Summer 2017

Efficient Algorithms for Solving Size-Shape-Topology Truss Optimization and Shortest Path Problems

Gelareh B. Sanjabi
Old Dominion University, gbakh001@odu.edu

EFFICIENT ALGORITHMS FOR SOLVING SIZE-SHAPE-TOPOLOGY TRUSS OPTIMIZATION AND SHORTEST PATH PROBLEMS

by

Gelareh B. Sanjabi
B.S. August 2003, Razi University, Iran
M.S. August 2013, Old Dominion University

A Dissertation Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

CIVIL ENGINEERING

OLD DOMINION UNIVERSITY
August 2017

Approved by:

Duc T. Nguyen (Co-Director)

Manwo Ng (Co-Director)

Mecit Cetin (Member)

Gene Hou (Member)

ABSTRACT

EFFICIENT ALGORITHMS FOR SOLVING SIZE-SHAPE-TOPOLOGY TRUSS OPTIMIZATION AND SHORTEST PATH PROBLEMS

Gelareh B. Sanjabi
Old Dominion University, 2017
Co-Directors: Dr. Duc T. Nguyen
Dr. ManWo Ng

Efficient numerical algorithms for solving structural and Shortest Path (SP) problems are proposed and explained in this study. A variant of the Differential Evolution (DE) algorithm for optimal (minimum) design of 2-D and 3-D truss structures is proposed. This proposed DE algorithm can handle size-shape-topology structural optimization. The design variables can be mixed continuous, integer/or discrete values. Constraints are nodal displacement, element stresses and buckling limitations.

For dynamic (time dependent) networks, two additional algorithms are also proposed in this study. A heuristic algorithm to find the departure time (at a specified source node) for a given (or specified) arrival time (at a specified destination node) of a given dynamic network. Finally, an efficient bidirectional Dijkstra shortest path (SP) heuristic algorithm is also proposed. Extensive numerical examples have been conducted in this study to validate the effectiveness and the robustness of the proposed three numerical algorithms.

I dedicate this thesis to my wonderful family. First, I must thank my loving mother who has given me her fullest support. My sister has never left my side and is very special. Many thanks to my understanding and patient husband who has supported me throughout the process. I also dedicate this thesis to my brothers, my nephew, and my nieces, I am grateful for your heartwarming support. Finally, I dedicate this work to my late father, who believed in the pursuit of academic excellence.

ACKNOWLEDGMENTS

My special thanks to Dr. Nguyen for providing me with invaluable guidance and encouragement throughout my graduate studies. I greatly appreciate his confidence in my work and his scholarly advice. His patience and encouragement have played a fundamental role in the completion of this work in the present form. I am grateful to him as he supported me during tough times and helped me through the process.

Many thanks to Dr. Ng for providing me with insight and helping me stay encouraged about my research.

I would like to thank Dr. Cetin for his support, interest, and time serving on my dissertation committee.

I would also like to thank Dr. Hou for the time serving on my dissertation committee and for his scholarly advice and encouragement.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

Page

LIST OF FIGURES

Page

CHAPTER 1

INTRODUCTION

A broad class of man-made structures are made of trusses such as bridges, towers, cranes, and roof support trusses. The individual elements of truss are generally rod elements (bars) which only carry axial forces. The rod elements are connected at two end nodes; the common connection type is pinned connections.

The ubiquity of the truss structures in industrial world is because of their simple and functional construction. However, the structures made from trusses can be very complex and difficult to model. As a result, the usage of the modern design optimization tools is necessary to achieve competitive and economic designs based on the basic design standards. The optimal design of truss structures can be divided in three types based on the category of the selected design variables. The first category of design variables is the truss size so that one can find the optimal cross sections of truss elements. The second category is the optimization of the parameters defining the shape of the structure. More specifically, the optimum location of the selected joints in the structure are determined. The last category is the optimization of topology variables in order to find the optimum number of required members in the structure. The objective of the optimization is to minimize the weight of the structure for a given loading condition subjected to the limitations of element stresses and buckling as well as nodal displacements. Size variables are the dimensions of the member's cross sectional area. They are selected from a list of standard profiles which means the size variables are treated as discrete variables. Shape variables which represent nodal coordinates are selected from a continuous space. This optimization is challenging because it is necessary to treat the continuous variables and discrete variables together for solving shape, size,

and topology simultaneously. Various evolutionary algorithm such as Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), Harmony Search (HS), Firefly Algorithm (FA) and Differential Evolution (DE) have been used for solving the mentioned problems. DE has shown superiority when compared with other evolutionary algorithms. It is also considered a powerful and reliable method for structural optimizations. DE is more successful in finding the global minimum with far less sensitivity to the selection of the initial guess. DE can converge fast and has far less tunable parameters when compared with GA (Ho-Huu et al., 2015). This reliable and versatile optimizer is a population based algorithm like other evolutionary algorithms which uses processes inspired by biological evolution of the nature, such as mutation, crossover, and selection.

Candidate solutions of the optimization problem are individuals in the population. The mutation and crossover help to improve the diversity of the population, while the selection process helps to explore the better candidates in the search domain. The original DE algorithm proposed by Storn and Price in 1995 was one of the most successful method for solving continuous optimization problems. The original DE algorithm provides a fast method of achieving the global minimum. In the past decades, various methods were utilized to improve the original DE algorithm. DE applications were also expanded to solve broader range of optimization problems. There was no constraint handling capability in the original DE algorithm. The improved DE algorithm was combined with multiple constraint handling methods to overcome this shortcoming. Notably the DE algorithm combined with the Deb's constraint handling method has provided the required flexibility to handle complex constraints within the DE context. Other improvements include adding the generalized rule in order to solve problems with discrete variables (Lampinen

& Zelinka, 1999), and addition of various mutation and crossover methods to increasing the diversity of population.

Truss optimization is considered a constrained optimization problem with mixed discrete-continuous design variables. In this study, several strategies were utilized to improve the state of the art DE algorithms available in literature for this application. The capability of DE algorithm depends significantly on the selected strategies for mutation, crossover, and selection operations as well as the value of DE control parameters.

In this study, the mutation and crossover strategies were modified to enhance the Improved $(\mu + \lambda)$-Differential Evolution (IDE) method (Jia et al., 2013). Three mutation and crossover strategies were used in IDE to generate the offspring population. In the proposed algorithm, one of the IDE mutation strategies (DE/rand/1/bin) was replaced by a combination of strategies originally used in unconstrained optimization (Mohamed et al., 2012). More specifically, "DE/rand/1" and a directed mutation strategy defined based on the weighted difference vector between the best and the worst individuals of a generation were combined and replaced (DE/rand/1/bin) in IDE. In the proposed algorithm, the scaling factors of the combined mutation strategy was calculated by random and ranked based methods. Also, the crossover rate of this strategy was obtained by a dynamic nonlinearly increasing probability. Moreover, the third crossover strategy of IDE was modified in the proposed algorithm to maintain the diversity of the population. Additionally, an Improved Adaptive Tradeoff Model (IATM) was used for selection phase of the algorithm (Wang & Cai, 2011). To be able to handle discrete variables a generalized method proposed by Lampinen & Zelinka in 1999 was adopted in this study.

Finally, the proposed algorithm was evaluated by solving several well-known benchmark problems. The numerical result obtained using the proposed differential evolution algorithm

outperformed the existing methods found in literature both in terms of finding the final optimal solution and the convergence rate. The proposed improved DE algorithm is further explained in Chapter 2.

The Shortest Path Problem (SPP) on static graphs has been one of the most studied problems in recent years, because of its practical application related to network problems. Although static shortest path algorithms play an essential role in problems which are not changing over time, recently more focus have been moved toward Time Dependent Shortest Path Problems (TDSPP). TDSPP is the SPP in which the cost of edges can vary as a function of time. For example in a road network, the shortest path from a specified source node to a destination node during low traffic periods is not the same as during rush over. The time and cost of travel are important factors from travel forecasting outlook. In the time dependent network, the travel time along each arc is treated as a function of the departure time along the arc. These functions are known for all times in advance. There are various application for TDSPP; network control, automobile driver guidance, ship routing, and dynamic traffic assignment are the most typical applications of TDSPP.

In part of Dynamic Traffic Assignment (DTA) problems, the goal is to find the earliest arrival time at a destination(s) from a specified source node at a specified departure time. This problem is called the backward shortest path problem, which is examined in this work. The link travel cost is defined as a piece-wise linear function. The network is assumed to be First In, First Out (FIFO) and non-FIFO. The implemented heuristic backward Dijkstra algorithm has been tested through some small and real life networks and their results are compared with the forward Dijkstra algorithm to show the accuracy of the resultant shortest path. Chapter 3 is devoted to describe the mentioned backward Dijkstra algorithm for finding the departure time.

Finding the shortest path from a source to a destination over a time dependent network is the essential problem in DTA and numerous other applications. This problem can be computed using a Dijkstra algorithm, but this may not be fast enough for useful applications. Therefore, in Chapter 4, a new heuristic time dependent bidirectional search algorithm is proposed which can find point to point shortest path and arrival time. This proposed algorithm is based on Dijkstra algorithm. The proposed Time Delay Factor method is combined with a piece-wise linear function to describe the link cost as a function of time. The backward Dijkstra SP algorithm, developed in Chapter 3, is aslo used for the backward search of the proposed bidirectional algorithm in Chapter 4. This algorithm is explained via a simple and small network. Then, its application is expanded to analyze real life networks and evaluated by several numerical examples. The results are compared with Dijkstra algorithm. The performance of the above mentioned algorithms in finding the shortest path, arrival time, and the computational cost are compared. The number of explored nodes is reduced in comparison with the classical time dependent forward Dijkstra algorithm.

CHAPTER 2

SIZE-SHAPE-TOPOLOGY OPTIMIZATION OF TRUSS USING DEFFERENTIAL EVOLUTION ALGORITHM

2.1 INTRODUCTION

The truss optimization problem is considered one of the challenging and practical engineering problems because it deals with continuous and discrete variables. In the last decade, various evolutionary algorithms have been implemented for structural optimization which are shown to be both robust and reliable computational tools in comparison with the conventional gradient-based methods.

This field of structural optimization is divided into three categories based on the selected design variables to optimize the size, shape, and topology of the structure. It was proven that by considering size and shape or size, shape, and topology variables simultaneously, the optimization problem can find enhanced optimum design. Hence, the obtained optimum design will save more material resulting in lighter structures when compared to pure size optimization methods (Gholizadeh, 2013). The difficulty of the problem will increase by including discrete design variables to the existing continuous ones. The necessity to include discrete design variables is caused by the existing limitations of the manufactured standard profiles and the related cost savings when off the shelf items are used. As a result, the designer is limited to use the available cross sectional areas in the manufacturer's catalogue.

Several metaheuristic algorithms, which are generated in the context of evolutionary algorithms, have been implemented to handle the mixed discrete-continuous difficulties in solving truss optimization problems. The popular methods in this field include Genetic Algorithm (GA) (Rajeev & Krishnamoorthy, 1992; Rajan, 1995; Ruiyi et al., 2009; Kaveh & Kalatjari, 2004; Balling et al., 2006; Tang et al., 2005), Ant Colony Optimization (ACO) (Camp & Bichon, 2004), Harmony Search (HS) (Lee et al., 2005), Evolutionary Strategy (ES) (Chen & Chen, 2008), Particle Swarm Optimization (PSO) (Kaveh & Talatahari, 2009), Firefly Algorithm (FA) (Gandomi et al., 2011; Miguel et al., 2013), Search Group Algorithm (SGA) (Goncalves et al., 2015), Differential Evolution (DE) (Wu & Tseng, 2009; 2010; Wang et al., 2009; Ho-Huu et al., 2015), etc.

Among many metaheuristic algorithm, Differential Evolution (DE) is a simple, robust, and reliable method (Storn & Price, 1995). It has been proven that DE is a powerful method for solving various optimization problems in science and technology (Das & Suganthan, 2011). DE is a population based method and follow the general procedures of evolutionary algorithms (EA).There are four basic operators in DE: Initialization, Mutation, Crossover, and Selection. Mutation and Crossover operators both diversify the population. Selection helps the progress of exploitation in finding better candidates in the search domain.

