11=AvgofMaxofMaxAbsDriftstot;
y12=MedianofMaxofMaxAbsDrifts;
```

```
y13=NinetiethofMaxofMaxAbsDrifts;

y21=AvgofMaxofMaxAbsAccstot;
y22=MedianofMaxofMaxAbsAccs;
y23=NinetiethofMaxofMaxAbsAccs;

y31=AvgofMaxofMaxAbsResDriftstot;
y32=MedianofMaxofMaxAbsResDrifts;
y33=NinetiethofMaxofMaxAbsResDrifts;

if Drift_Constraint=='None'
    Drift_Constraint=100;
end
if Acc_Constraint=='None'
    Acc_Constraint=100;
end
if Res_Drift_Constraint=='None'
    Res_Drift_Constraint=100;
end
Stories=3;
if Result==1
    y1=y11;
    y2=y21;
    y3=y31;
    if Drift_Constraint_method==1 || Acc_Constraint_method==1 ||
Res_Drift_Constraint_method==1
        for i=1:Stories
            if y11(1,i) > Drift_Constraint || y21(1,i) > Acc_Constraint ||
y31(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end

    if Drift_Constraint_method==2 || Acc_Constraint_method==2 ||
Res_Drift_Constraint_method==2
        for i=1:Stories
            if  y12(1,i) > Drift_Constraint || y22(1,i) > Acc_Constraint ||
y32(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end

    if Drift_Constraint_method==3 || Acc_Constraint_method==3 ||
Res_Drift_Constraint_method==3
        for i=1:Stories
            if  y13(1,i) > Drift_Constraint || y23(1,i) > Acc_Constraint ||
y33(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end
```

```
end

if Result==2
    y1=y12;
    y2=y22;
    y3=y32;
    if Drift_Constraint_method==1 || Acc_Constraint_method==1 ||
Res_Drift_Constraint_method==1
        for i=1:Stories
            if  y11(1,i) > Drift_Constraint || y21(1,i) > Acc_Constraint ||
y31(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end
    if Drift_Constraint_method==2 || Acc_Constraint_method==2 ||
Res_Drift_Constraint_method==2
        for i=1:Stories
            if  y12(1,i) > Drift_Constraint || y22(1,i) > Acc_Constraint ||
y32(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end

    if Drift_Constraint_method==3 || Acc_Constraint_method==3 ||
Res_Drift_Constraint_method==3
        for i=1:Stories
            if  y13(1,i) > Drift_Constraint || y23(1,i) > Acc_Constraint ||
y33(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end
end

if Result==3
    y1=y13;
    y2=y23;
    y3=y33;
    if Drift_Constraint_method==1 || Acc_Constraint_method==1 ||
Res_Drift_Constraint_method==1
        for i=1:Stories
            if  y11(1,i) > Drift_Constraint || y21(1,i) > Acc_Constraint ||
y31(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end
    if Drift_Constraint_method==2 || Acc_Constraint_method==2 ||
Res_Drift_Constraint_method==2
        for i=1:Stories
```

```
            if  y12(1,i) > Drift_Constraint || y22(1,i) > Acc_Constraint ||
y32(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end

    if Drift_Constraint_method==3 || Acc_Constraint_method==3 ||
Res_Drift_Constraint_method==3
        for i=1:Stories
            if  y13(1,i) > Drift_Constraint || y23(1,i) > Acc_Constraint ||
y33(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end
end
y=[y1;y2;y3;C12;C3;C12+C3];
```

Cost Function, Platform II

```
function
y=cost62(x,PopNum,intensity,Dir,eqNumI,eqNumJ,Drift_Constraint,Acc_Constraint
,Res_Drift_Constraint,Drift_Constraint_method,Acc_Constraint_method,Res_Drift
_Constraint_method,Result,N)
% Inputs
K = x(1);
C12 = x(2);
C3 = x(3);
a = x(4);
% Opening .tcl file, Reading & Saving in A{}
fid=fopen('VDMF.Original2.tcl','rt');
i = 1;
tline = fgetl(fid);
A{i} = tline;
while ischar(tline)
    i = i+1;
    tline = fgetl(fid);
    A{i} = tline;
end
fclose(fid);
% Editing Cells in A{}
text1 = 'set Pop';
text2 = 'set K';
text3 = 'set C12';
text4 = 'set C3';
text5 = 'set a';
text6 = 'set HazardList';
text7 = 'set eqDirList';
text8 = 'set eqNumI';
text9 = 'set eqNumJ';

line1= sprintf('%s {%0.0f}',text1,PopNum);
line2= sprintf('%s %0.2f',text2,K);
line3 = sprintf('%s %0.2f',text3,C12);
line4 = sprintf('%s %0.2f',text4,C3);
line5 = sprintf('%s %0.2f',text5,a);

if length(intensity)==1
    line6 = sprintf('%s {%0.0f}',text6,intensity(:));
elseif length(intensity)==2
    line6 = sprintf('%s {%0.0f %0.0f}',text6,intensity(:));
else
    line6 = sprintf('%s {%0.0f %0.0f %0.0f}',text6,intensity(:));
end
if length(Dir)==1
    line7 = sprintf('%s {"%s"}',text7,Dir{:});
else
    line7 = sprintf('%s {"%s" "%s"}',text7,Dir{:});
end
line8 = sprintf('%s %0.0f',text8,eqNumI);
line9 = sprintf('%s %0.0f',text9,eqNumJ);

A{1} = line1;
A{2} = line2;
```

```
A{3} = line3;
A{4} = line4;
A{5} = line5;
A{6} = line6;
A{7} = line7;
A{8} = line8;
A{9} = line9;

% Printing A{} in Text File
fid0=sprintf('VDMFRun%d.tcl',PopNum);
fid = fopen(fid0, 'wt');
for i = 1:numel(A)
    if A{i+1} == -1
        fprintf(fid,'%s', A{i});
        break
    else
        fprintf(fid,'%s \n', A{i});
    end
end
fclose(fid);
% Executing Opensees operations & Collecting Results
% Write In File
fid = fopen(fid0, 'wt');
for i = 1:numel(A)
    if A{i+1} == -1
        fprintf(fid,'%s', A{i});
        break
    else
        fprintf(fid,'%s\n', A{i});
    end
end
fclose(fid);

% calling OpenSees
pause(30)
system(sprintf('OpenSees.exe %s',fid0));
% % % system(sprintf('mpiexec -np %d OpenSeesMP.exe %s',np, fid0));
pause(30)
OpenSeesRun=sprintf('%s is done!',fid0)
pause(30)
Counter=(eqNumJ-eqNumI+1)*length(Dir);
fileID=cell(1,Counter);
fileID2=cell(1,Counter);
SaveMat=zeros(eqNumJ-eqNumI+1,length(Dir),N);
for eq=eqNumI:eqNumJ
    for dir=1:length(Dir)
        for Name=1:length(fileID)

fileID{Name}=sprintf('Final1_%d_%d_%d_%s.out',intensity,PopNum,eq,Dir{dir});

fileID2{Name}=sprintf('Final2_%d_%d_%d_%s.out',intensity,PopNum,eq,Dir{dir});
            while exist(fileID{Name})==0 || exist(fileID2{Name})==0
                % Editing Cells in A{}
                text8 = 'set eqNumI';
                text9 = 'set eqNumJ';
                line8 = sprintf('%s %0.0f',text8,eq);
                line9 = sprintf('%s %0.0f',text9,eq);
                A{8} = line8;
                A{9} = line9;
                % Printing A{} in Text File
                fid0=sprintf('VDMFRun%d.tcl',PopNum);
                fid = fopen(fid0, 'wt');
                for i = 1:numel(A)
                    if A{i+1} == -1
                        fprintf(fid,'%s', A{i});
                        break
                    else
                        fprintf(fid,'%s \n', A{i});
                    end
                end
                fclose(fid);
                % Executing Opensees operations & Collecting Results
                % Write In File
                fid = fopen(fid0, 'wt');
                for i = 1:numel(A)
                    if A{i+1} == -1
                        fprintf(fid,'%s', A{i});
                        break
                    else
                        fprintf(fid,'%s\n', A{i});
                    end
                end
                fclose(fid);
                OpenSeesRun='earthquake is missed here (Loop)!'
                system(sprintf('OpenSees.exe %s',fid0));
            end
            raw=load(fileID{Name});
            raw1=load(fileID2{Name});
            if (raw(2)- raw(1))<=0.1 && (raw1(2)- raw1(1))<=0.1
                SaveMat(eq,dir,PopNum)=1;
            else
                SaveMat(eq,dir,PopNum)=0;
            end
        end
    end
end

% re-change file to its original status
text8 = 'set eqNumI';
text9 = 'set eqNumJ';
line8 = sprintf('%s %0.0f',text8,eqNumI);
line9 = sprintf('%s %0.0f',text9,eqNumJ);
A{8} = line8;
A{9} = line9;
% Printing A{} in Text File
fid0=sprintf('VDMFRun%d.tcl',PopNum);
fid = fopen(fid0, 'wt');
for i = 1:numel(A)
    if A{i+1} == -1
        fprintf(fid,'%s', A{i});
        break
    else
        fprintf(fid,'%s \n', A{i});
    end
end
fclose(fid);
% Write In File
fid = fopen(fid0, 'wt');
for i = 1:numel(A)
    if A{i+1} == -1
        fprintf(fid,'%s', A{i});
        break
    else
        fprintf(fid,'%s\n', A{i});
    end
end
fclose(fid);
%
EQKeepMat=[];
for eq=eqNumI:eqNumJ
    if  mean(SaveMat(eq,:,PopNum))==1
        EQKeepMat=[EQKeepMat eq];
    end
end
EQKeepMat=unique(EQKeepMat);
% if numel(EQKeepMat) <= (eqNumJ-eqNumI)*0.8
if numel(EQKeepMat) < (eqNumJ-eqNumI)+1
    EQKeepMat=[];
end
save(['EQKeepMat_' num2str(PopNum) '.mat'],'EQKeepMat');