Since 1995, the DE algorithm has been progressively improved and established promising result in solving complex Constrained Optimization Problems (COPs). For instance, Mallipeddi and Suganthan (2010) proposed a DE algorithm with an Ensemble of Constraint Handling Techniques (ECHT) in which each population associates with its own constraint handling method. Liao (2010) presented two hybrid DE algorithms. One of them improved a basic DE algorithm with a local search operator, and the second one used a Harmony Search (HS) to comply with the

DE algorithm in order to find improved cooperative result. Mohamed and Sabry (2012) implemented a modified DE algorithm which is different in mutation method, control parameters, and constraint handling policy. Wang and Cia (2011) introduced $(\mu + \lambda)$ DE with an Improved Adaptive Trade-off Method (IATM) to solve COPs efficiently. In this method, each individual in the population produces three offspring by using three different mutation strategies and the next generation can be selected among the combined populations of parent and offspring. Then, Jia et al. (2013) improved this method by introducing a new mutation method and modified the constraint handling method in order to promote diversity and convergence of the population and called the method Improved Constrained Differential Evolution (ICDE). Wang et al. (2012) proposed a DE algorithm with a new crossover method called orthogonal crossover (OX) and claimed that this new method which is based on orthogonal design, can make a systematic and rational search in parent population. A hybrid version of DE with two differential mutation to increase the population diversity through the evolution was presented by Hernandez et al. (2013). Another algorithm proposed by Cui et al. (2016) named MPADE, adaptive differential evolution algorithm with novel mutation strategies in multiple sub-populations. They split the parent population into three sub-populations based on the value of objective function then three different DE strategies are applied on each sub-population to improve the exploitation and exploration.

Several DE algorithm have been proposed in the literature for solving truss optimization problems. A combined heuristic optimization method which is a combination of the threshold accepting algorithm with differential evolution proposed by Schmidt and Thierauf (2005). The aforementioned algorithm was designed for solving mixed discrete-continuous variables with emphasis on structural optimization. A novel DE for solving truss structures with both continuous and discrete variables was presented by Wang et al. (2009). Wu and Tseng (2010) applied Multi-

Population Differential Evolution (MPDE) with a penalty based, self-adaptive strategy for adjusting control parameters of the algorithm. They showed that the self-adaptive strategy improved the performance of MPDE especially for solving constrained truss optimization problems. Krempser et al. (2012) presented a differential evolution assisted by surrogate models. The surrogate model selects the best offspring found by different mutation strategies. Ho-Huu et al. (2015) modified ICDE (Jia et al., 2013) to handle discrete and continuous design variables for solving truss size and shape optimization. A generalized method for solving mixed integer-discrete-continuous optimization which was proposed by Lampinen and Zelinka (1999) is used to implement discrete-ICDE (D-ICDE). This method also used by Schmidt and Thierauf (2005) to handle discrete variables.

Pham (2016) introduced a discrete optimal sizing of truss using adaptive directional differential evolution (ADDE). He used a new self-adaption approach and a simple directional strategy to balance global exploration and local exploitation for promoting the final solution. Generally, the main goals of researchers are to increase the diversity of the population, balance exploration and exploitation, and improve the constraint handling ability of the DE.

A new variant of DE is proposed in this work to enhance the performance of differential evolution in size, shape, and topology optimization of trusses. The Improved ($\mu+\lambda$) Differential Evolution (IDE) method (Jia et al., 2013) generates the offspring population using three different mutation strategies. The first mutation strategy (DE/rand/1/bin) maintains the population diversity and has good global search capability. However, its local search capability and the resulting convergence rate are sub-optimal. In this work, a mutation strategy which was first introduced in Alternative Differential Evolution (ADE) algorithm (Mohamed et al., 2012) was adopted to overcome the aforementioned shortcomings in original IDE. More specifically, in the proposed

algorithm, the first IDE mutation strategy was replaced by a combination of "DE/rand/1" and "directed mutation" strategies through a linearly decreasing probability rule. The directed mutation strategy is a function of weighted difference between the best and the worst individuals of the current population.

In the proposed algorithm, a binomial crossover was used to generate the offspring population of this combined mutation strategy. Additionally, the crossover rate was calculated by a dynamic nonlinearly decreasing probability scheme similar to ADE (Mohamed et al., 2012). The other unique feature of the proposed algorithm is the method used to calculate the scaling factor of the directed mutation. This scaling factor was calculated by a proposed ranked based method which is a modified version of the method used in Individual Dependent Mutation (IDM) strategy (Tang et al., 2014). Unlike IDE, the scaling factors of "DE/rand/1" was obtained randomly. In the proposed algorithm, the selected scaling factor and crossover rates are unique to each strategy while in original IDE fixed values are used for scaling factor and the crossover rate in all strategies.

The third mutation in original IDE method is called "DE/current-to-rand/best/1" strategy, where no crossover operation was applied to generate offspring population of "DE/current-to-rand/1" mutation strategy. The diversity of the generated population in the proposed algorithm was improved by utilizing a binomial crossover method. The method used for selection operation can significantly affect the performance of the DE optimization algorithm. The Improved Adaptive Tradeoff Model (IATM), which was used in solving constrained optimization problems (Wang & Cai, 2011) was adopted as the selection operation method in the proposed algorithm.

Hence, both the local search capability and the convergence rate of the proposed algorithm for solving truss optimization problems are improved. As a result, the number of required

evaluations of the objective function and constraints are considerably reduced which translates into significant reduction of computational cost.

The presented method is applied to truss optimization problem with stress, displacement, and buckling constraints. Several numerical examples are solved and their result compared to the previous state-of-the-art methods to examine the performance and efficiency of the proposed method.

This chapter is organized as follows. The basic concept of truss optimization problems presented in Section 2.2. Section 2.3 contains a concise review of the original Differential Evolution. The proposed algorithm is explained in Section 2.4. The numerical implementation and conclusion are presented in Section 2.5 and Section 2.6, respectively.

2.2 BASIC CONCEPT OF TRUSS OPTIMIZATION

The truss optimization problem can be formulated as below:

Minimize
$$f(X) = \sum_{m=1}^{s} \rho_m x_m l_m \tag{1}$$

Subjected to :
$$\Delta(X) \leq \Delta_{allowable} \tag{2}$$
$$\sigma(X) \leq \sigma_{allowable} \tag{3}$$
$$\lambda(X) \leq \lambda_{allowable} \tag{4}$$
$$X = \{x_m^l \leq x_m \leq x_m^u\}, m = 1,2, \dots, s, \dots, D \tag{5}$$

where $f(X)$ is the objective function representing the structural weight of the truss in which ρ_m is the material density and l_m is the length of the mth member. X is the design variables vector which contains the size and shape variables of the truss elements. D is the number of design variables. $\Delta(X), \sigma(X),$ and $\lambda(X)$ represents nodal displacement, element stress, and buckling stress,

respectively, and all of them are determined within their allowable values. x_m is the mth design variable defined within the lower bound x_m^l and upper bound x_m^u, $m = 1,2, \dots, s$ are indices related to the area design variables and $m = s+1, \dots, D$ are indices related to the nodal coordinates.

The above optimization problem can be reformulated as:

Minimize
$$f(X) = \sum_{m=1}^{s} \rho_m x_m l_m \tag{6}$$

Subjected to :
$$\frac{\Delta(X)}{\Delta_{allowable}} - 1 \leq 0 \tag{7}$$

$$\frac{\sigma(X)}{\sigma_{allowable}} - 1 \leq 0 \tag{8}$$

$$\frac{\lambda(X)}{\lambda_{allowable}} - 1 \leq 0 \tag{9}$$

$$X = \{x_m^l \leq x_m \leq x_m^u\}, m = 1,2, \dots, s, \dots, D \tag{10}$$

When we have an inequality constraint like $g_k(X) \leq 0$, the constraint violation, $G_k(X)$, of a design variable or an individual X on the kth constraint is calculated by

$$G_k(X) = \max\{0, g_k(X)\}, k = 1, \dots, P \tag{11}$$

where P represents the number of constraints in the optimization problem and $g_k(X)$ is the value of kth constraint for individual X.

2.3 REVIEW OF DIFFERENTIAL EVOULOTION

2.3.1 BASIC DIFFERENTIAL EVOLUTION

Differential evolution is an efficient evolutionary algorithm for solving global optimization problem. This method was first proposed by Storn and Price in 1995. The four basic operations in DE are explained below.

2.3.1.1 INITIALIZATION

An initial population must be generated to stablish a starting point for the optimization process. The initial population contains NP individuals hence NP represents number of population. The population individuals are randomly sampled values selected within the range between the lower and upper limit of each design variable. This sampled population can be generated using Equation (12).

$$x_{ij}^{0} = x_{Lj} + rand_{ij}\left(x_{Uj} - x_{Lj}\right) \quad , i = 1,2, \dots, NP, j = 1,2, \dots, D \tag{12}$$

where x_{ij}^{0} is the initial population in which the super script, 0, is corresponding to the generation number. The subscript ij corresponds to the ith population, and the jth decision variable, respectively. Equation (12) guarantees that the lower and upper bound constraints on decision variables are satisfied. The $rand_{ij}$ denotes a uniformly distributed number between$[0,1]$, which is generating a new value for each decision variables. The lower bound for the jth decision variable is denoted by x_{Lj} and x_{Uj} stands for upper bound for the jth decision variable. D is the total number of decision variables.

2.3.1.2 MUTATION

DE generates a mutant (donor) vector v_i^{g+1} for each target vector x_i^g at each iteration through mutation operation. The superscript of the mutant and target vectors stands for generation number and the subscript i represents the ith population. The mutation strategy of DE generally named as "DE/x/y/z", where x denotes the *basic* vector to be perturb, y represents the *number of difference* vectors considered for perturbation, and z is the *crossover* method being used (exp: exponential; bin: binomial) (Das et al., 2016). Note that when the binomial crossover is used the mutation strategy can be named as "DE/x/y". Six most frequently used mutation strategies (Das et al., 2016) are described as follows:

DE/rand/1: $$v_i^{g+1} = x_{r1}^g + F.(x_{r2}^g - x_{r3}^g) \quad (13)$$

DE/rand/2: $$v_i^{g+1} = x_{r1}^g + F.\left(x_{r2}^g - x_{r3}^g\right) + F.(x_{r4}^g - x_{r5}^g) \quad (14)$$

DE/best/1: $$v_i^{g+1} = x_{best}^g + F.(x_{r1}^g - x_{r2}^g) \quad (15)$$

DE/best/2: $$v_i^{g+1} = x_{best}^g + F.\left(x_{r1}^g - x_{r2}^g\right) + F.(x_{r3}^g - x_{r4}^g) \quad (16)$$

DE/current-to-rand/1:
$$\begin{aligned} v_i^{g+1} &= x_i^g + F.\left(x_{r1}^g - x_i^g\right) + F.\left(x_{r2}^g - x_{r3}^g\right) \\ &= (1-F).x_i^g + F.x_{r1}^g + F.\left(x_{r2}^g - x_{r3}^g\right) \end{aligned} \quad (17)$$

DE/current-to-best/1:
$$\begin{aligned} v_i^{g+1} &= x_i^g + F.\left(x_{best}^g - x_i^g\right) + F.(x_{r1}^g - x_{r2}^g) \\ &= (1-F).x_i^g + F.x_{best}^g + F.\left(x_{r1}^g - x_{r2}^g\right) \end{aligned} \quad (18)$$

In Equations (13) to (18), $r1, r2, r3, r4$, and $r5$ are distinct integers randomly chosen from the range $[1,\ NP]$, and all are different from the base index i. These indices are randomly generated for each mutant vector. x_{best}^g is the best individual vector with the best objective function value of the current generation. The scaling factor, which is used to control the mutation scale, is denoted by F and controls the amplification of the differential variation and is restricted in $(0,1]$.

Note that in Equation (17), "DE/current-to-rand/1" strategy, the vector which generated with the scaled difference of the two other population members is a convex combination of the current target vector and another random member of the population for $F < 1$ (Das et al., 2016).

Since $1 - F + F = 1$, for $F, 1 - F \geq 0$, the $(1 - F)x_i^g + F.x_{r1}^g$ is a convex combination. This means the base vector for mutation is an arithmetic recombination between x_i^g and x_{r1}^gbecause it denotes a point on the line joining the target vector and a random chosen population member. Therefore, the resulting mutant vector is considered as a mutated recombinant. Similarly, the same argument is true for Equation (18).

2.3.1.3 CROSSOVER

To enhance the diversity of the perturbed mutant vectors, crossover is introduced in the DE algorithm. During crossover, the mutant vector, v_i^{g+1}, mixes its components with the target vector, x_i^g, to form the trial/offspring vector, u_i^{g+1}. Commonly two crossover methods are utilized in DE; binomial (uniform) method and exponential (or two point modulo) method. The most common method is binomial crossover, in which the number of components inherited from the mutant vector follow a nearly binomial distribution. This method can be formulate as:

$$u_{ij}^{g+1} = \begin{cases} v_{ij}^{g+1} & \text{if } j = k \text{ or } rand_{ij} \leq Cr \\ x_{ij}^g & \text{otherwise} \end{cases} \quad (19)$$

where $rand_{ij}$ is a uniform random number in $[0,1]$ range which is generated once for every component of each vector per iteration. Also, k is any randomly chosen natural number in $\{1,2,3,\dots,D\}$, D is the total number of decision variables, this random value ensures that trial vector, u_i^{g+1}, gets at least one element from mutant vector, v_i^{g+1}. Furthermore, Cr is the Crossover

rate which has to be determined by user. It controls how many components of trial vector, u_i^{g+1}, are inherited from the mutant vector, v_i^{g+1}.

2.3.1.4 SELECTION

After crossover, DE evaluates both the objective function and constraints for all the trial/offspring vector, u_i^{g+1}, in the population. Selection determines whether the target (parent) vector, x_i^g, or the trial (offspring) vector, u_i^{g+1}, survives to the next iteration (i.e., $g+1$). The original DE proposed by Storn and Price (1995) was implemented for unconstraint optimization problems and always the vector which yields a smaller cost function (for minimization problem) will survive to the next generation. Thus, each individual of the trial population is compared with its counterpart in the current population, following Equation (20).

$$x_{\boldsymbol{i}}^{\boldsymbol{g+1}} = \begin{cases} u_{\boldsymbol{i}}^{\boldsymbol{g+1}} & \text{if } f\left(u_{\boldsymbol{i}}^{\boldsymbol{g+1}}\right) \le f\left(x_{\boldsymbol{i}}^{\boldsymbol{g}}\right) \\ x_i^g & \text{otherwise} \end{cases} \tag{20}$$

Note that the initialized vector, x_i^0, is calculated by Equation (12) (i.e., $x_{ij}^0 = x_{Lj} + rand_{ij}(x_{Uj} - x_{Lj})$), also the mutant/donor vector, v_i^{g+1}, is obtained using strategies mentioned in Equations (13) to (18). The trial/offspring vector, $u_{\boldsymbol{i}}^{\boldsymbol{g+1}}$, is generated by crossover method stated in Equation (19), and the target vector, $x_{\boldsymbol{i}}^{\boldsymbol{g+1}}$, is obtained by the selection operation.

2.3.2 CONSTRAINT HANDLING

In this section, the added constraint handling methods to DE algorithm are explained.

2.3.2.1 BOUNDARY (LOWER OR UPPER BOUND) CONSTRAINTS

It is important to make sure that the value of generated vectors lie within their allowed range after reproduction. After mutation, one should check the lower and upper bounds for the

mutant vectors. Four different techniques were studied in this work to make the solutions feasible and repair the variables violating their given lower and upper bounds as explained below.

I. Setting the decision variables to the violated bound (Price, 1999) by enforcing Equation (21).

$$v_{ij}^{g+1} = \begin{cases} x_{Lj} & \text{, if } \quad v_{ij}^{g+1} \leq x_{Lj} \\ x_{Uj} & \text{, if } \quad v_{ij}^{g+1} \geq x_{Uj} \end{cases} \tag{21}$$

II. Randomly reinitiate the considered decision variable within its allowable bounds (use Equation (12)) (Price, 1999).

III. Setting the decision variable midway between its parent initial value x_{ij}^{g} and the violated bound (i.e., x_{Lj} and x_{Uj}) (Price, 1999).

$$v_{ij}^{g+1} = \begin{cases} \dfrac{x_{ij}^{g} + x_{Lj}}{2} & \text{if } \quad v_{ij}^{g+1} \leq x_{Lj} \\ \dfrac{x_{ij}^{g} + x_{Uj}}{2} & \text{if } \quad v_{ij}^{g+1} \geq x_{Uj} \end{cases} \tag{22}$$

IV. Use the violated bound (i.e., x_{Lj} and x_{Uj}) is used as a symmetry center for sending the considered variable to the feasible side of the boundary. The distance to the violated bound is equal to the initial constraint violation in this approach which is further explained in Equation (23) (Mezura-Montez et al., 2006).

$$v_{ij}^{g+1} = \begin{cases} 2x_{Lj} - v_{ij}^{g+1} & \text{, if } \quad v_{ij}^{g+1} \leq x_{Lj} \\ 2x_{Uj} - v_{ij}^{g+1} & \text{, if } \quad v_{ij}^{g+1} \geq x_{Uj} \end{cases} \tag{23}$$

Say $v_{ij}^{g+1} = \alpha_L x_{Lj}$, when $\alpha_L \in [0,1]$, then $2x_{Lj} - v_{ij}^{g+1} = 2x_{Lj} - \alpha_L x_{Lj} = (2 - \alpha_L).x_{Lj} \geq x_{Lj}$. Say $v_{ij}^{g+1} = (\alpha_U + 1).x_{Uj}$, when $\alpha_U \in [0,1]$, then $2x_{Uj} - v_{ij}^{g+1} = 2x_{Uj} - (\alpha_U + 1).x_{Uj} = (1 - \alpha_U).x_{Uj} \leq x_{Uj}$.

2.3.2.2 CONSTRAINT FUNCTIONS

The initial DE algorithm was proposed to solve unconstrained optimization problems. Numerous methods were proposed to add constraint handling to evolutionary optimization algorithms. One well-known method is proposed by Deb (2000) and is based on feasibility rules. He used this method for genetic algorithm (GA) and stated that his method was initiated from the work of Goldberg in 1992. The Deb's constraint handling methods is a tournament selection because each individual of the current population is compared with the corresponding individual in the offspring population. In other words, the ith individual of the current population is compared with the ith individual of the offspring population. This method is enforced by following the criteria below.

— Any feasible solution is preferred to any infeasible solution.

— Among two feasible solution, the one having better objective function is preferred.

— Among two infeasible solution, the one having smaller constraint violation is preferred.

Note that in the mentioned method, there is no need to apply penalty parameters because solutions are never compared simultaneously in terms of both the objective function and constraint violation values. In other words, in all of three above mentioned criteria, solutions are either compared in terms of objective function values or constraint violation values.

To evaluate the mentioned constraint handling method in conjunction with DE algorithm, 6 test problems was solved and the numerical result were compared with those of genetic algorithm proposed by Deb (2000). Four test problems were selected from the Deb's paper (Deb, 2000) to compare the performance of DE with GA. In addition, two more milestone problems, namely the Ackley and Rastrigin problems were considered to further compare DE and GA performance. Each

of these two benchmark problems has 5 variables. The details of this comparison using all 6 benchmark problems are given in Appendix A.

Table 1. Comparison of the best solution of the DE and the GA optimizer

Test Problem	True Optimum	DE 's Best Solution	GA 's Best Solution	Error (%)	
				DE	GA
Problem 1	13.59085	13.5908	13.9511	0	2.651
Problem 3	-15	-14.9888	-14.9902	0.075	0.065
Problem 4	7049.330923	7056.7	7153.6	0.105	1.479
Problem 5	680.6300573	681.0933	681.1196	0.068	0.072
Ackley	0	8.8818E-16	6.37E-4	0	0.064
Rastrigin	0	8.8818E16	8.8818E-14	0	0

In 5 out of 6 benchmark problems considered above, DE outperforms GA in finding the optimum solution. As seen in Table 1, only for test problem 3 the best computed optimal value found by GA is closer to the true optimum solution. More specifically, in test problem 3, the relative error of the optimum value found by DE is 0.075% while the relative error of GA is 0.065%. The numerical results clearly show that DE has outperformed GA in finding the global optimum solution.

Improved Adaptive Trade-off Model (IATM) is another constraints handling method proposed by Wang and Cai (2011) for solving constrained optimization problems. This method was used when the parent and offspring population are combined. Generally, the combined population certainly experiences three possibilities: (1) The infeasible situation occurs when all the individual in the combined population are infeasible; (2) The semi-infeasible situation occurs when population contains both feasible and infeasible individuals; and (3) The feasible situation

occurs when all the individuals in the population are feasible. In this method, for each situation one constraint handling method is considered as follows.

I. *The Constraint Handling Method for the Infeasible Situation*: In a constrained optimization problem, it does not make sense to use the individuals who are far from the boundaries of feasible region (Wang et al., 2008). Consequently, in this method, the population is sorted based on the value of constraint violation in ascending order. Then, NP (number of population) individuals are selected to survive to the next generation. Two methods are implemented for calculating the degree of constraint violation of individuals which depends on the constraints properties. After initiating the initial population and function evaluation, the difference between the violations of constraints is calculated by following Equation (24).

$$G^{max} = max_i\ G_k(X_i), i = 1, \dots., NP, k = 1, \dots, P \tag{24}$$

$$diff = max(G^{max}) - min(G^{max}) \tag{25}$$

where $G^{max}$is a set which contains the maximum values of constraint violation of each constraint among all population individuals $i = 1, \dots, NP$. P is the number of existing constraints in the optimization problem.

When the difference ($diff$) among the constraints is significant (i.e., greater than 200), the degree of constraint violation of each constraint is normalized through dividing it by the largest constraint violation of that constraint in the population:

$$\acute{G}(X_i) = \sum_{k=1}^{P} \left(\frac{G_k(X_i)}{G^{max}(k)} \right), i = 1, \dots, NP \tag{26}$$

The maximum constraint violation of each constraint is denoted by ${G^{max}}_k(X_i)$ in Equation (25) and Equation (26), where P is the number of existing constraints in the optimization problem, NP is the number of individuals in the initial population, and $\acute{G}(X_i)$ is the normalized

degree of constraint violation of X_i individual. Then, the mean of the normalized constraint violations of each individual is considered as its degree of constraint violation which is formulated in Equation (27).

$$G(X_i) = \frac{\acute{G}(X_i)}{P}, i = 1, \dots, NP \tag{27}$$

The right hand side of Equation (27) should be between zero and one. If the difference ($diff$) is small (i.e., less than 200), the degree of constraint violation of each individual can be found by calculating the summation of all the constraint violations. In Equation (28), $G(X_i)$ is the degree of constraint violation of X_i individual, also G_k represents the k th constraint, P is the number of existing constraints in the optimization problem, and $(\mu + \lambda)$ is the number of individuals in the combined population. Note that the μ is the size of parent population (i.e., $\mu = NP$). In this study, μ and NP are used interchangeably to mean the number of individual in the parent population. The parent population is used to generate an offspring population of size λ. The combined population is formed by combining the parent population with the offspring population, so the combined population is of size $(\mu + \lambda)$.

$$G(X_i) = \sum_{k=1}^{P} G_k(X_i), i = 1, \dots, (\mu + \lambda) \tag{28}$$

II. *The Constraint Handling Method for the Semi-Infeasible Situation*: an adaptive fitness transformation scheme is considered which in addition to transfer some feasible individuals with small objective function values to the next generation it also transfers some infeasible individuals with minor constraint violation and small value of objective function. More detail of this method follows. First, the individuals in the population are divided into two group of feasible and non-feasible individuals. More specifically, Z_1 and Z_2sets are initiated to record the subscript of the feasible and infeasible individuals, respectively.