% abs of drift and acceleration of the floors
Unit_Conversion_Drift=0.01;
Unit_Conversion_Acc=386.4;
Data_Drift1=[];
Data_Drift2=[];
Data_Acc1=[];
Data_Acc2=[];
Data_Res_Drift1=[];
Data_Res_Drift2=[];
MaxofMaxAbsDrifts=zeros(numel(EQKeepMat),3);
MaxofMaxAbsAccs=zeros(numel(EQKeepMat),3);
MaxofMaxAbsResDrifts=zeros(numel(EQKeepMat),3);
AvgofMaxofMaxAbsDriftstot=zeros(length(intensity),3);
AvgofMaxofMaxAbsAccstot=zeros(length(intensity),3);
AvgofMaxofMaxAbsResDriftstot=zeros(length(intensity),3);
MedianofMaxofMaxAbsDrifts=zeros(length(intensity),3);
MedianofMaxofMaxAbsAccs=zeros(length(intensity),3);
MedianofMaxofMaxAbsResDrifts=zeros(length(intensity),3);
NinetiethofMaxofMaxAbsDrifts=zeros(length(intensity),3);
NinetiethofMaxofMaxAbsAccs=zeros(length(intensity),3);
NinetiethofMaxofMaxAbsResDrifts=zeros(length(intensity),3);

for ii=1:length(intensity)
    if isempty(EQKeepMat)==1
        jj=1;
        MaxofMaxAbsDrifts(jj,:)=10+unifrnd(0,1);
        MaxofMaxAbsAccs(jj,:)=10+unifrnd(0,1);
        MaxofMaxAbsResDrifts(jj,:)=10+unifrnd(0,1);
    end
    if isempty(EQKeepMat)~=1
        for jj=1:numel(EQKeepMat)
            for kk=1:length(Dir)
                if kk==1
                    filename1=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/Drift-
Story1_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj),Dir{kk});
                    filename2=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/Drift-
Story2_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj),Dir{kk});
                    filename3=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/Drift-
Story3_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj),Dir{kk});

                    story1_res=load(filename1);
                    story2_res=load(filename2);
                    story3_res=load(filename3);
                    A=size(story1_res,1);
                    B=size(story2_res,1);
                    C=size(story3_res,1);
                    D=min([A B C]);
                    Data_Drift1=[story1_res(1:D) story2_res(1:D)
story3_res(1:D)]./Unit_Conversion_Drift;
                    Data_Res_Drift1=[abs(story1_res(end))
abs(story2_res(end)) abs(story3_res(end))]./Unit_Conversion_Drift;

                    filename7=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/FloorAccEnv_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj
),Dir{kk});
                    Data_Acc1=[load(filename7)];
                    Data_Acc1=Data_Acc1(end,2:4)./Unit_Conversion_Acc;
                end

                if kk==2
                    filename4=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/Drift-
Story1_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj),Dir{kk});
                    filename5=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/Drift-
Story2_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj),Dir{kk});
                    filename6=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/Drift-
Story3_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj),Dir{kk});

                    story11_res=load(filename4);
                    story22_res=load(filename5);
                    story33_res=load(filename6);

                    AA=size(story11_res,1);
                    BB=size(story22_res,1);
                    CC=size(story33_res,1);
                    DD=min([AA BB CC]);

                    Data_Drift2=[story11_res(1:DD) story22_res(1:DD)
story33_res(1:DD)]./Unit_Conversion_Drift;
```

```
                    Data_Res_Drift2=[abs(story11_res(end))
abs(story22_res(end)) abs(story33_res(end))]./Unit_Conversion_Drift;

                    filename8=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/FloorAccEnv_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj
),Dir{kk});
                    Data_Acc2=[load(filename8)];
                    Data_Acc2=Data_Acc2(end,2:4)./Unit_Conversion_Acc;
                end
            end

            if  length(Dir)==1
                MaxDrifts=[max(abs(Data_Drift1))];
                MaxAccs=[(abs(Data_Acc1))];
                MaxResDrifts=[(abs(Data_Res_Drift1))];
            elseif length(Dir)==2
                MaxDrifts=max([max(abs(Data_Drift1));max(abs(Data_Drift2))]);
                MaxAccs=max([(abs(Data_Acc1));(abs(Data_Acc2))]);

MaxResDrifts=max([(abs(Data_Res_Drift1));(abs(Data_Res_Drift2))]);
            end
            MaxofMaxAbsDrifts(jj,:)=MaxDrifts;
            MaxofMaxAbsAccs(jj,:)=MaxAccs;
            MaxofMaxAbsResDrifts(jj,:)=MaxResDrifts;
        end
    end
    if Result==1
        AvgofMaxofMaxAbsDriftstot(ii,:)=mean(MaxofMaxAbsDrifts);
        AvgofMaxofMaxAbsAccstot(ii,:)=mean(MaxofMaxAbsAccs);
        AvgofMaxofMaxAbsResDriftstot(ii,:)=mean(MaxofMaxAbsResDrifts);
    elseif Result==2

MedianofMaxofMaxAbsDrifts(ii,:)=[prctile_one(MaxofMaxAbsDrifts(:,1),0.5)
prctile_one(MaxofMaxAbsDrifts(:,2),0.5)
prctile_one(MaxofMaxAbsDrifts(:,3),0.5)];
        MedianofMaxofMaxAbsAccs(ii,:)=[prctile_one(MaxofMaxAbsAccs(:,1),0.5)
prctile_one(MaxofMaxAbsAccs(:,2),0.5) prctile_one(MaxofMaxAbsAccs(:,3),0.5)];

MedianofMaxofMaxAbsResDrifts(ii,:)=[prctile_one(MaxofMaxAbsResDrifts(:,1),0.5
) prctile_one(MaxofMaxAbsResDrifts(:,2),0.5)
prctile_one(MaxofMaxAbsResDrifts(:,3),0.5)];
    elseif Result==3

NinetiethofMaxofMaxAbsDrifts(ii,:)=[prctile_one(MaxofMaxAbsDrifts(:,1),1)
prctile_one(MaxofMaxAbsDrifts(:,1),1) prctile_one(MaxofMaxAbsDrifts(:,3),1)];
        NinetiethofMaxofMaxAbsAccs(ii,:)=[prctile_one(MaxofMaxAbsAccs(:,1),1)
prctile_one(MaxofMaxAbsAccs(:,2),1) prctile_one(MaxofMaxAbsAccs(:,3),1)];

NinetiethofMaxofMaxAbsResDrifts(ii,:)=[prctile_one(MaxofMaxAbsResDrifts(:,1),
1) prctile_one(MaxofMaxAbsResDrifts(:,2),1)
prctile_one(MaxofMaxAbsResDrifts(:,3),1)];
    end
end
%%
y11=AvgofMaxofMaxAbsDriftstot;
y12=MedianofMaxofMaxAbsDrifts;
```

```
y13=NinetiethofMaxofMaxAbsDrifts;

y21=AvgofMaxofMaxAbsAccstot;
y22=MedianofMaxofMaxAbsAccs;
y23=NinetiethofMaxofMaxAbsAccs;

y31=AvgofMaxofMaxAbsResDriftstot;
y32=MedianofMaxofMaxAbsResDrifts;
y33=NinetiethofMaxofMaxAbsResDrifts;

if Drift_Constraint=='None'
    Drift_Constraint=100;
end
if Acc_Constraint=='None'
    Acc_Constraint=100;
end
if Res_Drift_Constraint=='None'
    Res_Drift_Constraint=100;
end
Stories=3;
if Result==1
    y1=y11;
    y2=y21;
    y3=y31;
    if Drift_Constraint_method==1 || Acc_Constraint_method==1 ||
Res_Drift_Constraint_method==1
        for i=1:Stories
            if y11(1,i) > Drift_Constraint || y21(1,i) > Acc_Constraint ||
y31(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end

    if Drift_Constraint_method==2 || Acc_Constraint_method==2 ||
Res_Drift_Constraint_method==2
        for i=1:Stories
            if  y12(1,i) > Drift_Constraint || y22(1,i) > Acc_Constraint ||
y32(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end

    if Drift_Constraint_method==3 || Acc_Constraint_method==3 ||
Res_Drift_Constraint_method==3
        for i=1:Stories
            if  y13(1,i) > Drift_Constraint || y23(1,i) > Acc_Constraint ||
y33(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end
```

```
end

if Result==2
    y1=y12;
    y2=y22;
    y3=y32;
    if Drift_Constraint_method==1 || Acc_Constraint_method==1 ||
Res_Drift_Constraint_method==1
        for i=1:Stories
            if  y11(1,i) > Drift_Constraint || y21(1,i) > Acc_Constraint ||
y31(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end
    if Drift_Constraint_method==2 || Acc_Constraint_method==2 ||
Res_Drift_Constraint_method==2
        for i=1:Stories
            if  y12(1,i) > Drift_Constraint || y22(1,i) > Acc_Constraint ||
y32(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end

    if Drift_Constraint_method==3 || Acc_Constraint_method==3 ||
Res_Drift_Constraint_method==3
        for i=1:Stories
            if  y13(1,i) > Drift_Constraint || y23(1,i) > Acc_Constraint ||
y33(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end
end

if Result==3
    y1=y13;
    y2=y23;
    y3=y33;
    if Drift_Constraint_method==1 || Acc_Constraint_method==1 ||
Res_Drift_Constraint_method==1
        for i=1:Stories
            if  y11(1,i) > Drift_Constraint || y21(1,i) > Acc_Constraint ||
y31(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end
    if Drift_Constraint_method==2 || Acc_Constraint_method==2 ||
Res_Drift_Constraint_method==2
        for i=1:Stories
```

```
            if  y12(1,i) > Drift_Constraint || y22(1,i) > Acc_Constraint ||
y32(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end

    if Drift_Constraint_method==3 || Acc_Constraint_method==3 ||
Res_Drift_Constraint_method==3
        for i=1:Stories
            if  y13(1,i) > Drift_Constraint || y23(1,i) > Acc_Constraint ||
y33(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end
end
y=[y1;y2;y3;C12;C3;C12+C3];
```

Cost Function, Platform III

```
function
y=cost63(x,PopNum,np,intensity,Dir,eqNumI,eqNumJ,Drift_Constraint,Acc_Constra
int,Res_Drift_Constraint,Drift_Constraint_method,Acc_Constraint_method,Res_Dr
ift_Constraint_method,Result,N)
% Inputs
K = x(1);
C12 = x(2);
C3 = x(3);
a = x(4);
% Opening .tcl file, Reading & Saving in A{}
fid=fopen('VDMF.Original3.tcl','rt');
i = 1;
tline = fgetl(fid);
A{i} = tline;
while ischar(tline)
    i = i+1;
    tline = fgetl(fid);
    A{i} = tline;
end
fclose(fid);
% Editing Cells in A{}
text1 = 'set Pop';
text2 = 'set K';
text3 = 'set C12';
text4 = 'set C3';
text5 = 'set a';
text6 = 'set HazardList';
text7 = 'set eqDirList';
text8 = 'set eqNumI';
text9 = 'set eqNumJ';

line1= sprintf('%s {%0.0f}',text1,PopNum);
line2= sprintf('%s %0.2f',text2,K);
line3 = sprintf('%s %0.2f',text3,C12);
line4 = sprintf('%s %0.2f',text4,C3);
line5 = sprintf('%s %0.2f',text5,a);

if length(intensity)==1
    line6 = sprintf('%s {%0.0f}',text6,intensity(:));
elseif length(intensity)==2
    line6 = sprintf('%s {%0.0f %0.0f}',text6,intensity(:));
else
    line6 = sprintf('%s {%0.0f %0.0f %0.0f}',text6,intensity(:));
end
if length(Dir)==1
    line7 = sprintf('%s {"%s"}',text7,Dir{:});
else
    line7 = sprintf('%s {"%s" "%s"}',text7,Dir{:});
end
line8 = sprintf('%s %0.0f',text8,eqNumI);
line9 = sprintf('%s %0.0f',text9,eqNumJ);