$$Z_1 = \{i | G(X_i) = 0, i = 1, \dots, (\mu + \lambda)\} \quad (29)$$

$$Z_2 = \{i | G(X_i) > 0, i = 1, \dots, (\mu + \lambda)\} \quad (30)$$

Then, the best and the worst feasible member of the population are identified within the feasible group and denoted by X_{best} and X_{worst}, respectively. Next, a new value for objective function is found using the following equation:

$$\acute{f}(X_i) = \begin{cases} f(X_i), & i \in Z_1 \\ \max\{\varphi \times f(X_{best}) + (1 - \varphi) \times f(X_{worst}), f(X_i)\}, & i \in Z_2 \end{cases} \quad (31)$$

where $\acute{f}(X_i)$ is the converted objective function, φ is the feasibility proportion of the combined population which can be found by dividing the number of feasible individual in the combined population by the number of individuals in the combined population (i.e., $(\mu + \lambda)$). If the value of parameter φ is large (i.e., the population contains more feasible solution than infeasible solution), the converted objective function $\acute{f}(X_i)$ will be calculated by the left hand side of the Equation (31) (i.e., $\varphi \times f(X_{best}) + (1 - \varphi) \times f(X_{worst})$). As a result, $\acute{f}(X_i)$ of infeasible individuals ($i \in Z_2$) are smaller. Consequently, the probability of their survival to the next generation may increase. In contrast, if the value of parameter φ is small (i.e., the population contains more infeasible solution than feasible solution), $\acute{f}(X_i)$ will be calculated by the right hand side of the Equation (31) (i.e., $f(X_i)$). Therefore, $\acute{f}(X_i)$ of infeasible individuals ($i \in Z_2$) may be greater which increases the probability of the feasible solution survival to the next population. The purpose of this conversion is to maintain a reasonable balance between feasible and infeasible individuals of the new population.

Now, a normalized objective function is calculated by normalizing the converted objective function following Equation (32):

$$f_{norm}(X_i) = \frac{\acute{f}(X_i) - min_{j\in(Z_1\cup Z_2)}\acute{f}(X_j)}{max_{j\in(Z_1\cup Z_2)}\acute{f}(X_j) - min_{j\in(Z_1\cup Z_2)}\acute{f}(X_j)}, i = 1, \dots, (\mu + \lambda) \quad (32)$$

The constraint violation should have the same order of magnitude with the objective function value, so the normalization process presented in Equation (33) is used:

$$G_{norm}(X_i) = \begin{cases} 0, & i \in Z_1 \\ G(X_j), & , i \in Z_2, \text{second methd} \\ \dfrac{G(X_i) - min_{j\in Z_2} G(X_j)}{max_{j\in Z_2} G(X_j) - min_{j\in Z_2} G(X_j)}, & i \in Z_2, \text{first method} \end{cases} \quad (33)$$

where $G(X_i)$ is the degree of constraint violation of each individual. Also, $min_{j\in Z_2} G(X_j)$and $max_{j\in Z_2} G(X_j)$ are the minimum and maximum value of constraint violation in the population, respectively.

Finally, the final value of objective function is found by adding the normalized objective function and the normalized constraint violation of each individual in the combined population:

$$f_{final}(X_i) = f_{norm}(X_i) + G_{norm}(X_i), i = 1, \dots, (\mu + \lambda) \quad (34)$$

Then, the population is sorted based on the value of final objective function in ascending order and the μ (i.e., the number of individual in the parent population) individuals are selected to survive and transfer to the next generation. By using the above mentioned method some potential feasible and infeasible individuals of the combined population may survive into the next generation.

III. *The Constraint Handling Method for the Feasible Situation*: When the solution is feasible, all the individuals are compared based on the value of their objective function. Since the degree of constraint violation is zero the value of the objective function will be the only criteria to select the next generation. The μ (i.e., the number of individual in the parent population)

individuals with smallest value of objective function are selected among all the individuals in the combined population to construct the next generation.

2.3.3 HANDLING OF INTEGER AND DISCRETE VARIABLES

One method for solving a mixed integer problem with DE was proposed by Lapinen and Zelinka in 1999. In this method, the DE algorithm internally works with continuous variable. However integer values are used to evaluate the objective function. The basic idea is explained by Equation (35).

$$y_i = \begin{cases} x_i\,, & \text{for continuous variables} \\ INT(x_i), & \text{for integer variables} \end{cases} \tag{35}$$

where *INT ( )* is a function which converts a real valued variable to the corresponding integer valued one "before evaluation" of the objective function. The integer values are not used anywhere else in the algorithm which is essential to obtain a diverse population and maintain its robust performance (Lapinen & Zelinka, 1999).

The initialization of the population for integer variables is different from that of continuous variables given in Equation (12).

$$x_{ij}^0 = x_{Lj} + rand_{ij}\left(x_{Uj} - x_{Lj} + 1\right), i = 1,2,\dots,NP, j = \text{integer} \tag{36}$$

It is also necessary to modify the boundary constraint handling method for integer variables following Equation (37):

$$v_{ij}^{g+1} = x_{Lj} + rand_{ij}\left(x_{Uj} - x_{Lj} + 1\right), \quad \text{if} \quad v_{ij}^{g+1} \le x_{Lj} \textbf{ or } \; v_{ij}^{g+1} \ge x_{Uj} \tag{37}$$

There is a straight forward method to handle discrete variables. Suppose a set of discrete variables with l elements should be assigned to discrete design variables. First, the set of discrete variables are sorted in ascending order and saved in set named E. Then, another set, L, is initiated containing variable indices 1 to l (i.e., $L = \{1,2,\dots,l\}$). Hence, set L represents the index of

elements in set E. This conversion makes it possible to transform the original set of discrete variables E to a set of continuous integer variables L. Therefore, the original population is generated via following formula:

$$x_{ij}^{0} = x_{Lj} + round\left(rand_{ij}\left(x_{Uj} - x_{Lj}\right)\right), i = 1,2,\dots,NP, j = \text{discrete} \quad (38)$$

Since the discrete variable should be chosen from set L, the upper bound and lower bound values are equal to the smallest and largest values in the set (i.e., $x_{Lj} = 1$, and $x_{Uj} = l$). Consequently, Equation (38) can be modified as follows:

$$x_{ij}^{0} = 1 + round\left(rand_{ij}(l-1)\right), i = 1,2,\dots,NP, j = \text{discrete} \quad (39)$$

Based on Equation (39), an integer value between 0 and $(l-1)$ (the span between lower and upper bounds) is added to the lower bound, x_{Lj}, which has the value of 1. Therefore, all initial individuals are integers while the diversity of the initial population is ensured.

Since the variables have to be integer values between 0 and l, it is also necessary to modify the mutation method. To this end, Equation (13) to (18) are modified as follows:

DE/rand/1: $$v_i^{g+1} = x_{r1}^{g} + round\left(F.\left(x_{r2}^{g} - x_{r3}^{g}\right)\right) \quad (40)$$

DE/rand/2: $$v_i^{g+1} = x_{r1}^{g} + round\left(F.\left(x_{r2}^{g} - x_{r3}^{g}\right) + F.\left(x_{r4}^{g} - x_{r5}^{g}\right)\right) \quad (41)$$

DE/best/1: $$v_i^{g+1} = x_{best}^{g} + round\left(F.\left(x_{r1}^{g} - x_{r2}^{g}\right)\right) \quad (42)$$

DE/best/2: $$v_i^{g+1} = x_{best}^{g} + round\left(F.\left(x_{r1}^{g} - x_{r2}^{g}\right) + F.\left(x_{r3}^{g} - x_{r4}^{g}\right)\right) \quad (43)$$

DE/current-to-rand/1: $$v_i^{g+1} = x_i^{g} + round\left(F.\left(x_{r1}^{g} - x_i^{g}\right) + F.\left(x_{r2}^{g} - x_{r3}^{g}\right)\right) \quad (44)$$

DE/current-to-best/1: $$v_i^{g+1} = x_i^{g} + round\left(F.\left(x_{best}^{g} - x_i^{g}\right) + F.\left(x_{r1}^{g} - x_{r2}^{g}\right)\right) \quad (45)$$

It is important to note that the crossover operation exchanges information between target vector and mutant vector. Consequently, the elements of the trial vectors are also integers. For

selection operation, it is necessary to transform the integer values to corresponding discrete value in the set E. Because for objective function and constraints evaluation we need to use the original discrete values for discrete design variables.

In this method, the evolution of population and constraint handling method is the same as the original DE algorithm. Hence, by using this method, the advantages of the DE algorithm are preserved and while expanding the application and usefulness of DE algorithms (Ho-Huu et al., 2015).

2.3.4 DE PARAMETERS

There are three total DE control parameters that should be tuned: the population size, NP, the scale or mutation Factor, F, and the crossover probability or Crossover rate, Cr. The scale factor, F, controls the size of the search area around the base individual. The Crossover rate, Cr, implies the probability of inheriting elements from the mutant individual in the development of each trial individual.

There is no consistent methodology for determining the control parameters of an evolutionary algorithm (Brest et al., 2006). The performance of DE is sensitive to the selected control parameters. Changing the DE parameters leads to variations in DE performance characteristics. The DE control parameters should be tuned for each individual problem and may differ from one problem to the other (Islam et al., 2012). Tuning the DE control parameters is challenging because of their dependency on the nature and size of the optimization problems (Azad & Fernandes, 2013). As a common practice, most of the traditional DE algorithms use a set of fixed control parameters or set them within some predefined ranges (Brest et al., 2006). It is necessary to note that DE is much more sensitive to the choice of F than it is to the choice of Cr.

Setting the control parameters can be classified into three categories: constant, random, and adaptive (including self-adaptive) (Tang et al., 2015). In constant parameter setting, used in classic DE, parameters are defined before starting the search process and kept constant for all the iterations. Storn and Price (1997) declared that it is not difficult to choose control parameters for finding good results. Based on their experience, a suitable range for NP (number of population) is between $5D$ and $10D$, where D is the number of decision variables. They also suggested that NP must be at least 4 to guarantee that DE will have enough mutually different vectors with which to work. For control parameter F, the best and reasonable initial choice is 0.5. If the population converges prematurely, then F and/or NP should be increased. Values of F smaller than 0.4 and greater than 1, are only occasionally effective. An appropriate first choice for Cr is 0.1, considering that a large Cr often speeds up the convergence, so it's proper to first try $Cr = 0.9$ or $Cr = 1.0$ for checking if a quick solution is desired.

Rönkkönen et al. (2005) suggested that a reasonable range of NP is between $2D$ and $40D$. Also, the control parameter F should be selected between 0.4 and 0.95 (0.9 provides a compromise between exploration and exploitation), and that Cr should be drawn from the range (0.0, 0.2) if the problem is separable, or [0.9, 1] if the problem is both nonseparable (i.e., the objective function is nonseperable) and multimodal (i.e., optimization that involves finding all or most of the multiple solutions of a problem). They set F and Cr to 0.9, and NP to 30 for all experimental functions in the CEC 2005 contest benchmark suite. Feoktistov (2006), in his book, recommended that parameter F should be a constant in the range [0, 2]. Gamperle et al. (2002) stated that a proper range of NP is between $3D$ and $8D$, an effective initial value for F is 0.6, and a suitable range for Cr is [0.3, 0.9]. The diverse conclusions found by these researchers implies it is nearly impossible

that one constant parameter setting fits all problems, and effective control parameters are problem-dependent.

Random parameter setting can be adopted to automatically set the control parameter values and avoid manual parameter assignments. The common rules which are usually employed for generating diverse values for the control parameters are: Linear variation, probability distribution, and specified heuristic (Tang et al., 2015). In a method presented by Das et al. (2005), the control parameter F was set using two different approaches. In one approach, the value of the control F was assigned randomly and in the other approach, it was defined as a time-varying value. In the random method, F was set to be a random real number from the range of (0.5, 1); for the time varying method, F was reduced linearly within a given range of $[0.4, 1.2]$. In a different study, F was randomly generated from a normal distribution, $N(0.5, 0.3)$, for each target individual in the current population (Qin et al., 2009). Abbass (2002) generated control parameter F from the standard normal distribution $N(0, 1)$. Similarly, Omran et al. (2005) generated the control parameter Cr from a normal distribution $N(0.5, 0.15)$. It was shown that the random setting increases searching diversity which can improve the exploration ability of the DE algorithm.