A{1} = line1;
A{2} = line2;
```

```
A{3} = line3;
A{4} = line4;
A{5} = line5;
A{6} = line6;
A{7} = line7;
A{8} = line8;
A{9} = line9;

% Printing A{} in Text File
fid0=sprintf('VDMFRun%d.tcl',PopNum);
fid = fopen(fid0, 'wt');
for i = 1:numel(A)
    if A{i+1} == -1
        fprintf(fid,'%s', A{i});
        break
    else
        fprintf(fid,'%s \n', A{i});
    end
end
fclose(fid);
% Executing Opensees operations & Collecting Results
% Write In File
fid = fopen(fid0, 'wt');
for i = 1:numel(A)
    if A{i+1} == -1
        fprintf(fid,'%s', A{i});
        break
    else
        fprintf(fid,'%s\n', A{i});
    end
end
fclose(fid);

% calling OpenSees
pause(30)
% % system(sprintf('OpenSees.exe %s',fid0));
system(sprintf('mpiexec -np %d OpenSeesMP.exe %s',np, fid0));
pause(30)
OpenSeesRun=sprintf('%s is done!',fid0)
pause(30)
Counter=(eqNumJ-eqNumI+1)*length(Dir);
fileID=cell(1,Counter);
fileID2=cell(1,Counter);
SaveMat=zeros(eqNumJ-eqNumI+1,length(Dir),N);
for eq=eqNumI:eqNumJ
    for dir=1:length(Dir)
        for Name=1:length(fileID)

fileID{Name}=sprintf('Final1_%d_%d_%d_%s.out',intensity,PopNum,eq,Dir{dir});

fileID2{Name}=sprintf('Final2_%d_%d_%d_%s.out',intensity,PopNum,eq,Dir{dir});
            while exist(fileID{Name})==0 || exist(fileID2{Name})==0
                % Editing Cells in A{}
                text8 = 'set eqNumI';
                text9 = 'set eqNumJ';
                line8 = sprintf('%s %0.0f',text8,eq);
                line9 = sprintf('%s %0.0f',text9,eq);
                A{8} = line8;
                A{9} = line9;
                % Printing A{} in Text File
                fid0=sprintf('VDMFRun%d.tcl',PopNum);
                fid = fopen(fid0, 'wt');
                for i = 1:numel(A)
                    if A{i+1} == -1
                        fprintf(fid,'%s', A{i});
                        break
                    else
                        fprintf(fid,'%s \n', A{i});
                    end
                end
                fclose(fid);
                % Executing Opensees operations & Collecting Results
                % Write In File
                fid = fopen(fid0, 'wt');
                for i = 1:numel(A)
                    if A{i+1} == -1
                        fprintf(fid,'%s', A{i});
                        break
                    else
                        fprintf(fid,'%s\n', A{i});
                    end
                end
                fclose(fid);
                OpenSeesRun='earthquake is missed here (Loop)!'
                system(sprintf('OpenSees.exe %s',fid0));
            end
            raw=load(fileID{Name});
            raw1=load(fileID2{Name});
            if (raw(2)- raw(1))<=0.1 && (raw1(2)- raw1(1))<=0.1
                SaveMat(eq,dir,PopNum)=1;
            else
                SaveMat(eq,dir,PopNum)=0;
            end
        end
    end
end

% re-change file to its original status
text8 = 'set eqNumI';
text9 = 'set eqNumJ';
line8 = sprintf('%s %0.0f',text8,eqNumI);
line9 = sprintf('%s %0.0f',text9,eqNumJ);
A{8} = line8;
A{9} = line9;
% Printing A{} in Text File
fid0=sprintf('VDMFRun%d.tcl',PopNum);
fid = fopen(fid0, 'wt');
for i = 1:numel(A)
    if A{i+1} == -1
        fprintf(fid,'%s', A{i});
        break
    else
        fprintf(fid,'%s \n', A{i});
    end
end
fclose(fid);
% Write In File
fid = fopen(fid0, 'wt');
for i = 1:numel(A)
    if A{i+1} == -1
        fprintf(fid,'%s', A{i});
        break
    else
        fprintf(fid,'%s\n', A{i});
    end
end
fclose(fid);
%
EQKeepMat=[];
for eq=eqNumI:eqNumJ
    if  mean(SaveMat(eq,:,PopNum))==1
        EQKeepMat=[EQKeepMat eq];
    end
end
EQKeepMat=unique(EQKeepMat);
% if numel(EQKeepMat) <= (eqNumJ-eqNumI)*0.8
if numel(EQKeepMat) < (eqNumJ-eqNumI)+1
    EQKeepMat=[];
end
save(['EQKeepMat_' num2str(PopNum) '.mat'],'EQKeepMat');

% abs of drift and acceleration of the floors
Unit_Conversion_Drift=0.01;
Unit_Conversion_Acc=386.4;
Data_Drift1=[];
Data_Drift2=[];
Data_Acc1=[];
Data_Acc2=[];
Data_Res_Drift1=[];
Data_Res_Drift2=[];
MaxofMaxAbsDrifts=zeros(numel(EQKeepMat),3);
MaxofMaxAbsAccs=zeros(numel(EQKeepMat),3);
MaxofMaxAbsResDrifts=zeros(numel(EQKeepMat),3);
AvgofMaxofMaxAbsDriftstot=zeros(length(intensity),3);
AvgofMaxofMaxAbsAccstot=zeros(length(intensity),3);
AvgofMaxofMaxAbsResDriftstot=zeros(length(intensity),3);
MedianofMaxofMaxAbsDrifts=zeros(length(intensity),3);
MedianofMaxofMaxAbsAccs=zeros(length(intensity),3);
MedianofMaxofMaxAbsResDrifts=zeros(length(intensity),3);
NinetiethofMaxofMaxAbsDrifts=zeros(length(intensity),3);
NinetiethofMaxofMaxAbsAccs=zeros(length(intensity),3);
NinetiethofMaxofMaxAbsResDrifts=zeros(length(intensity),3);

for ii=1:length(intensity)
    if isempty(EQKeepMat)==1
        jj=1;
        MaxofMaxAbsDrifts(jj,:)=10+unifrnd(0,1);
        MaxofMaxAbsAccs(jj,:)=10+unifrnd(0,1);
        MaxofMaxAbsResDrifts(jj,:)=10+unifrnd(0,1);
    end
    if isempty(EQKeepMat)~=1
        for jj=1:numel(EQKeepMat)
            for kk=1:length(Dir)
                if kk==1
                    filename1=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/Drift-
Story1_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj),Dir{kk});
                    filename2=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/Drift-
Story2_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj),Dir{kk});
                    filename3=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/Drift-
Story3_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj),Dir{kk});

                    story1_res=load(filename1);
                    story2_res=load(filename2);
                    story3_res=load(filename3);
                    A=size(story1_res,1);
                    B=size(story2_res,1);
                    C=size(story3_res,1);
                    D=min([A B C]);
                    Data_Drift1=[story1_res(1:D) story2_res(1:D)
story3_res(1:D)]./Unit_Conversion_Drift;
                    Data_Res_Drift1=[abs(story1_res(end))
abs(story2_res(end)) abs(story3_res(end))]./Unit_Conversion_Drift;

                    filename7=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/FloorAccEnv_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj
),Dir{kk});
                    Data_Acc1=[load(filename7)];
                    Data_Acc1=Data_Acc1(end,2:4)./Unit_Conversion_Acc;
                end

                if kk==2
                    filename4=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/Drift-
Story1_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj),Dir{kk});
                    filename5=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/Drift-
Story2_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj),Dir{kk});
                    filename6=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/Drift-
Story3_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj),Dir{kk});

                    story11_res=load(filename4);
                    story22_res=load(filename5);
                    story33_res=load(filename6);

                    AA=size(story11_res,1);
                    BB=size(story22_res,1);
                    CC=size(story33_res,1);
                    DD=min([AA BB CC]);

                    Data_Drift2=[story11_res(1:DD) story22_res(1:DD)
story33_res(1:DD)]./Unit_Conversion_Drift;
```

```
                    Data_Res_Drift2=[abs(story11_res(end))
abs(story22_res(end)) abs(story33_res(end))]./Unit_Conversion_Drift;

                    filename8=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/FloorAccEnv_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj
),Dir{kk});
                    Data_Acc2=[load(filename8)];
                    Data_Acc2=Data_Acc2(end,2:4)./Unit_Conversion_Acc;
                end
            end

            if  length(Dir)==1
                MaxDrifts=[max(abs(Data_Drift1))];
                MaxAccs=[(abs(Data_Acc1))];
                MaxResDrifts=[(abs(Data_Res_Drift1))];
            elseif length(Dir)==2
                MaxDrifts=max([max(abs(Data_Drift1));max(abs(Data_Drift2))]);
                MaxAccs=max([(abs(Data_Acc1));(abs(Data_Acc2))]);

MaxResDrifts=max([(abs(Data_Res_Drift1));(abs(Data_Res_Drift2))]);
            end
            MaxofMaxAbsDrifts(jj,:)=MaxDrifts;
            MaxofMaxAbsAccs(jj,:)=MaxAccs;
            MaxofMaxAbsResDrifts(jj,:)=MaxResDrifts;
        end
    end
    if Result==1
        AvgofMaxofMaxAbsDriftstot(ii,:)=mean(MaxofMaxAbsDrifts);
        AvgofMaxofMaxAbsAccstot(ii,:)=mean(MaxofMaxAbsAccs);
        AvgofMaxofMaxAbsResDriftstot(ii,:)=mean(MaxofMaxAbsResDrifts);
    elseif Result==2

MedianofMaxofMaxAbsDrifts(ii,:)=[prctile_one(MaxofMaxAbsDrifts(:,1),0.5)
prctile_one(MaxofMaxAbsDrifts(:,2),0.5)
prctile_one(MaxofMaxAbsDrifts(:,3),0.5)];
        MedianofMaxofMaxAbsAccs(ii,:)=[prctile_one(MaxofMaxAbsAccs(:,1),0.5)
prctile_one(MaxofMaxAbsAccs(:,2),0.5) prctile_one(MaxofMaxAbsAccs(:,3),0.5)];