Adaptive control parameter is another automatic parameter setting method in which adjusts the control parameters according to the feedback from the searching process (Liu & Lampinen, 2005; Brest et al., 2006), or through evolutionary operation (Abbass, 2002; Teo, 2006). Liu and Lampinen (2005) introduced the fuzzy adaptive differential evolution algorithm, which uses fuzzy logic controllers to tune the control parameters F and Cr by incorporating relative objective function values and the individuals of successive generations as inputs. One of the famous self-adaptive methods was proposed by Brest et al. (2006). This method assigns the values from the ranges [0.1, 1.0] and [0.0, 1.0] in an adaptive manner with probabilities τ_1 and τ_2 to F and Cr,

respectively. Control parameters F and Cr are generated for each individual member of the population during mutation and crossover operations. Zhang and Sanderson (2009) introduced another adaptive method which Cr is generated from a normal distribution and F is generated by a Cauchy distribution for each individual at each generation. Later, an improve version of this method was proposed by Tanabe and Fukunaga (2013). They used a different success-history based mechanism to update F and Cr. Qin et al. (2009) also presented a DE variant with an adapting control parameters method. They also used a normal distribution with mean value 0.5 and standard deviation 0.3, denoted by $N(0.5,0.3)$ for generating F. In this method, they adaptively adjusts Cr values following a normal distribution with the mean value depending on the previous successful Cr values.

Mohamed et al. (2012) presented a self-adaptive control parameters method for their novel DE algorithm. For generating F, a uniform random probability distribution was used and for Cr a dynamic nonlinearly increased probability scheme was implemented. A difference-based mechanism was proposed by Tang et al. (2015) in which the control parameters and mutation operators were set for each individual based on the current generation value of the objective function which improved the DE convergence rate and diversity. Other self-adaptive methods implemented by researchers in Fan and Yan (2015) and Zamuda and Brest (2015) in which each individual had its own mutation and crossover parameters. One famous method which was proposed by Brest et al. (2006) is explained below.

In this method, control parameters F and Cr are encoded into the individual. The better values of these (encoded) control parameters lead to better individuals which are more likely to survive and generate offspring and consequently propagate these better parameter values.

For each individual, with a probability τ_1, control parameter F is reinitialized a new random value in the range of [0.1, 1.0], otherwise, it retains its earlier value in the next generation. The control parameter Cr is adapted in the same way, but with a different re-initialization range of [0.0, 1.0] and with the probability of τ_2 for each individual. With probabilityτ_2, Cr takes a random value in [0.0, 1.0], otherwise it is kept unchanged. This control parameters setting method is calculated as described:

$$F_i^{g+1} = \begin{cases} F_l + \text{rand}_1 F_u & if\ \text{rand}_2 \leq \tau_1 \\ F_i^g & otherwise \end{cases} \tag{46}$$

$$Cr_i^{g+1} = \begin{cases} \text{rand}_3 & if\ \text{rand}_4 \leq \tau_2 \\ Cr_i^g & otherwise \end{cases} \tag{47}$$

where $i = 1,2, \ldots, NP$ (NP represent number of population), $rand_j$, $j \in \{1,2,3,4\}$ are uniform random values $\in$[0.0,1.0]. τ_1 and τ_2 represent probabilities to adjust factors F and Cr, respectively. Brest et al. (2006) set τ_1 and τ_2 to 0.1. The value of F_l =0.1 and F_u =0.9, the new F could take a value from [0.1, 1.0] in a random manner. The new Cr was assigned a value from the range of [0.0, 1.0]. $F_i^{g+1}$and Cr_i^{g+1} were calculated before the mutation was performed. Therefore, they influence the mutation, crossover, and selection operations of the new vector x_i^{g+1}. The initial value for F and Cr were set to 0.9 and 0.5, respectively.

2.4 MODIFIED DIFFERENTIAL EVOLUTION

Generally, all the variant of DE algorithm are based on the basic DE, but they employ various methods for mutation, crossover, and selection operations as well as in setting the control parameters. Table 2 contains a pseudo code to show the procedure of DE algorithm. The DE ability to solve a specific problem depends significantly on the choice of methods to perform operations

such as mutation, crossover, and selection (Price et al., 2006), and the setting of control parameters (Eiben et al., 1999; Gämperle et al., 2002).

Table 2. Differential evolution algorithm

```
Initialization
Objective Function and Constraints Evaluation
for g=1: Number of Generations
    for i=1: NP (Number of Populations)
        for j=1: Number of design variables
            Generate Trial Vector:
                Perform mutation
                Perform crossover
                Check Boundary Constraint
        end for
        Evaluate the Objective Function and Constraints
        Select the offspring population
    end for
    Select the best solution as the optimum solution
End for
```

Improper combinations of mutation methods and control parameters can cause stagnation or premature convergence because of over exploration or over exploitation, respectively. In exploration, the algorithm searches every promising solution area with good diversity. In exploitation, the algorithm executes a local search in some promising solution areas to find the optimal point with a high convergence rate.

The crossover operator constructs a new trial/offspring vector from the current and mutant vectors. It also controls which components and how many of them are mutated in each vector of the current population. The parameter Cr can be considered as a mutation probability because it controls the number of components inherited from the mutant vector. Previous studies showed that

parameter Cr influences the convergence speed and its proper value is problem specific (Gämperle et al., 2002).

Hence, choosing appropriate methods for DE operations and setting control parameters to get a good balance between the algorithm's effectiveness (solution quality) and efficiency (convergence rate) is still an open field of research.

The Improved Constrained Differential Evolution (ICDE) algorithm combines an improved $(\mu + \lambda)$-differential evolution (IDE) method with an Archiving-based Adaptive Tradeoff Method (ArATM) to solve constrained optimization problem. Recently a DE variation is proposed to solve discrete-continuous truss optimization problems (Ho-Huu et al., 2015). This method which is called D-ICDE integrates a discrete variable scheme into the ICDE (Jia et al., 2013) for solving discrete-continuous truss optimization problems.

In this study, a new variant of the Differential Evolution algorithm for solving mixed discrete-continuous truss optimization problem is proposed. To validate the proposed algorithm the numerical results are compared with those of D-ICDE.

The IDE method is a combination of mutation and crossover operations which generates the offspring population. The proposed DE algorithm developed in this work modifies both mutation and crossover strategies of IDE method. In addition, the DE parameters such as scaling factor and crossover rate are modified. For the selection operation of the proposed algorithm the Improved Adaptive Trade off Method (IATM) is adopted which was initially proposed by Wang and Cia (2011). The IATM was used in the Constrained Differential Evolution (CDE) algorithm and explained in Section 2.3.2.2 under constraint handling methods.

Note that the IDE was combined with an ArATM in ICDE algorithm (Jia et al., 2013). This is different form the proposed algorithm where a modified IDE is combined with IATM method.

The contribution of this work is further explained in the following sections. In Section 2.4., first the original IDE method is explained then, the proposed modified IDE is described in details. Moreover, the differences between IATM used in the proposed algorithm and ArATM is clarified in Section 2.4.2.

2.4.1 THE OFFSPRING GENERATION

As mentioned before, a proper combination of mutation and crossover strategies can improve the performance of DE algorithm. IDE is introduced as a reliable method in the field of constrained evolutionary optimization. It serves as the search engine of DE algorithm which includes the mutation and crossover operations. In this study, the mutation and crossover strategies as well as scaling factors and crossover rate of the original IDE were modified to generate a superior DE algorithm for solving truss optimization problems.

To explain the proposed algorithm in depth, first the original IDE is presented in Section 2.4.1.1. Then, in Section 2.4.1.2, the proposed modified IDE method is described.

2.4.1.1 THE ORIGINAL IDE

The original IDE adopts three mutation and crossover strategies to generate the offspring population (Jia et al., 2013). The parent population, P_g, with μ (i.e., the number of parent population, NP) individuals generates offspring population, Q_g, with λ (i.e., the size of the offspring population) individuals by operating the following procedure:

Step.1 Set $Q_g = \Phi$;

Step.2 For each individual $X_i, i = 1,2, \dots, \mu$ in P_g

Step.3 generate the first offspring, Y_1, by using the "DE/rand/1" strategy and the binomial crossover;

Step.4 generate the second offspring, Y_2, by using the "DE/rand/2" strategy and the binomial crossover (explained in Section 2.3.1.3);

Step.5 generate the third offspring, Y_3, by using a new mutation strategy "DE/current-to-rand/best/1" and the improved Breeder Genetic Algorithm (BGA) mutation;

Step.6 $Q_g = Q_g \cup Y_1 \cup Y_2 \cup Y_3$

Step.7 End

The mutation strategy "DE/current-to-rand/best/1" which was used in ***Step.5*** was first proposed by Jia et al. (2013). This mutation strategy is a combination of two mutation strategies: "DE/current-to-rand/1", and "DE/current-to-best/1". In the "DE/current-to-rand/best/1" strategy, first the current generation number is compared with a threshold generation number which is found using Equation (48).

$$threshold\ generation\ number = k \times total\ generation\ number \tag{48}$$

If the $current\ generation\ number$ is smaller than the $threshold\ generation\ number$, then "DE/current-to-rand/1" strategy is used to generate the third offspring, Y_3, and crossover strategy is not applied to the mutant vector. Factor k in Equation (48) is set to $k = 0.6$. If the $current\ generation\ number$ is greater than $threshold\ generation\ number$, then "DE/current-to-best/1" strategy is used the third offspring, Y_3, along with the improved BGA mutation strategy (Wang et al., 2007) in order to increase the diversity of the population. The improved BGA mutation strategy is applied to the mutation vector, v_{ij}, with a probability p_m for producing the offspring Y_3. This methods works as follows:

$$v_{ij} = \begin{cases} v_{ij} \pm rang_i \times \sum_{r=o}^{15} \alpha_r 2^{-r}, if\ rand < \frac{1}{D} \\ v_{ij}, otherwise \end{cases} \quad , i = 1, \dots, NP; j = 1, \dots, D \tag{49}$$

where $rand$ is a uniformly distributed random number between 0 and 1, and $rang_i$ is the mutation range and is set to

$$rang_i = (x_{Uj} - x_{Lj}).\left(1 - \frac{current\ generaton\ number}{total\ generaton\ number}\right)^6 \quad (50)$$

The + and – sign in Equation (49) is chosen with a probability of 0.5 (i.e., if a uniformly distributed random number between 0 and 1 is less than 0.5 the sign is +, else the sign is –), and $\alpha_r \in \{0,1\}$ is randomly generated with probability of $p(\alpha_r = 1) = 1/16$.

To further explain the implementation of the ‘‘DE/current-to-rand/best/1’’ strategy, a schematic diagram is depicted in Figure 1. The whole evolutionary process is divided into two phases by considering the threshold generation number as shown in Figure 1. Generally, the evolutionary process is divided into three stages: the early stage, the middle stage, and the later stage. It is expected that the first phase includes the early stage and some part of the middle stage of evolution, while the second phase includes the remaining part of the middle stage and the later stage of evolution (Jia et al., 2013).

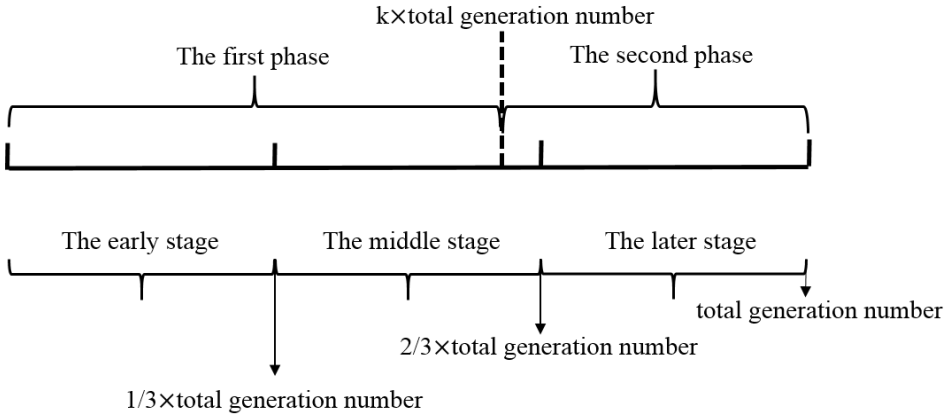

Figure 1. The schematic diagram for the evolutionary process during the ‘’ DE/current-to-rand/best/1 ‘’ strategy is divided in to two phases and three stages.

In order to prevent the population from getting stuck in a local optimum, the global search ability of the population should be improved in the first phase. The ‘‘DE/current-to-rand/1’’ strategy is a proper choice here because in this strategy, the individuals learn the information from other individuals randomly chosen from the population. Therefore, the ‘‘DE/current-to-rand/1’’ strategy was used in the first phase of this study in which the first scaling factor of this strategy was randomly chosen between 0 and 1 to further enhance the global search ability.

The ‘‘DE/current-to-best/1’’ strategy exploits the information of the best individuals in the current population. As a result, it accelerates the convergence of the population and guides the population toward the global optimum. Hence, this strategy is well suited and used for the second phase in this study. As discussed before, in order to preserve a good balance between the diversity and convergence of the population, the ‘‘DE/current-to-rand/best/1’’ strategy is implemented by combining the ‘‘DE/current-to-rand/1’’ and ‘‘DE/current-to-best/1’’ strategies.

2.4.1.2 OFFSPRING GENERATION – MODIFIED IDE

The IDE method is modified in the proposed algorithm to enhance its local search capability and increase its convergence rate. More specifically, ***Step 3*** and ***Step 5*** of the original IDE, which were explained in the previous section, were modified. The proposed modified IDE method is described in following procedure.

Step.1 Set $Q_g = \Phi$;

Step.2 For each individual $X_i, i = 1,2, \dots, \mu$ in P_g;

Step.3 Generate the first offspring, Y_1, according to the following procedure:

- Find the scaling factor by the proposed rank based or random method for “directed mutation” or “DE/rand/1” strategies, respectively;

- Perform mutation by using a combination of "DE/rand/1" and the "directed mutation" strategies through a linearly decreasing probability rule;
- Calculate the crossover rate by a dynamic nonlinearly decreasing probability scheme;
- Generate the offspring population by binomial crossover;

Step.4 Generate the second offspring, Y_2, by using the "DE/rand/2" strategy and the binomial crossover ($F = 0.6$, $Cr = 0.95$);

Step.5 Generate the third offspring, Y_3, by using a "DE/current-to-rand/best/1" strategy. Set factor k in Equation (48) to $k = 0.6$.

- If $current\ generation\ number \leq threshold\ generation\ number$, then use "DE/current-to-rand/1" and ***binomial crossover*** strategies to generate the third offspring, Y_3 ($F = 0.9,\ Cr = 0.95$). Note no crossover is used in this step of the original IDE;
- If $current\ generation\ number > threshold\ generation\ number$, then use "DE/current-to-best/1" strategy along with the improved BGA mutation strategy ($F = 0.6$) to generate the third offspring, Y_3;

Step.6 $Q_g = Q_g \cup Y_1 \cup Y_2 \cup Y_3$

Step.7 End

Unlike the original IDE, in ***Step.3***, a mutation strategy is used which has been introduced first in Alternative Differential Evolution (ADE) for solving unconstrained optimization problems (Mohamed et al., 2012). The proposed mutation strategy is a combination of the "directed mutation" and "DE/rand/1" strategies. The directed mutation strategy is obtained based on the weighted difference vector between the best and the worst individuals in the current population. This strategy is combined with "DE/rand/1" through a linear decreasing probability rule as follows:

If

$$rand \geq (1 - \frac{current\ generation\ number}{total\ generaton\ number}) \tag{51}$$

then

$$v_i^{g+1} = x_r^g + F_1.\left(x_{best}^g - x_{worst}^g\right) \tag{52}$$

else

$$v_i^{g+1} = x_{r1}^g + F_2.\left(x_{r2}^g - x_{r3}^g\right) \tag{53}$$

The directed mutation strategy is formulated in Equation (52) in which r is a randomly chosen value, not equal to i, and is picked from the range of $[1, NP]$. Also, x_r^g is a randomly selected individual among the current population. x_{best}^g and x_{worst}^g are individuals with the best and worst objective function values, respectively.

The mutation scaling factor, F, is an important parameter that controls the evolving rate of the population. The value of F has a considerable effect on exploration: small values of F lead to premature convergence, and high values of F slow down the search (Feoktistov, 2006). In this work, two scaling factors F_1 and F_2 are proposed for the two different mutation rules.

For the mutation presented in Equation (52), the generated difference vector is in fact a directed difference vector from the worst to the best vectors of the current population. In order to maintain the same search direction for all the target vectors, F_1 must have a positive value. Therefore, in this study, F_1 is defined as a normal random number picked from a normal distribution with mean value of (i/NP) and standard deviation of 0.1; i is the index of the current target vector.

This method of finding F_1 is a rank-based method because all the individuals in the current population are sorted in ascending order based on the value of the objective function. Consequently, X_i is the ith superior individual in the current population.

This proposed ranked-based scheme is a modified version of the scheme used in Individual Dependent Mutation (IDM) strategy (Tang et al., 2015). Tang et al., (2015) used the based vector index of mutation strategy instead of the current population index used in this study (i.e., i). For this mutation, the binomial crossover is used.

The difference vector in the "DE/rand/1" strategy, Equation (53), is a pure random difference vector because the objective function values are not used for finding their indices. Since the best direction that can lead to good exploration is unknown, F_2 is introduced as a uniform random variable in the interval of $(-1, 0) \cup (0, 1)$. Furthermore, F_2 helps to advance the exploration and to cover the whole search space by generating mutant vectors with opposite directions. Also, F_2 is set to be random for each target vector such as F_1.

The basic mutation strategy (DE/rand/1) with the constant scaling factor, the new directed mutation strategy, and the modified basic mutation strategy are depicted in Figure 2 (Mohamed et al., 2011). The process of generating a mutation vector, v_i, for each individual x_i using the "DE/rand/1" mutation strategy and a constant scaling factor, F, is depicted in Figure 2a.

Using the directed mutation strategy, two new mutant vectors $v_1$and v_2 are generated for two target vectors x_1 and x_2 with small, and large random positive scaling factors, respectively. This is illustrated in Figure 2b. Additionally, v_i is the mutation vector generated for individual x_i using the "DE/rand/1" mutation strategy with random scaling factor F_2 where F_2 is assigned both a large negative number and a small positive number shown in Figure 2c.

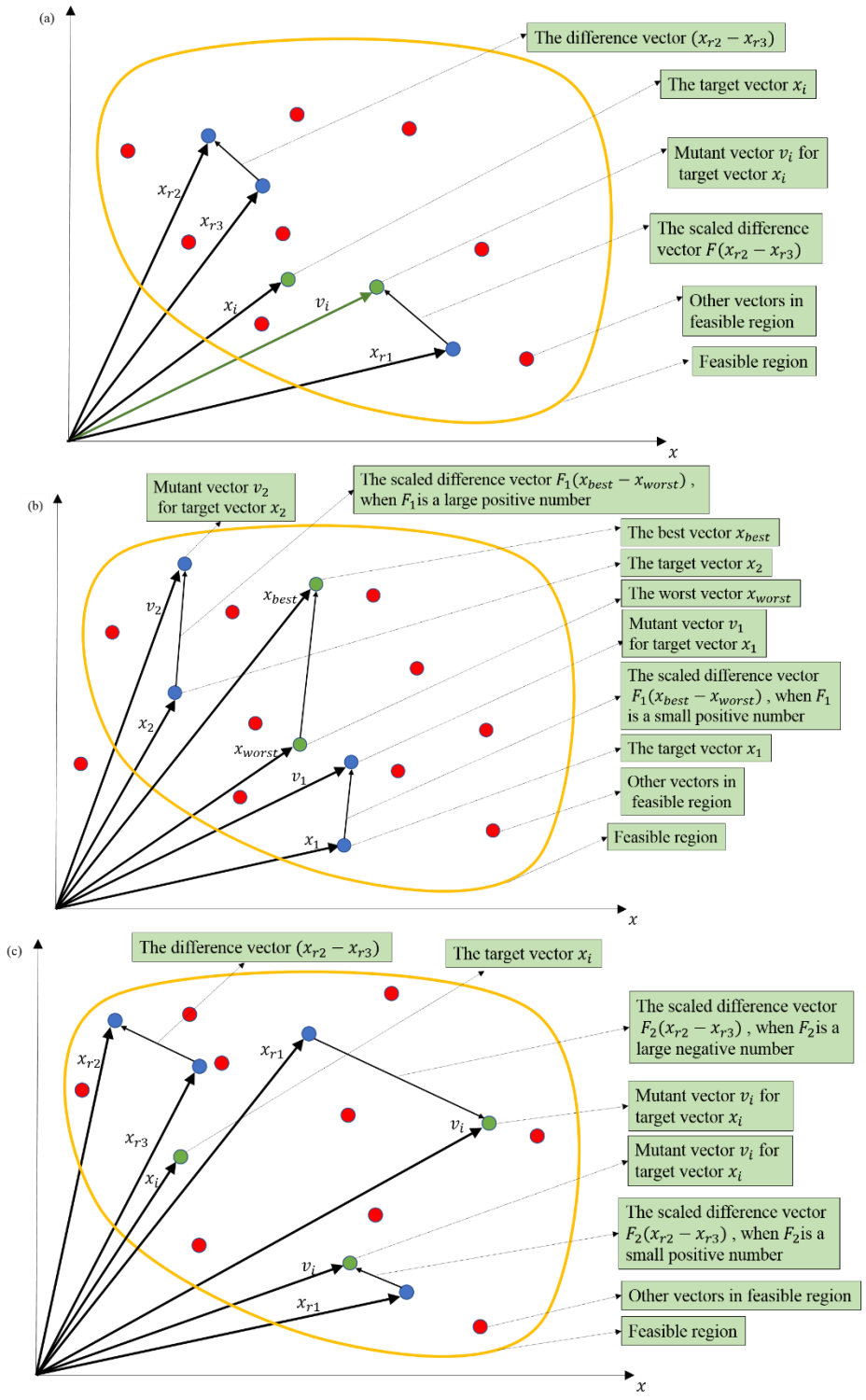

Figure 2. (a) An illustration of the DE/rand/1 a basic DE mutation strategy in two-dimensional parametric space. (b) An illustration of the new directed mutation strategy in two-dimensional parametric space (local exploitation). (c) An illustration of the modified DE/rand/1 basic DE mutation strategy in two-dimensional parametric space (global exploration).

After mutation operation, the binomial crossover is used to generate the trial vector. The Crossover rate, Cr, controls the population diversity. A dynamic non-linearly increasing crossover probability strategy is used (Mohamed et al., 2012) as follows:

$$Cr = Cr_{max} + (Cr_{min} - Cr_{max}).\left(1 - \frac{current\ generation\ number}{total\ generation\ number}\right)^4 \quad (54)$$

where Cr_{min} and Cr_{max} are the minimum and maximum value of the Cr, respectively. The optimal setting for these parameters are $Cr_{min} = 0.1$ and $Cr_{max} = 0.9$ in this study which Cr_{max} is different from the one used by Mohamed et al. (2012).

Based on this strategy, the algorithm starts at $current\ generation\ number$ = 1 with Cr value close to $Cr_{min} = 0.1$. However, as generation number increases toward $total\ generaton\ number$ the Cr value increases to reach $Cr_{max} = 0.9$.

In order to avoid high level of diversity in the early stages, small value of Cr is considered as a good initial rates (Storn & Price, 1997). Furthermore, when the maximum value of Cr is close to $Cr_{max} = 0.9$ a balance can be achieved between exploration and exploitation. The mutation vector contributes more to the trial/offspring vector for larger values of Cr.