MedianofMaxofMaxAbsResDrifts(ii,:)=[prctile_one(MaxofMaxAbsResDrifts(:,1),0.5
) prctile_one(MaxofMaxAbsResDrifts(:,2),0.5)
prctile_one(MaxofMaxAbsResDrifts(:,3),0.5)];
    elseif Result==3

NinetiethofMaxofMaxAbsDrifts(ii,:)=[prctile_one(MaxofMaxAbsDrifts(:,1),1)
prctile_one(MaxofMaxAbsDrifts(:,1),1) prctile_one(MaxofMaxAbsDrifts(:,3),1)];
        NinetiethofMaxofMaxAbsAccs(ii,:)=[prctile_one(MaxofMaxAbsAccs(:,1),1)
prctile_one(MaxofMaxAbsAccs(:,2),1) prctile_one(MaxofMaxAbsAccs(:,3),1)];

NinetiethofMaxofMaxAbsResDrifts(ii,:)=[prctile_one(MaxofMaxAbsResDrifts(:,1),
1) prctile_one(MaxofMaxAbsResDrifts(:,2),1)
prctile_one(MaxofMaxAbsResDrifts(:,3),1)];
    end
end
%%
y11=AvgofMaxofMaxAbsDriftstot;
y12=MedianofMaxofMaxAbsDrifts;
```

```
y13=NinetiethofMaxofMaxAbsDrifts;

y21=AvgofMaxofMaxAbsAccstot;
y22=MedianofMaxofMaxAbsAccs;
y23=NinetiethofMaxofMaxAbsAccs;

y31=AvgofMaxofMaxAbsResDriftstot;
y32=MedianofMaxofMaxAbsResDrifts;
y33=NinetiethofMaxofMaxAbsResDrifts;

if Drift_Constraint=='None'
    Drift_Constraint=100;
end
if Acc_Constraint=='None'
    Acc_Constraint=100;
end
if Res_Drift_Constraint=='None'
    Res_Drift_Constraint=100;
end
Stories=3;
if Result==1
    y1=y11;
    y2=y21;
    y3=y31;
    if Drift_Constraint_method==1 || Acc_Constraint_method==1 ||
Res_Drift_Constraint_method==1
        for i=1:Stories
            if y11(1,i) > Drift_Constraint || y21(1,i) > Acc_Constraint ||
y31(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end

    if Drift_Constraint_method==2 || Acc_Constraint_method==2 ||
Res_Drift_Constraint_method==2
        for i=1:Stories
            if  y12(1,i) > Drift_Constraint || y22(1,i) > Acc_Constraint ||
y32(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end

    if Drift_Constraint_method==3 || Acc_Constraint_method==3 ||
Res_Drift_Constraint_method==3
        for i=1:Stories
            if  y13(1,i) > Drift_Constraint || y23(1,i) > Acc_Constraint ||
y33(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end
```

```
end

if Result==2
    y1=y12;
    y2=y22;
    y3=y32;
    if Drift_Constraint_method==1 || Acc_Constraint_method==1 ||
Res_Drift_Constraint_method==1
        for i=1:Stories
            if  y11(1,i) > Drift_Constraint || y21(1,i) > Acc_Constraint ||
y31(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end
    if Drift_Constraint_method==2 || Acc_Constraint_method==2 ||
Res_Drift_Constraint_method==2
        for i=1:Stories
            if  y12(1,i) > Drift_Constraint || y22(1,i) > Acc_Constraint ||
y32(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end

    if Drift_Constraint_method==3 || Acc_Constraint_method==3 ||
Res_Drift_Constraint_method==3
        for i=1:Stories
            if  y13(1,i) > Drift_Constraint || y23(1,i) > Acc_Constraint ||
y33(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end
end

if Result==3
    y1=y13;
    y2=y23;
    y3=y33;
    if Drift_Constraint_method==1 || Acc_Constraint_method==1 ||
Res_Drift_Constraint_method==1
        for i=1:Stories
            if  y11(1,i) > Drift_Constraint || y21(1,i) > Acc_Constraint ||
y31(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end
    if Drift_Constraint_method==2 || Acc_Constraint_method==2 ||
Res_Drift_Constraint_method==2
        for i=1:Stories
```

```
            if  y12(1,i) > Drift_Constraint || y22(1,i) > Acc_Constraint ||
y32(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end

    if Drift_Constraint_method==3 || Acc_Constraint_method==3 ||
Res_Drift_Constraint_method==3
        for i=1:Stories
            if  y13(1,i) > Drift_Constraint || y23(1,i) > Acc_Constraint ||
y33(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end
end
y=[y1;y2;y3;C12;C3;C12+C3];
```

Cost Function, Platform IV

```
function
y=cost64(x,PopNum,np,intensity,Dir,eqNumI,eqNumJ,Drift_Constraint,Acc_Constra
int,Res_Drift_Constraint,Drift_Constraint_method,Acc_Constraint_method,Res_Dr
ift_Constraint_method,Result,N)
% Inputs
K = x(1);
C12 = x(2);
C3 = x(3);
a = x(4);
% Opening .tcl file, Reading & Saving in A{}
fid=fopen('VDMF.Original4.tcl','rt');
i = 1;
tline = fgetl(fid);
A{i} = tline;
while ischar(tline)
    i = i+1;
    tline = fgetl(fid);
    A{i} = tline;
end
fclose(fid);
% Editing Cells in A{}
text1 = 'set Pop';
text2 = 'set K';
text3 = 'set C12';
text4 = 'set C3';
text5 = 'set a';
text6 = 'set HazardList';
text7 = 'set eqDirList';
text8 = 'set eqNumI';
text9 = 'set eqNumJ';

line1= sprintf('%s {%0.0f}',text1,PopNum);
line2= sprintf('%s %0.2f',text2,K);
line3 = sprintf('%s %0.2f',text3,C12);
line4 = sprintf('%s %0.2f',text4,C3);
line5 = sprintf('%s %0.2f',text5,a);

if length(intensity)==1
    line6 = sprintf('%s {%0.0f}',text6,intensity(:));
elseif length(intensity)==2
    line6 = sprintf('%s {%0.0f %0.0f}',text6,intensity(:));
else
    line6 = sprintf('%s {%0.0f %0.0f %0.0f}',text6,intensity(:));
end
if length(Dir)==1
    line7 = sprintf('%s {"%s"}',text7,Dir{:});
else
    line7 = sprintf('%s {"%s" "%s"}',text7,Dir{:});
end
line8 = sprintf('%s %0.0f',text8,eqNumI);
line9 = sprintf('%s %0.0f',text9,eqNumJ);

A{1} = line1;
A{2} = line2;
```

```
A{3} = line3;
A{4} = line4;
A{5} = line5;
A{6} = line6;
A{7} = line7;
A{8} = line8;
A{9} = line9;

% Printing A{} in Text File
fid0=sprintf('VDMFRun%d.tcl',PopNum);
fid = fopen(fid0, 'wt');
for i = 1:numel(A)
    if A{i+1} == -1
        fprintf(fid,'%s', A{i});
        break
    else
        fprintf(fid,'%s \n', A{i});
    end
end
fclose(fid);
% Executing Opensees operations & Collecting Results
% Write In File
fid = fopen(fid0, 'wt');
for i = 1:numel(A)
    if A{i+1} == -1
        fprintf(fid,'%s', A{i});
        break
    else
        fprintf(fid,'%s\n', A{i});
    end
end
fclose(fid);

% calling OpenSees
pause(30)
% % % system(sprintf('OpenSees.exe %s',fid0));
system(sprintf('mpiexec -np %d OpenSeesMP.exe %s',np, fid0));
pause(30)
OpenSeesRun=sprintf('%s is done!',fid0)
pause(30)
Counter=(eqNumJ-eqNumI+1)*length(Dir);
fileID=cell(1,Counter);
fileID2=cell(1,Counter);
SaveMat=zeros(eqNumJ-eqNumI+1,length(Dir),N);
for eq=eqNumI:eqNumJ
    for dir=1:length(Dir)
        for Name=1:length(fileID)

fileID{Name}=sprintf('Final1_%d_%d_%d_%s.out',intensity,PopNum,eq,Dir{dir});

fileID2{Name}=sprintf('Final2_%d_%d_%d_%s.out',intensity,PopNum,eq,Dir{dir});
            while exist(fileID{Name})==0 || exist(fileID2{Name})==0
                % Editing Cells in A{}
                text8 = 'set eqNumI';
                text9 = 'set eqNumJ';
                line8 = sprintf('%s %0.0f',text8,eq);
                line9 = sprintf('%s %0.0f',text9,eq);
                A{8} = line8;
                A{9} = line9;
                % Printing A{} in Text File
                fid0=sprintf('VDMFRun%d.tcl',PopNum);
                fid = fopen(fid0, 'wt');
                for i = 1:numel(A)
                    if A{i+1} == -1
                        fprintf(fid,'%s', A{i});
                        break
                    else
                        fprintf(fid,'%s \n', A{i});
                    end
                end
                fclose(fid);
                % Executing Opensees operations & Collecting Results
                % Write In File
                fid = fopen(fid0, 'wt');
                for i = 1:numel(A)
                    if A{i+1} == -1
                        fprintf(fid,'%s', A{i});
                        break
                    else
                        fprintf(fid,'%s\n', A{i});
                    end
                end
                fclose(fid);
                OpenSeesRun='earthquake is missed here (loop)!'
                system(sprintf('OpenSees.exe %s',fid0));
                pause(10)
            end
            raw=load(fileID{Name});
            raw1=load(fileID2{Name});
            if (raw(2)- raw(1))<=0.1 && (raw1(2)- raw1(1))<=0.1
                SaveMat(eq,dir,PopNum)=1;
            else
                SaveMat(eq,dir,PopNum)=0;
            end
        end
    end
end

% re-change file to its original status
text8 = 'set eqNumI';
text9 = 'set eqNumJ';
line8 = sprintf('%s %0.0f',text8,eqNumI);
line9 = sprintf('%s %0.0f',text9,eqNumJ);
A{8} = line8;
A{9} = line9;
% Printing A{} in Text File
fid0=sprintf('VDMFRun%d.tcl',PopNum);
fid = fopen(fid0, 'wt');
for i = 1:numel(A)
    if A{i+1} == -1
        fprintf(fid,'%s', A{i});
        break
    else
```

```
        fprintf(fid,'%s \n', A{i});
    end
end
fclose(fid);
% Write In File
fid = fopen(fid0, 'wt');
for i = 1:numel(A)
    if A{i+1} == -1
        fprintf(fid,'%s', A{i});
        break
    else
        fprintf(fid,'%s\n', A{i});
    end
end
fclose(fid);
%
EQKeepMat=[];
for eq=eqNumI:eqNumJ
    if  mean(SaveMat(eq,:,PopNum))==1
        EQKeepMat=[EQKeepMat eq];
    end
end
EQKeepMat=unique(EQKeepMat);
% if numel(EQKeepMat) <= (eqNumJ-eqNumI)*0.8
if numel(EQKeepMat) < (eqNumJ-eqNumI)+1
    EQKeepMat=[];
end
save(['EQKeepMat_' num2str(PopNum) '.mat'],'EQKeepMat');

% abs of drift and acceleration of the floors
Unit_Conversion_Drift=0.01;
Unit_Conversion_Acc=386.4;
Data_Drift1=[];
Data_Drift2=[];
Data_Acc1=[];
Data_Acc2=[];
Data_Res_Drift1=[];
Data_Res_Drift2=[];
MaxofMaxAbsDrifts=zeros(numel(EQKeepMat),3);
MaxofMaxAbsAccs=zeros(numel(EQKeepMat),3);
MaxofMaxAbsResDrifts=zeros(numel(EQKeepMat),3);
AvgofMaxofMaxAbsDriftstot=zeros(length(intensity),3);
AvgofMaxofMaxAbsAccstot=zeros(length(intensity),3);
AvgofMaxofMaxAbsResDriftstot=zeros(length(intensity),3);
MedianofMaxofMaxAbsDrifts=zeros(length(intensity),3);
MedianofMaxofMaxAbsAccs=zeros(length(intensity),3);
MedianofMaxofMaxAbsResDrifts=zeros(length(intensity),3);
NinetiethofMaxofMaxAbsDrifts=zeros(length(intensity),3);
NinetiethofMaxofMaxAbsAccs=zeros(length(intensity),3);
NinetiethofMaxofMaxAbsResDrifts=zeros(length(intensity),3);

for ii=1:length(intensity)
    if isempty(EQKeepMat)==1
        jj=1;
        MaxofMaxAbsDrifts(jj,:)=10+unifrnd(0,1);
        MaxofMaxAbsAccs(jj,:)=10+unifrnd(0,1);
```

```
            MaxofMaxAbsResDrifts(jj,:)=10+unifrnd(0,1);
        end
        if isempty(EQKeepMat)~=1
            for jj=1:numel(EQKeepMat)
                for kk=1:length(Dir)
                    if kk==1
                        filename1=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/Drift-
Story1_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj),Dir{kk});
                        filename2=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/Drift-
Story2_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj),Dir{kk});
                        filename3=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/Drift-
Story3_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj),Dir{kk});

                        story1_res=load(filename1);
                        story2_res=load(filename2);
                        story3_res=load(filename3);
                        A=size(story1_res,1);
                        B=size(story2_res,1);
                        C=size(story3_res,1);
                        D=min([A B C]);
                        Data_Drift1=[story1_res(1:D) story2_res(1:D)
story3_res(1:D)]./Unit_Conversion_Drift;
                        Data_Res_Drift1=[abs(story1_res(end))
abs(story2_res(end)) abs(story3_res(end))]./Unit_Conversion_Drift;

                        filename7=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/FloorAccEnv_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj
),Dir{kk});
                        Data_Acc1=[load(filename7)];
                        Data_Acc1=Data_Acc1(end,2:4)./Unit_Conversion_Acc;
                    end

                    if kk==2
                        filename4=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/Drift-
Story1_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj),Dir{kk});
                        filename5=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/Drift-
Story2_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj),Dir{kk});
                        filename6=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/Drift-
Story3_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj),Dir{kk});

                        story11_res=load(filename4);
                        story22_res=load(filename5);
                        story33_res=load(filename6);

                        AA=size(story11_res,1);
                        BB=size(story22_res,1);
                        CC=size(story33_res,1);
                        DD=min([AA BB CC]);
                        Data_Drift2=[story11_res(1:DD) story22_res(1:DD)
story33_res(1:DD)]./Unit_Conversion_Drift;
                        Data_Res_Drift2=[abs(story11_res(end))
abs(story22_res(end)) abs(story33_res(end))]./Unit_Conversion_Drift;

                        filename8=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/FloorAccEnv_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj
),Dir{kk});
                        Data_Acc2=[load(filename8)];
                        Data_Acc2=Data_Acc2(end,2:4)./Unit_Conversion_Acc;
                    end
                end

                if  length(Dir)==1
                    MaxDrifts=[max(abs(Data_Drift1))];
                    MaxAccs=[(abs(Data_Acc1))];
                    MaxResDrifts=[(abs(Data_Res_Drift1))];
                elseif length(Dir)==2
                    MaxDrifts=max([max(abs(Data_Drift1));max(abs(Data_Drift2))]);
                    MaxAccs=max([(abs(Data_Acc1));(abs(Data_Acc2))]);

MaxResDrifts=max([(abs(Data_Res_Drift1));(abs(Data_Res_Drift2))]);
                end
                MaxofMaxAbsDrifts(jj,:)=MaxDrifts;
                MaxofMaxAbsAccs(jj,:)=MaxAccs;
                MaxofMaxAbsResDrifts(jj,:)=MaxResDrifts;
            end
        end
        if Result==1
            AvgofMaxofMaxAbsDriftstot(ii,:)=mean(MaxofMaxAbsDrifts);
            AvgofMaxofMaxAbsAccstot(ii,:)=mean(MaxofMaxAbsAccs);
            AvgofMaxofMaxAbsResDriftstot(ii,:)=mean(MaxofMaxAbsResDrifts);
        elseif Result==2

MedianofMaxofMaxAbsDrifts(ii,:)=[prctile_one(MaxofMaxAbsDrifts(:,1),0.5)
prctile_one(MaxofMaxAbsDrifts(:,2),0.5)
prctile_one(MaxofMaxAbsDrifts(:,3),0.5)];
            MedianofMaxofMaxAbsAccs(ii,:)=[prctile_one(MaxofMaxAbsAccs(:,1),0.5)
prctile_one(MaxofMaxAbsAccs(:,2),0.5) prctile_one(MaxofMaxAbsAccs(:,3),0.5)];

MedianofMaxofMaxAbsResDrifts(ii,:)=[prctile_one(MaxofMaxAbsResDrifts(:,1),0.5
) prctile_one(MaxofMaxAbsResDrifts(:,2),0.5)
prctile_one(MaxofMaxAbsResDrifts(:,3),0.5)];
        elseif Result==3

NinetiethofMaxofMaxAbsDrifts(ii,:)=[prctile_one(MaxofMaxAbsDrifts(:,1),1)
prctile_one(MaxofMaxAbsDrifts(:,1),1) prctile_one(MaxofMaxAbsDrifts(:,3),1)];
            NinetiethofMaxofMaxAbsAccs(ii,:)=[prctile_one(MaxofMaxAbsAccs(:,1),1)
prctile_one(MaxofMaxAbsAccs(:,2),1) prctile_one(MaxofMaxAbsAccs(:,3),1)];

NinetiethofMaxofMaxAbsResDrifts(ii,:)=[prctile_one(MaxofMaxAbsResDrifts(:,1),
1) prctile_one(MaxofMaxAbsResDrifts(:,2),1)
prctile_one(MaxofMaxAbsResDrifts(:,3),1)];
        end
end
%%
```

```
y11=AvgofMaxofMaxAbsDriftstot;
y12=MedianofMaxofMaxAbsDrifts;
y13=NinetiethofMaxofMaxAbsDrifts;

y21=AvgofMaxofMaxAbsAccstot;
y22=MedianofMaxofMaxAbsAccs;
y23=NinetiethofMaxofMaxAbsAccs;

y31=AvgofMaxofMaxAbsResDriftstot;
y32=MedianofMaxofMaxAbsResDrifts;
y33=NinetiethofMaxofMaxAbsResDrifts;

if Drift_Constraint=='None'
    Drift_Constraint=100;
end
if Acc_Constraint=='None'
    Acc_Constraint=100;
end
if Res_Drift_Constraint=='None'
    Res_Drift_Constraint=100;
end
Stories=3;
if Result==1
    y1=y11;
    y2=y21;
    y3=y31;
    if Drift_Constraint_method==1 || Acc_Constraint_method==1 ||
Res_Drift_Constraint_method==1
        for i=1:Stories
            if y11(1,i) > Drift_Constraint || y21(1,i) > Acc_Constraint ||
y31(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end

    if Drift_Constraint_method==2 || Acc_Constraint_method==2 ||
Res_Drift_Constraint_method==2
        for i=1:Stories
            if  y12(1,i) > Drift_Constraint || y22(1,i) > Acc_Constraint ||
y32(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end

    if Drift_Constraint_method==3 || Acc_Constraint_method==3 ||
Res_Drift_Constraint_method==3
        for i=1:Stories
            if  y13(1,i) > Drift_Constraint || y23(1,i) > Acc_Constraint ||
y33(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end
end

if Result==2
    y1=y12;
    y2=y22;
    y3=y32;
    if Drift_Constraint_method==1 || Acc_Constraint_method==1 ||
Res_Drift_Constraint_method==1
        for i=1:Stories
            if  y11(1,i) > Drift_Constraint || y21(1,i) > Acc_Constraint ||
y31(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end
    if Drift_Constraint_method==2 || Acc_Constraint_method==2 ||
Res_Drift_Constraint_method==2
        for i=1:Stories
            if  y12(1,i) > Drift_Constraint || y22(1,i) > Acc_Constraint ||
y32(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end

    if Drift_Constraint_method==3 || Acc_Constraint_method==3 ||
Res_Drift_Constraint_method==3
        for i=1:Stories
            if  y13(1,i) > Drift_Constraint || y23(1,i) > Acc_Constraint ||
y33(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end
end

if Result==3
    y1=y13;
    y2=y23;
    y3=y33;
    if Drift_Constraint_method==1 || Acc_Constraint_method==1 ||
Res_Drift_Constraint_method==1
        for i=1:Stories
            if  y11(1,i) > Drift_Constraint || y21(1,i) > Acc_Constraint ||
y31(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end
    if Drift_Constraint_method==2 || Acc_Constraint_method==2 ||
Res_Drift_Constraint_method==2
        for i=1:Stories
            if  y12(1,i) > Drift_Constraint || y22(1,i) > Acc_Constraint ||
y32(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end

    if Drift_Constraint_method==3 || Acc_Constraint_method==3 ||
Res_Drift_Constraint_method==3
        for i=1:Stories
            if  y13(1,i) > Drift_Constraint || y23(1,i) > Acc_Constraint ||
y33(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end
end
y=[y1;y2;y3;C12;C3;C12+C3];
```

APPENDIX B

FIND NEW RANGES FROM BROAD RANGES OF DAMPING COEFFICIENTS

Main Function