In the beginning of the search process, the vectors in the population are completely different from each other and population is completely diverse. To prevent the high level of diversity that may result in premature convergence and decrease convergence speed, the value of Cr must be small. Then, over generations, the vectors in the population become more and more similar and the diversity of the population will decrease. Hence, at this stage, the value of Cr must be large to promote diversity and increase the convergence rate.

Moreover, in ***Step 5*** of the proposed modified IDE, when the $current\ generation\ number$ is smaller than $threshold\ generation\ number$ and "DE/current-to-rand/1" strategy is

used for generating the third offspring Y_3, the binomial crossover strategy is applied to the mutant vector.

2.4.2 IATM AND ITS DIFFERENCES WITH ArATM

In IDE method, the parent population, P_g, with size μ (i.e., the number of population $\mu = NP$) generates an offspring population, Q_g, with size λ. Then, a combined population $P_g + Q_g$ is constructed by combining the parent and offspring populations. The combined population has ($\mu + \lambda$) members and may inevitably experience one of these three situations: the infeasible situation, the semi-feasible situation, and the feasible situation. As mentioned in Section 2.3.3 under constraint handling method, in the infeasible situation, the population only contains infeasible individuals; in the semi-feasible situation, the populations contain both feasible and infeasible individuals; and in the feasible situation, all the individuals in the population are feasible.

The original IDE method was combined with ArATM as the selection operator in the ICDE algorithm proposed by Jia et al. (2013). In this study to decrease the computational cost, IATM (Wang & Cai., 2011) method was adopted as the selection operator. The IATM method was explained in details in Section 2.3.3. The differences between this method and ArATM are summarized below:

I. For the infeasible situation, IATM and ArATM implement different constraint-handling methods. ArATM randomly selects $randsize$ individuals from a predefined archive, A, then puts them into the combined population, H_g, to improve the diversity of the population. Where $randsize$ is an integer randomly generated between 0 and the size of A. The archive A is used to store the individuals of H_g which cannot survive to the next population. Then, ArATM uses the hierarchical non-dominated individual selection method developed by Wang et al. (2008) to select the promising individuals for the next generation. In IATM method, the population

is sorted based on the degree of constraint violation in ascending order. Then, the μ individuals are selected to survive to the next generation. Different methods were implemented for calculating the degree of constraint violation of individuals. These methods are problem specific and depend on the property of the constraints as mentioned in Section 2.3.2.2.

II. For the semi-feasible situation, in ArATM, the objective function is converted based on the feasibility proportion of the combined population, H_g, to achieve a good balance between the diversity and the convergence of the population. However, the conversion of the objective function in IATM, is based on the feasibility proportion of the last population and in each generation there is no need to save the individuals that were not survived to next generations.

2.4.2 PROPOSED DE ALGORITHM

The main procedure of the proposed DE algorithm is shown in Table 3. Initially, a population, P_g, of size μ is produced. Then, this population is used to create an offspring population, Q_g, of size λ by using the proposed modified IDE method explained in Section 2.4.1.2. After combining P_g and Q_g , IATM presented in Section 2.3.2.2 is applied to select μ promising individuals for the next population. This procedure will continue until the termination criterion is satisfied.

The discrete variables rule which was explained in Section 2.3.3 is used to handle the discrete variables. To initialize the discrete variables, Equation (36) is used. The mutation operations are modified following the proposed modifications of Section 2.3.3.

After crossover, the integer values corresponding to the discrete variables should be exchanged to their real values before evaluation of the objective function and constraints. This is the only part of the code that the real value of discrete variables should be used.

Table 3. The procedures of the proposed DE algorithm

set NP , number of population, $\mu = NP$
initialization: generate an initial population, P_0, by randomly sampling from the search space S ;
compute the variable $diff$ using Equation (24) to determine the suitable method for calculating degree of constraint violation of individuals during the evolution;
Evaluation: Evaluate objective function and constrains for each individual in the initial population P_0;
$g = 0$;
repeat
  $g = g + 1$;
  $P_g = P_{(g-1)}$;
  $Q_g = \Phi$;
  for *each individual in the population P_g* ***do***
    - *generate the first offspring, Y_1, by using the proposed combined mutation strategy and the binomial crossover;*
    - *generate the second offspring, Y_2, by using the ''DE/rand/2'' strategy and the binomial crossover;*
    - *generate the third offspring, Y_3 , by using the ''DE/current-to-rand/best/1', binomial crossover, and the improved breeder genetic algorithm (BGA) mutation;*
    - $Q_g = Q_g \cup Y_1 \cup Y_2 \cup Y_3$

  end
  compute the objective function value and the degree of constraint violation for each individual in the population Q_g ;
  check number of feasible solution in the combined population $(P_g + Q_g)$ of size $(\mu + \lambda)$;
  determinate the current situation of the combined population $(P_g + Q_g)$ in terms of feasibility;
  select the best μ individuals from the combined population $(P_g + Q_g)$based on the IATM and generate the new population P_{g+1};
  find the best and the worst individuals of the newly generated population P_{g+1};
until *the stopping criterion is met;*

output*: print out the best individual of the population P_g*

2.5 NUMERICAL IMPLEMENTATION

In this section, the performance of the proposed DE algorithm is tested on five well-known benchmark truss optimization problems. For all of these examples, the population number is consider as 15 ($NP = \mu = 15$) and λ =45. The examples are divided into two groups: planar trusses and space trusses based on characteristics of their structure. Note that the proposed algorithm was coded in the Matlab environment (Matlab R2015b). The test problems were executed on an *Intel(R) Core(TM)2 Duo CPU P8600@ 2.40GHz* PC under Windows 7.

2.5.1 FIFTEEN BAR PLANAR TRUSS

This fifteen-bar planar truss was also studied by Wu and Chow (1995), Tang et al. (2005), Miguel et al. (2013), and Ho-Huu et al. (2015). The ground structure is illustrated in Figure 3. The objective is to minimize the weight of the truss with stress constraints. A vertical load of $10{,}000\ lb$ is applied on node 8. The stress limit (σ) is 25,000 (psi) for both tensile and compressive stresses for all members. Young's modulus (E) is specified as 1.0×10^7 (psi) and the material density (ρ) is 0.1 ($\frac{lb}{in^3}$). The x and y coordinates of joints 2, 3, 6, 7 are allowed to vary, joints 6 and 7 are constrained to have the same x coordinates as joints 2, and 3, respectively. Joints 4 and 8 are allowed to move only in y direction. The problem includes 15 sizing variables (cross-sectional area of members) and 8 geometry variables ($x_2 = x_6$, $x_3 = x_7$, y_2, y_3, y_4, y_6, y_7, y_8). Side constraints for geometry variables are:

- $100\ (in.) \leq x_2 \leq 140\ (in.)$
- $220\ (in.) \leq x_3 \leq 260\ (in.)$
- $100(in.) \leq y_2 \leq 140(in.)$
- $100(in.) \leq y_3 \leq 140(in.)$
- $50(in.) \leq y_4 \leq 90(in.)$
- $-20(in.) \leq y_6 \leq 20(in.)$

- $-20(in.) \leq y_7 \leq 20(in.)$
- $20\ (in.) \leq y_8 \leq 60\ (in.)$

The cross-sectional areas are taken from the set D =(0.111, 0.141, 0.174, 0.220, 0.270,0.287,0.347,0.440,0.539,0.954,1.081,1.174,1.333, 1.488, 1.764,2.142, 2.697, 2.800, 3.131, 3.565, 3.813, 4.805, 5.952, 6.572,7.192, 8.525, 9.300, 10.850, 13.330, 14.290, 17.170, 19.180) (in^2).

First, the problem is solved as a size and shape optimization with discrete variables; then as a size, shape, and topology optimization problem.

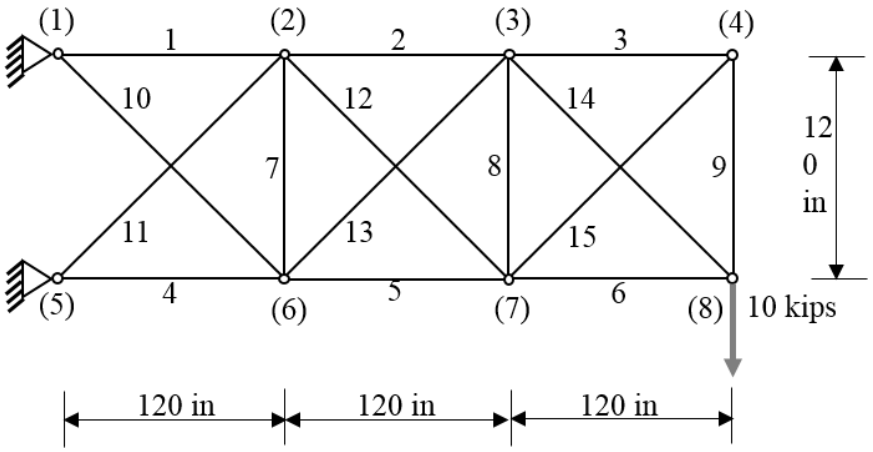

Figure 3. 15 bar planar truss

2.5.1.1 SIZE AND SHAPE OPTIMIZATION

This problem has been solved in the literature using different methods. The result of this work compared with the result of Tang et al. (2005), Miguel et al. (2013), and Ho-Huu et al. (2015); they used a genetic algorithm, a firefly algorithm, and a D-ICDE algorithm for solving this problem, respectively. The numerical results obtained in this study are compared with the

aforementioned results found in literature and presented in Table 4. To keep the tradition, similar to previous studies, the presented numerical result is the best solution found over 100 runs.

As presented in Table.4, the best result found in the literature for this problem achieves the minimum truss weight of 74.6818 $(lb.)$ after 7980 objective function evaluation in study of Ho-Huu et al. (2015). The proposed algorithm in the present study, achieves a lower minimum truss weight of 72.73 $(lb.)$ only after 7216 objective function evaluations.

Table 4. Optimum size and shape solution for 15 bar planar truss

Design Variables	**Tang et al., 2005**	**Miguel et al., 2013**	**Ho-Huu et al., 2015**	**This work**
$A_1(in^2)$	1.081	0.954	1.081	0.954
$A_2(in^2)$	0.539	0.539	0.539	0.539
$A_3(in^2)$	0.278	0.22	0.141	0.174
$A_4(in^2)$	0.954	0.954	0.954	0.954
$A_5(in^2)$	0.954	0.539	0.539	0.539
$A_6(in^2)$	0.22	0.22	0.278	0.278
$A_7(in^2)$	0.111	0.111	0.111	0.111
$A_8(in^2)$	0.111	0.111	0.111	0.111
$A_9(in^2)$	0.287	0.287	0.141	0.174
$A_{10}(in^2)$	0.22	0.44	0.347	0.44
$A_{11}(in^2)$	0.44	0.44	0.44	0.44
$A_{12}(in^2)$	0.44	0.22	0.27	0.174
$A_{13}(in^2)$	0.111	0.22	0.27	0.174
$A_{14}(in^2)$	0.22	0.27	0.278	0.278
$A_{15}(in^2)$	0.347	0.22	0.174	0.174
$x_2\ (in)$	133.612	114.967	100.0309	110.3585
$x_3\ (in)$	243.752	27.04	238.701	246.825
$y_2(in)$	100.449	125.919	132.8471	134.228
$y_3(in)$	104.738	111.067	125.3669	112.8878
$y_4\ (in)$	73.762	58.298	60.3072	55.4056
$y_6\ (in)$	-10.067	-17.564	-10.6651	-18.0723
$y_7(in)$	-1.339	-5.821	-12.2457	2.0917
$y_8\ (in)$	50.402	31.465	59.9931	55.0023
Weight (lb.)	**79.82**	**75.55**	**74.6818**	**72.73**
Function Evaluation	8000	8000	7980	7216

The maximum value of stress constraint in this problem was 24979 (psi). The constraint violation was zero for the optimum solution. The convergence history of this example is presented in Figure 4. In addition, the proposed algorithm in this study achieved a higher convergence rate when compared with previous studies. The geometry of truss after optimization is shown in Figure 5.