```
clc;
clear;
close all;
diary MATLAB&OPENSEES.out;
tic
%%
% Find the acceptable range of Ci (damper's coefficients) of VDMF
%% Problem Definition
nVar=4;              % Number of Decision Variables
VarSize=[1 nVar];    % Size of Decision Variables Matrix
VarMin=[2000 0 0 0.5];              % Lower Bound of Input Variables [axial
stiffness, damping coefficient of first and second floors, damping
coefficient of third floor, velocity exponent of design]
VarMax=[2000 300 300 0.5];          % Upper Bound of Input Variables [axial
stiffness, damping coefficient of first and second floors, damping
coefficient of third floor, velocity exponent of design]
%% OpenSees Parameters
intensity=[10];                     % intensity=[50] or [10] or [2]
eqNumI=1;                          % eqNumI is the starting earthquake number
eqNumJ=3;                          % eqNumJ is the final earthquake number
Dir={'FP'};                        % Dir={'FN','FP'} or Dir={'FN'} or
Dir={'FP'}
%% Constraint tag
Drift_Constraint=1.2;                % set drift threshold  (unit Percentage
in/in) or/ 'None'.         'None' means no drift constraint
Acc_Constraint=0.5;                % set acceleration threshold (unit g) or/
'None'.                   'None' means no acceleration constraint (Unit g)
Res_Drift_Constraint=0.2;           % set residual drift threshold (unit
Percentage in/in) or/ 'None'. 'None' means no residual drift constraint (Unit
Percentage in/in)
% Constraints add on methods
Drift_Constraint_method=3;           % "1" Average,  "2" 50th percentile,
"3" 90th percentile
Acc_Constraint_method=3;
Res_Drift_Constraint_method=3;
%% Results tag, cost function calculation
% Result = 1;              % "1" generates the average of drift and
acceleration profiles
Result = 3;              % "2" generates the median (50th percentile) of
drift and acceleration profiles
% Result = 3;              % "3" generates the 90th percentile of drift and
acceleration profiles
%% Parameters
nPop=30;              % Maximum number of equally spaced  Size
NumStory=3;           % Number of stories
NumEDP=3;             % This should not change, Drift, Acc, Residual Drift
%% Initialization
empty_particle.Position=[];
empty_particle.Velocity=[];
empty_particle.Cost=[];
empty_particle.Best.Position=[];
empty_particle.Best.Cost=[];
empty_particle.IsDominated=[];
empty_particle.GridIndex=[];
empty_particle.GridSubIndex=[];
pop=repmat(empty_particle,nPop,1);

Acceptable_Cost=cell(NumStory,NumEDP,nPop);

CostFunction2=@loss62E;
mypool= parpool('local',nPop);
mypool.IdleTimeout=Inf;
C=zeros(NumStory,2);
Cdesign=zeros(NumStory,2);
Czero8=zeros(NumStory,2);
for alpha=1:2
    if alpha==1
        VarMin(4)=VarMin(4); % the velocity exponet for finding the maximum
of Ci brackets
        VarMax(4)=VarMax(4);
    end
    if alpha==2
        VarMin(4)=0.8; % the velocity exponet for finding the minimum of Ci
brackets
        VarMax(4)=0.8;
    end
    parfor  i=1:nPop
        %     PRT=i
        adjust=[];
        for ii=1:nVar
            adjust=[adjust unifrnd(VarMin(ii),VarMax(ii),[1,1])];
            if ii==2 || ii==3
                A=VarMin(ii):(VarMax(ii)-VarMin(ii))/(nPop-1):VarMax(ii);
                adjust(ii)=A(i);
            end
        end
        pop(i).Position=adjust;

pop(i).Cost=CostFunction2(pop(i).Position,i,intensity,Dir,eqNumI,eqNumJ,Drift
_Constraint,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,Acc_C
onstraint_method,Res_Drift_Constraint_method,Result,nPop);

        while  pop(i).Cost(1,1) <= 0.02 || pop(i).Cost(1,2) <= 0.02 ||
pop(i).Cost(1,3)  <= 0.02

pop(i).Cost=CostFunction2(pop(i).Position,i,intensity,Dir,eqNumI,eqNumJ,Drift
_Constraint,Acc_Constraint,Res_Drift_Constraint,Drift_Constraint_method,Acc_C
onstraint_method,Res_Drift_Constraint_method,Result,nPop);
        end
    end

    for i=1:nPop
        for j=1:NumStory
            for k=1:NumEDP
                Acceptable_Cost{k,j,i}=pop(i).Cost(k,j);
            end
        end
    end
    Decision_Mat=zeros(1,nPop,NumStory);
    for k=1:NumStory
```

```
        for i=1:nPop
            if sum([Acceptable_Cost{:,k,i}]) < 10*NumEDP
                Decision_Mat(1,i,k)=1;
            end
        end
    end
    range1=[];
    range2=[];
    range3=[];
    for k=1:NumStory
        for i=1:nPop
            if Decision_Mat(1,i,k)==1
                if k==1
                    range1=[range1 i];
                end
                if k==2
                    range2=[range2 i];
                end
                if k==3
                    range3=[range3 i];
                end
            end
        end
    end

    Acceptable_Position1=zeros(numel(range1),nVar);
    for i=1:numel(range1)
        Acceptable_Position1(i,:)=pop(range1(i)).Position;
    end

    Acceptable_Position2=zeros(numel(range2),nVar);
    for i=1:numel(range2)
        Acceptable_Position2(i,:)=pop(range2(i)).Position;
    end

    Acceptable_Position3=zeros(numel(range2),nVar);
    for i=1:numel(range3)
        Acceptable_Position3(i,:)=pop(range3(i)).Position;
    end

    C1=[min(Acceptable_Position1(:,2)) max(Acceptable_Position1(:,2))];
    C2=[min(Acceptable_Position2(:,2)) max(Acceptable_Position2(:,2))];
    C3=[min(Acceptable_Position3(:,2)) max(Acceptable_Position3(:,2))];
    if alpha==1
        C(:,alpha+1)=[C1(2);C2(2);C3(2)];
        Cdesign(:,2)=[C1(2);C2(2);C3(2)];
        Cdesign(:,1)=[C1(1);C2(1);C3(1)];
    end
    if alpha==2
        C(:,alpha-1)=[C1(1);C2(1);C3(1)];
        Czero8(:,2)=[C1(2);C2(2);C3(2)];
        Czero8(:,1)=[C1(1);C2(1);C3(1)];
    end

    for EQ=eqNumI:eqNumJ
        for POP=1:nPop
```

```
            for direct=1:length(Dir)

fID=sprintf('Final1_%d_%d_%d_%s.out',intensity,POP,EQ,Dir{direct});

fID2=sprintf('Final2_%d_%d_%d_%s.out',intensity,POP,EQ,Dir{direct});
                fID3=sprintf('Oak-%d-50-Output-%d',intensity,POP);

                if exist(fID)==2
                    delete(fID);
                end

                if exist(fID2)==2
                    delete(fID2);
                end

                if exist(fID3)==1
                    rmdir(fID3,'s');
                end

%
%
%
%
FName=sprintf('Final1_%d_%d_%d_%s.out',intensity,POP,EQ,Dir{direct});
%
FName2=sprintf('Final2_%d_%d_%d_%s.out',intensity,POP,EQ,Dir{direct});
%                  FName3=sprintf('Oak-%d-50-Output-%d',intensity,POP);
%                  delete(FName);
%                  delete(FName2);
%                  rmdir(FName3,'s')
            end
        end
    end
end
C
% reduced brackets of C12 and C3; % in this study assumed C1=C2
C12=C(1:2,1:2);
C12min=max(C12(:,1));
C12max=min(C12(:,2));
C12=[C12min C12max];
C3=[C(3,1) C(3,2)];
% the modified reduced brackets of C12 and C3
C12Modified=[C12min-((VarMax(2)-VarMin(2))/(nPop-1)) C12max+((VarMax(2)-
VarMin(2))/(nPop-1))];
C3Modified=[C(3,1)-((VarMax(2)-VarMin(2))/(nPop-1)) C(3,2)+((VarMax(2)-
VarMin(2))/(nPop-1))];
if C12Modified(1)<0
    C12Modified(1)=0;
end
if C3Modified(1)<0
    C3Modified(1)=0;
end
% the interdependency of the C12 and C3
% rule 1
MaxInterdependencyModified=(abs(max((C12Modified(2)-
C3Modified(1)),(C12Modified(1)-C3Modified(2)))));
```

```
% MaxInterdependencyModified=MaxInterdependencyModified+((VarMax(2)-
VarMin(2))/(nPop-1));
% rule 2
MinInterdependencyModified=abs(C12Modified(1)-C3Modified(1));
% MinInterdependencyModified=MinInterdependencyModified-((VarMax(2)-
VarMin(2))/(nPop-1));
if MinInterdependencyModified < 0
    MinInterdependencyModified=0;
end

% rule 3 (logical relationship between C12 and C3
if C12(1) > C3(1)
    disp='C12 is always greater than C3'
end
if C12(1) < C3(1)
    disp='C3 is always greater than C12'
end
if C12(1) == C3(1)
    disp='C12 is equal to C12'
end

if max(Cdesign(1,1),Cdesign(2,1))==0
    disp='C12 is zero'
end
if Cdesign(3,1)==0
    disp='C3 is zero'
end

CalcTime=toc;
save matlab.mat;
delete(mypool);
diary off;
```

Cost Function

```
function
y=loss62E(x,PopNum,intensity,Dir,eqNumI,eqNumJ,Drift_Constraint,Acc_Constrain
t,Res_Drift_Constraint,Drift_Constraint_method,Acc_Constraint_method,Res_Drif
t_Constraint_method,Result,N)
% Inputs
K = x(1);
C12 = x(2);
C3 = x(3);
a = x(4);
% Opening .tcl file, Reading & Saving in A{}
fid=fopen('VDMF.Original2.tcl','rt');
i = 1;
tline = fgetl(fid);
A{i} = tline;
while ischar(tline)
    i = i+1;
    tline = fgetl(fid);
    A{i} = tline;
end
fclose(fid);
```

```
% Editing Cells in A{}
text1 = 'set Pop';
text2 = 'set K';
text3 = 'set C12';
text4 = 'set C3';
text5 = 'set a';
text6 = 'set HazardList';
text7 = 'set eqDirList';
text8 = 'set eqNumI';
text9 = 'set eqNumJ';

line1= sprintf('%s {%0.0f}',text1,PopNum);
line2= sprintf('%s %0.2f',text2,K);
line3 = sprintf('%s %0.2f',text3,C12);
line4 = sprintf('%s %0.2f',text4,C3);
line5 = sprintf('%s %0.2f',text5,a);

if length(intensity)==1
    line6 = sprintf('%s {%0.0f}',text6,intensity(:));
elseif length(intensity)==2
    line6 = sprintf('%s {%0.0f %0.0f}',text6,intensity(:));
else
    line6 = sprintf('%s {%0.0f %0.0f %0.0f}',text6,intensity(:));
end
if length(Dir)==1
    line7 = sprintf('%s {"%s"}',text7,Dir{:});
else
    line7 = sprintf('%s {"%s" "%s"}',text7,Dir{:});
end
line8 = sprintf('%s %0.0f',text8,eqNumI);
line9 = sprintf('%s %0.0f',text9,eqNumJ);

A{1} = line1;
A{2} = line2;
A{3} = line3;
A{4} = line4;
A{5} = line5;
A{6} = line6;
A{7} = line7;
A{8} = line8;
A{9} = line9;

% Printing A{} in Text File
fid0=sprintf('VDMFRun%d.tcl',PopNum);
fid = fopen(fid0, 'wt');
for i = 1:numel(A)
    if A{i+1} == -1
        fprintf(fid,'%s', A{i});
        break
    else
        fprintf(fid,'%s \n', A{i});
    end
end
fclose(fid);
% Executing Opensees operations & Collecting Results
% Write In File
```

```
fid = fopen(fid0, 'wt');
for i = 1:numel(A)
    if A{i+1} == -1
        fprintf(fid,'%s', A{i});
        break
    else
        fprintf(fid,'%s\n', A{i});
    end
end
fclose(fid);

% calling OpenSees
pause(30)
system(sprintf('OpenSees.exe %s',fid0));
% % % system(sprintf('mpiexec -np %d OpenSeesMP.exe %s',np, fid0));
pause(30)
OpenSeesRun=sprintf('%s is done!',fid0)
pause(30)
Counter=(eqNumJ-eqNumI+1)*length(Dir);
fileID=cell(1,Counter);
fileID2=cell(1,Counter);
SaveMat=zeros(eqNumJ-eqNumI+1,length(Dir),N);
for eq=eqNumI:eqNumJ
    for dir=1:length(Dir)
        for Name=1:length(fileID)

fileID{Name}=sprintf('Final1_%d_%d_%d_%s.out',intensity,PopNum,eq,Dir{dir});

fileID2{Name}=sprintf('Final2_%d_%d_%d_%s.out',intensity,PopNum,eq,Dir{dir});
            while exist(fileID{Name})==0 || exist(fileID2{Name})==0
                % Editing Cells in A{}
                text8 = 'set eqNumI';
                text9 = 'set eqNumJ';
                line8 = sprintf('%s %0.0f',text8,eq);
                line9 = sprintf('%s %0.0f',text9,eq);
                A{8} = line8;
                A{9} = line9;
                % Printing A{} in Text File
                fid0=sprintf('VDMFRun%d.tcl',PopNum);
                fid = fopen(fid0, 'wt');
                for i = 1:numel(A)
                    if A{i+1} == -1
                        fprintf(fid,'%s', A{i});
                        break
                    else
                        fprintf(fid,'%s \n', A{i});
                    end
                end
                fclose(fid);
                % Executing Opensees operations & Collecting Results
                % Write In File
                fid = fopen(fid0, 'wt');
                for i = 1:numel(A)
                    if A{i+1} == -1
                        fprintf(fid,'%s', A{i});
                        break
                    else
```

```
                            fprintf(fid,'%s\n', A{i});
                        end
                    end
                    fclose(fid);
                    OpenSeesRun='earthquake is missed here (Loop)!'
                    system(sprintf('OpenSees.exe %s',fid0));
                end
                raw=load(fileID{Name});
                raw1=load(fileID2{Name});
                if (raw(2)- raw(1))<=0.1 && (raw1(2)- raw1(1))<=0.1
                    SaveMat(eq,dir,PopNum)=1;
                else
                    SaveMat(eq,dir,PopNum)=0;
                end
            end
        end
    end

% re-change file to its original status
text8 = 'set eqNumI';
text9 = 'set eqNumJ';
line8 = sprintf('%s %0.0f',text8,eqNumI);
line9 = sprintf('%s %0.0f',text9,eqNumJ);
A{8} = line8;
A{9} = line9;
% Printing A{} in Text File
fid0=sprintf('VDMFRun%d.tcl',PopNum);
fid = fopen(fid0, 'wt');
for i = 1:numel(A)
    if A{i+1} == -1
        fprintf(fid,'%s', A{i});
        break
    else
        fprintf(fid,'%s \n', A{i});
    end
end
fclose(fid);
% Write In File
fid = fopen(fid0, 'wt');
for i = 1:numel(A)
    if A{i+1} == -1
        fprintf(fid,'%s', A{i});
        break
    else
        fprintf(fid,'%s\n', A{i});
    end
end
fclose(fid);
%
EQKeepMat=[];
for eq=eqNumI:eqNumJ
    if  mean(SaveMat(eq,:,PopNum))==1
        EQKeepMat=[EQKeepMat eq];
    end
end
EQKeepMat=unique(EQKeepMat);
% if numel(EQKeepMat) <= (eqNumJ-eqNumI)*0.8
```

```
if numel(EQKeepMat) < (eqNumJ-eqNumI)+1
    EQKeepMat=[];
end
save(['EQKeepMat_' num2str(PopNum) '.mat'],'EQKeepMat');

% abs of drift and acceleration of the floors
Unit_Conversion_Drift=0.01;
Unit_Conversion_Acc=386.4;
Data_Drift1=[];
Data_Drift2=[];
Data_Acc1=[];
Data_Acc2=[];
Data_Res_Drift1=[];
Data_Res_Drift2=[];
MaxofMaxAbsDrifts=zeros(numel(EQKeepMat),3);
MaxofMaxAbsAccs=zeros(numel(EQKeepMat),3);
MaxofMaxAbsResDrifts=zeros(numel(EQKeepMat),3);
AvgofMaxofMaxAbsDriftstot=zeros(length(intensity),3);
AvgofMaxofMaxAbsAccstot=zeros(length(intensity),3);
AvgofMaxofMaxAbsResDriftstot=zeros(length(intensity),3);
MedianofMaxofMaxAbsDrifts=zeros(length(intensity),3);
MedianofMaxofMaxAbsAccs=zeros(length(intensity),3);
MedianofMaxofMaxAbsResDrifts=zeros(length(intensity),3);
NinetiethofMaxofMaxAbsDrifts=zeros(length(intensity),3);
NinetiethofMaxofMaxAbsAccs=zeros(length(intensity),3);
NinetiethofMaxofMaxAbsResDrifts=zeros(length(intensity),3);

for ii=1:length(intensity)
    if isempty(EQKeepMat)==1
        jj=1;
        MaxofMaxAbsDrifts(jj,:)=10+unifrnd(0,1);
        MaxofMaxAbsAccs(jj,:)=10+unifrnd(0,1);
        MaxofMaxAbsResDrifts(jj,:)=10+unifrnd(0,1);
    end
    if isempty(EQKeepMat)~=1
        for jj=1:numel(EQKeepMat)
            for kk=1:length(Dir)
                if kk==1
                    filename1=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/Drift-
Story1_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj),Dir{kk});
                    filename2=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/Drift-
Story2_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj),Dir{kk});
                    filename3=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/Drift-
Story3_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj),Dir{kk});

                    story1_res=load(filename1);
                    story2_res=load(filename2);
                    story3_res=load(filename3);
                    A=size(story1_res,1);
                    B=size(story2_res,1);
                    C=size(story3_res,1);
                    D=min([A B C]);
                    Data_Drift1=[story1_res(1:D) story2_res(1:D)
story3_res(1:D)]./Unit_Conversion_Drift;
                    Data_Res_Drift1=[abs(story1_res(end))
abs(story2_res(end)) abs(story3_res(end))]./Unit_Conversion_Drift;

                    filename7=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/FloorAccEnv_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj
),Dir{kk});
                    Data_Acc1=[load(filename7)];
                    Data_Acc1=Data_Acc1(end,2:4)./Unit_Conversion_Acc;
                end

                if kk==2
                    filename4=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/Drift-
Story1_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj),Dir{kk});
                    filename5=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/Drift-
Story2_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj),Dir{kk});
                    filename6=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/Drift-
Story3_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj),Dir{kk});

                    story11_res=load(filename4);
                    story22_res=load(filename5);
                    story33_res=load(filename6);

                    AA=size(story11_res,1);
                    BB=size(story22_res,1);
                    CC=size(story33_res,1);
                    DD=min([AA BB CC]);

                    Data_Drift2=[story11_res(1:DD) story22_res(1:DD)
story33_res(1:DD)]./Unit_Conversion_Drift;
                    Data_Res_Drift2=[abs(story11_res(end))
abs(story22_res(end)) abs(story33_res(end))]./Unit_Conversion_Drift;

                    filename8=sprintf('Oak-%d-50-Output-
%d/DriftAcceleration/FloorAccEnv_%d_%s.out',intensity(ii),PopNum,EQKeepMat(jj
),Dir{kk});
                    Data_Acc2=[load(filename8)];
                    Data_Acc2=Data_Acc2(end,2:4)./Unit_Conversion_Acc;
                end
            end

            if  length(Dir)==1
                MaxDrifts=[max(abs(Data_Drift1))];
                MaxAccs=[(abs(Data_Acc1))];
                MaxResDrifts=[(abs(Data_Res_Drift1))];
            elseif length(Dir)==2
                MaxDrifts=max([max(abs(Data_Drift1));max(abs(Data_Drift2))]);
                MaxAccs=max([(abs(Data_Acc1));(abs(Data_Acc2))]);

MaxResDrifts=max([(abs(Data_Res_Drift1));(abs(Data_Res_Drift2))]);
            end
            MaxofMaxAbsDrifts(jj,:)=MaxDrifts;
            MaxofMaxAbsAccs(jj,:)=MaxAccs;
            MaxofMaxAbsResDrifts(jj,:)=MaxResDrifts;
        end
    end
    if Result==1
        AvgofMaxofMaxAbsDriftstot(ii,:)=mean(MaxofMaxAbsDrifts);
        AvgofMaxofMaxAbsAccstot(ii,:)=mean(MaxofMaxAbsAccs);
        AvgofMaxofMaxAbsResDriftstot(ii,:)=mean(MaxofMaxAbsResDrifts);
    elseif Result==2

MedianofMaxofMaxAbsDrifts(ii,:)=[prctile_one(MaxofMaxAbsDrifts(:,1),0.5)
prctile_one(MaxofMaxAbsDrifts(:,2),0.5)
prctile_one(MaxofMaxAbsDrifts(:,3),0.5)];
        MedianofMaxofMaxAbsAccs(ii,:)=[prctile_one(MaxofMaxAbsAccs(:,1),0.5)
prctile_one(MaxofMaxAbsAccs(:,2),0.5) prctile_one(MaxofMaxAbsAccs(:,3),0.5)];

MedianofMaxofMaxAbsResDrifts(ii,:)=[prctile_one(MaxofMaxAbsResDrifts(:,1),0.5
) prctile_one(MaxofMaxAbsResDrifts(:,2),0.5)
prctile_one(MaxofMaxAbsResDrifts(:,3),0.5)];
    elseif Result==3

NinetiethofMaxofMaxAbsDrifts(ii,:)=[prctile_one(MaxofMaxAbsDrifts(:,1),1)
prctile_one(MaxofMaxAbsDrifts(:,1),1) prctile_one(MaxofMaxAbsDrifts(:,3),1)];
        NinetiethofMaxofMaxAbsAccs(ii,:)=[prctile_one(MaxofMaxAbsAccs(:,1),1)
prctile_one(MaxofMaxAbsAccs(:,2),1) prctile_one(MaxofMaxAbsAccs(:,3),1)];

NinetiethofMaxofMaxAbsResDrifts(ii,:)=[prctile_one(MaxofMaxAbsResDrifts(:,1),
1) prctile_one(MaxofMaxAbsResDrifts(:,2),1)
prctile_one(MaxofMaxAbsResDrifts(:,3),1)];
    end
end
%%
y11=AvgofMaxofMaxAbsDriftstot;
y12=MedianofMaxofMaxAbsDrifts;
y13=NinetiethofMaxofMaxAbsDrifts;

y21=AvgofMaxofMaxAbsAccstot;
y22=MedianofMaxofMaxAbsAccs;
y23=NinetiethofMaxofMaxAbsAccs;

y31=AvgofMaxofMaxAbsResDriftstot;
y32=MedianofMaxofMaxAbsResDrifts;
y33=NinetiethofMaxofMaxAbsResDrifts;

if Drift_Constraint=='None'
    Drift_Constraint=100;
end
if Acc_Constraint=='None'
    Acc_Constraint=100;
end
if Res_Drift_Constraint=='None'
    Res_Drift_Constraint=100;
end
Stories=3;
if Result==1
```

```
    y1=y11;
    y2=y21;
    y3=y31;
    if Drift_Constraint_method==1 || Acc_Constraint_method==1 ||
Res_Drift_Constraint_method==1
        for i=1:Stories
            if y11(1,i) > Drift_Constraint || y21(1,i) > Acc_Constraint ||
y31(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end

    if Drift_Constraint_method==2 || Acc_Constraint_method==2 ||
Res_Drift_Constraint_method==2
        for i=1:Stories
            if  y12(1,i) > Drift_Constraint || y22(1,i) > Acc_Constraint ||
y32(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end

    if Drift_Constraint_method==3 || Acc_Constraint_method==3 ||
Res_Drift_Constraint_method==3
        for i=1:Stories
            if  y13(1,i) > Drift_Constraint || y23(1,i) > Acc_Constraint ||
y33(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end
end

if Result==2
    y1=y12;
    y2=y22;
    y3=y32;
    if Drift_Constraint_method==1 || Acc_Constraint_method==1 ||
Res_Drift_Constraint_method==1
        for i=1:Stories
            if  y11(1,i) > Drift_Constraint || y21(1,i) > Acc_Constraint ||
y31(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end
    if Drift_Constraint_method==2 || Acc_Constraint_method==2 ||
Res_Drift_Constraint_method==2
        for i=1:Stories
            if  y12(1,i) > Drift_Constraint || y22(1,i) > Acc_Constraint ||
y32(1,i) > Res_Drift_Constraint
```

```
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end

    if Drift_Constraint_method==3 || Acc_Constraint_method==3 ||
Res_Drift_Constraint_method==3
        for i=1:Stories
            if  y13(1,i) > Drift_Constraint || y23(1,i) > Acc_Constraint ||
y33(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end
end
if Result==3
    y1=y13;
    y2=y23;
    y3=y33;
    if Drift_Constraint_method==1 || Acc_Constraint_method==1 ||
Res_Drift_Constraint_method==1
        for i=1:Stories
            if  y11(1,i) > Drift_Constraint || y21(1,i) > Acc_Constraint ||
y31(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end
    if Drift_Constraint_method==2 || Acc_Constraint_method==2 ||
Res_Drift_Constraint_method==2
        for i=1:Stories
            if  y12(1,i) > Drift_Constraint || y22(1,i) > Acc_Constraint ||
y32(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end

    if Drift_Constraint_method==3 || Acc_Constraint_method==3 ||
Res_Drift_Constraint_method==3
        for i=1:Stories
            if  y13(1,i) > Drift_Constraint || y23(1,i) > Acc_Constraint ||
y33(1,i) > Res_Drift_Constraint
                y1(1,i)=10+unifrnd(0,1); y2(1,i)=10+unifrnd(0,1);
y3(1,i)=10+unifrnd(0,1);
            end
        end
    end
end
y=[y1;y2;y3];
```

REFERENCES

Akcelyan, S., and Lignos, D. (2018). “Seismic retrofit of steel tall buildings with bilinear oil dampers.” *16ECEE*, Thessaloniki, Greece, 1–11.

Altieri, D., Tubaldi, E., De Angelis, M., Patelli, E., and Dall’Asta, A. (2018). “Reliability-based optimal design of nonlinear viscous dampers for the seismic protection of structural systems.” *B Earthq Eng*, 16(2), 963–982.

ASCE (American Society of Civil Engineers) (2013). *Seismic Evaluation and Retrofit of Existing Buildings*, ASCE 41-13, Reston, VA.

Castaldo, P., and De Iuliis, M. (2014). “Optimal integrated seismic design of structural and viscoelastic bracing-damper systems.” *Earthq Eng Struct D*, 43(12), 1809–1827.

Coello, C. C., and Lechuga, M. S. (2002). “MOPSO: A proposal for multiple objective particle swarm optimization.” *Proc. CEC '02*, IEEE, Honolulu, HI, USA, 1051–1056.

CEN (Comité Européen de Normalisation/European Committee for Standardization). (2005). Eurocode 8 Part 1: General Rules, Seismic Actions and Rules for Buildings. CEN: Brussels, Belgium.

Constantinou, M. C., and Symans, M. D. (1993). “Seismic response of structures with supplemental damping.” *Struct Des Tall Spec*, 2(2), 77–92.

Corne, D. W., Jerram, N. R., Knowles, J. D., and Oates, M. J. (2001). “PESA-II: Region-based selection in evolutionary multiobjective optimization.” *GECCO'01*, Morgan Kaufmann Publishers, San Francisco, CA, USA, 283–290.

Deb, K., and Jain, H. (2013). “An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints.” *IEEE T Evolut Comput*, 18(4), 577–601.

Del Gobbo, G. M., Williams, M. S., and Blakeborough, A. (2018). “Comparing fluid viscous damper placement methods considering total-building seismic performance.” *Earthq Eng Struct D*, 47(14), 2864–2886.

Fallah-Mehdipour, E., Haddad, O. B., and Mariño, M. A. (2011). “MOPSO algorithm and its application in multipurpose multireservoir operations.” *J Hydroinform*, 13(4), 794 811.

Federal Emergency Management Agency (2012). “Seismic performance assessment of buildings.” *FEMA P-58*, Washington DC, USA.

Filiatrault, A., and Christopoulos, C. (2006). *Principles of passive supplemental damping and seismic isolation*, IUSS Press, Pavia, Italy.

Fonseca, C. M., and Fleming, P. J. (1993). “Genetic algorithms for multiobjective optimization: formulation discussion and generalization.” *Proc. of the 5th Int. Conf. on Genetic Algorithms, Morgan Kaufman Publishers*, San Francisco, 416-423.

Holland, J.H. (1975). *Adaptation in natural and artificial systems,* University of Michigan Press, Ann Arbor, Michigan; re-issued by MIT Press (1992).

Kennedy, J., and Eberhart, R. (1995). “Particle swarm optimization (PSO).” *Proc. IEEE International Conference on Neural Networks*, Vol. 4, Perth, Australia, 1942–1948.

Knowles, J. D., and Corne, D. W. (2000). “Approximating the nondominated front using the Pareto archived evolution strategy.” *Evol. Comput*, 8(2), 149–172.

Lalwani, S., Singhal, S., Kumar, R., and Gupta, N. (2013). “A comprehensive survey: Applications of multi-objective particle swarm optimization (MOPSO) algorithm.” *Trans. comb.*, 2(1), 39–101.

McKenna, F., and Fenves, G. L. (2004). Open system for earthquake engineering simulation OpenSEES. Pacific Earthquake Engineering Research Center, University of California: Berkeley, CA.

Narkhede, D. I., and Sinha, R. (2014). “Behavior of nonlinear fluid viscous dampers for control of shock vibrations.” *J Sound Vib*, 333(1), 80–98.

Pollini, N., Lavan, O., and Amir, O. (2016). “Towards realistic minimum-cost optimization of viscous fluid dampers for seismic retrofitting.” *B Earthq Eng*, 14(3), 971–998.

Terzic, V., and Mahin, S. A. (2017). “Using PBEE to assess and improve performance of different structural systems for low-rise steel buildings.” *Int. J. of Safety and Security Eng.*, 7(4), 532–544.

Thomas, M. (2018). “Computational efficiency of multi-objective particle swarm optimization considering a single degree of freedom system under earthquake loading.” B.S. thesis, California State University, Long Beach.

Vedarajan, G., Chan, L. C., and Goldberg, D. E. (1997). “Investment portfolio optimization using genetic algorithms.” *In Late breaking papers at GP-97*, Stanford University, California: Stanford Bookstore, Stanford, CA, 255–263.

Wang, S., and Mahin, S. A. (2018a). “High-performance computer-aided optimization of viscous dampers for improving the seismic performance of a tall building.” *Soil Dyn Earthq Eng.*, 113, 454–461.

Wang, S., and Mahin, S. A. (2018b). “Seismic upgrade of an existing tall building using different supplemental energy dissipation devices.” *J. Struct. Eng.*, 144(7), 04018091.

Zhang, Q., and Li, H. (2007). “MOEA/D: A multiobjective evolutionary algorithm based on decomposition.” *IEEE Evolut Comput.*, 11(6), 712–731.

Zitzler, E., Laumanns, M., and Thiele, L. (2001). “SPEA2: Improving the strength Pareto evolutionary algorithm.” *TIK-report 103*, Zurich, Switzerland.