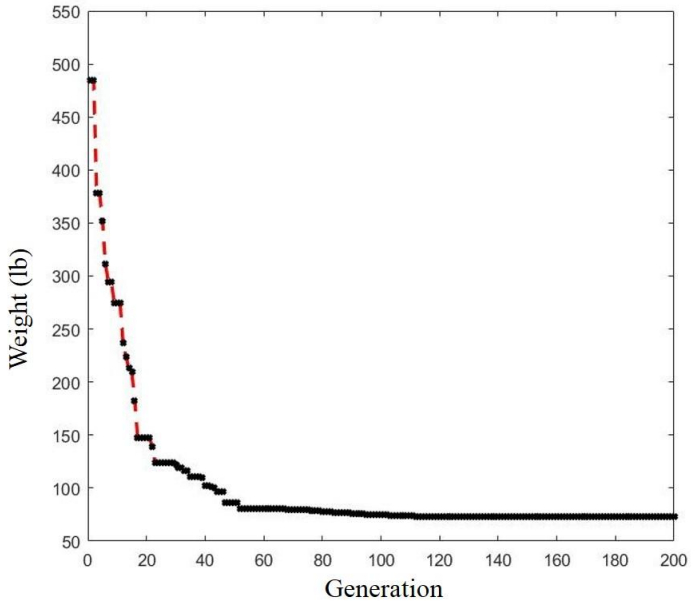

Figure 4. Convergence history for size and shape optimization of 15 bar truss.

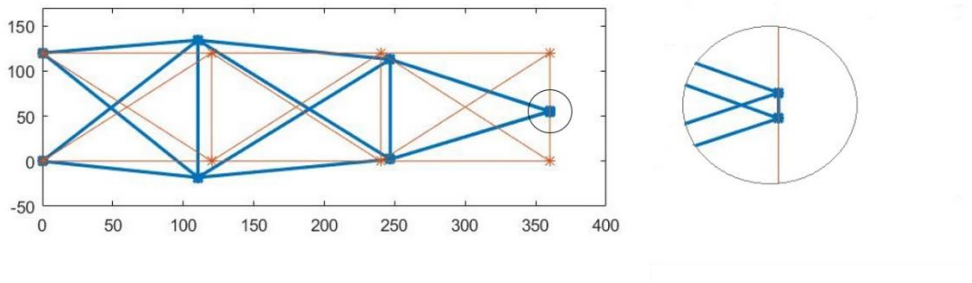

Figure 5. Geometry and optimal shape for size and shape optimization of 15-bar truss.

2.5.1.2 SIZE, SHAPE, AND TOPOLOGY OPTIMIZATION

The optimization of the size, shape, and topology is considered in this section. This is also a discrete-continuous optimization problem and the only different with the previous problem is that the truss element(s) can be removed as along as the structure remain stable and the design solution satisfies all the constraints. The numerical results are presented in Table 5 where the optimum solution of this study is compared with previous results found by Tang et al. (2005), Miguel et al. (2013), and Goncalves et al. (2015) using a genetic algorithm, a firefly algorithm, and a search group algorithm, respectively.

Table 5. Optimum size, shape, and topology solution for 15 bar planar truss.

Design Variables	**Tang 2005**	**Miguel 2013**	**Goncalves 2015**	**This work**
$A_1(in^2)$	1.081	0.954	0.954	0.954
$A_2(in^2)$	0.539	0.539	0.954	0.539
$A_3(in^2)$	0	0.141	0	0.174
$A_4(in^2)$	1.081	0.954	2.142	0.954
$A_5(in^2)$	0.954	0.539	1.081	0.539
$A_6(in^2)$	0.44	0.278	1.333	0.27
$A_7(in^2)$	0	0.141	0.111	0
$A_8(in^2)$	0.141	0	0.141	0
$A_9(in^2)$	0	3.813	0.374	0.22
$A_{10}(in^2)$	0.27	0.44	0.44	0.44
$A_{11}(in^2)$	0.27	0.44	0	0.44
$A_{12}(in^2)$	0.539	0.22	0.141	0.174
$A_{13}(in^2)$	0.141	0.22	1.488	0.22
$A_{14}(in^2)$	0.44	0.347	0.539	0.27
$A_{15}(in^2)$	0	0.141	0.111	0.174
$x_2\ (in)$	111.85	112.027	135.945	110.3891
$x_3\ (in)$	242.45	247.076	234.961	251.3094
$y_2(in)$	104.02	137.514	104.173	135.9221
$y_3(in)$	109.22	116.776	110.63	115.4637
$y_4\ (in)$	-	50.162	54.8032	58.092
$y_6\ (in)$	-10.82	-10.905	2.99213	-16.0743
$y_7(in)$	-11.13	-3.179	6.10237	4.3532
$y_8\ (in)$	48.84	48.825	46.0236	57.9002
Weight (lb.)	**77.84**	**74.33**	**123.452**	**70.4869**

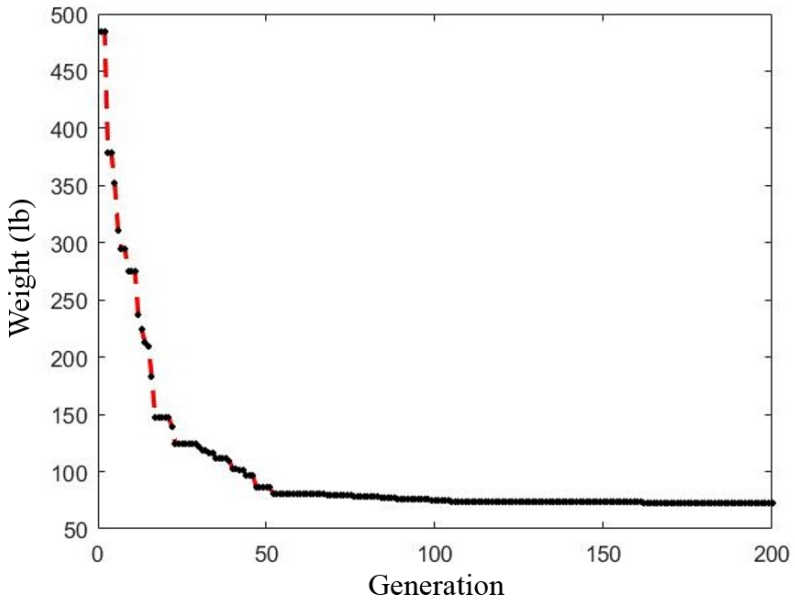

Figure 6. Convergence history for size, shape, and topology optimization of 15 bar truss.

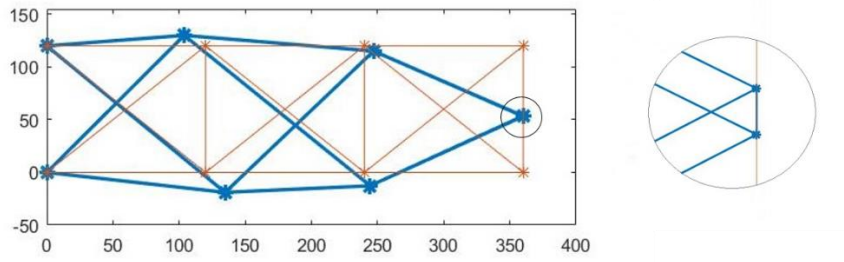

Figure 7. Geometry and optimal shape for size, shape, and topology optimization of 15 bar truss.

2.5.2 EIGHTEEN BAR PLANAR TRUSS

The ground structure of a planar truss is shown in Figure 8. Similar to the 15 bar planar truss structure, the objective function is to minimize the truss weight. The system is restricted within the stress and buckling constraints. More specifically, buckling constraints defined for those members of truss which are under compression. The absolute value of the stress for these members

should be less than corresponding buckling stress of those member ($\frac{kEA_i}{l_i^2}, i = 1, \dots, 18$). The cross-section arears of the members of the truss are put in four groups as follows:

1) $A_1 = A_4 = A_8 = A_{12} = A_{16}$
2) $A_2 = A_6 = A_{10} = A_{14} = A_{18}$
3) $A_3 = A_7 = A_{11} = A_{15}$
4) $A_5 = A_9 = A_{13} = A_{17}$

The coordinates x and y corresponding to the nodes 3, 5, 7, and 9 are taken as geometric variables. There are 4 discrete area variables and 8 nodal coordinate variables in this system. The stress limit (σ) is 20,000 (psi) for both tensile and compressive stresses of all members. Young's modulus (E) is specified as 1.0×10^7 (psi). The material density (ρ) is 0.1 $(\frac{lb.}{in^3})$. The buckling coefficient (k) is 4. Five vertical loads of 20,000 $(lb.)$ imposed on nodes 1, 2, 4, 6, and 8. The sections are taken from a profile list D of 80 sections starting with an area of 2.0 $(in.^2)$ increasing in the steps of 0.25 $(in.^2)$ to 21.75 $(in.^2)$. Side constraints for geometry variables are defined as following:

- $775\ (in.) \leq\ x_3 \leq 1225\ (in.)$
- $525(in.) \leq\ x_5 \leq 975(in.)$
- $275(in.) \leq x_7 \leq 725\ (in.)$
- $25(in.) \leq\ x_3 \leq 475(in.)$
- $75(in.) \leq y_3, y_5, y_7, y_9 \leq 1225(in.)$

The 18 bar planar truss problem is a size and shape optimization problem previously solved by several researcher (Hasancebi & Erbatur, 2002; Kaveh & Kalatjari, 2004; Rahami et al., 2008; Ho-Huu et al., 2015). The presented results are extracted from 100 independent runs of the algorithm. The optimal designs found by proposed algorithm and its comparison with those available in the literature is presented in Table 6.

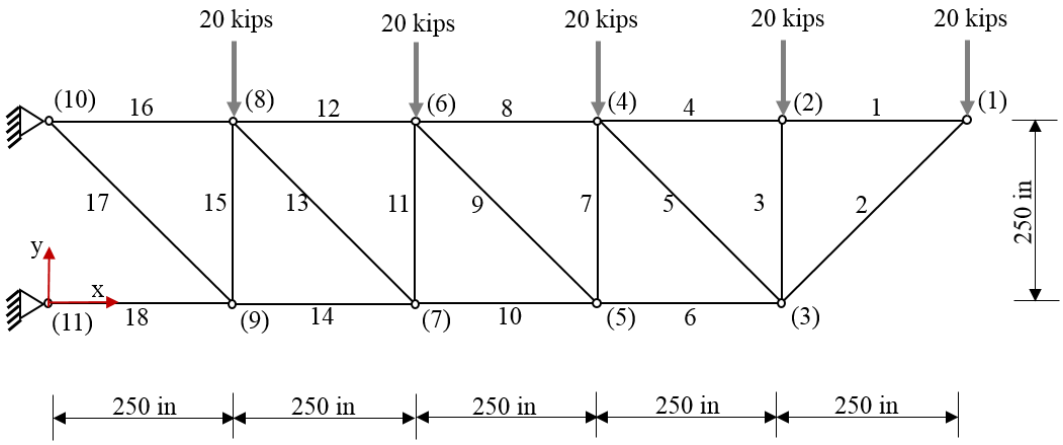

Figure 8. 18 bar planar truss.

Table 6. Optimum size, and shape solution for 18 bar planar truss.

Design Variables	**Hasancebi & Erbatur (2002)**	**Kaveh & Kalatjari (2004)**	**Rahami et al. (2008)**	**Ho-Huu et al. (2015)**	**This work**
$A_1(in^2)$	12.5	12.25	12.75	13	9.75
$A_2(in^2)$	18.25	18	18.5	17.5	18.75
$A_3(in^2)$	5.5	5.25	4.75	6.5	4.75
$A_5(in^2)$	3.75	4.25	3.25	3	3.5
x_3(in)	933	913	917.4475	914.06	928.421
y_3(in)	188	186.8	193.7899	183.46	201.0471
x_5(in)	658	650	654.3243	640.53	668.2497
y_5(in)	148	150.5	159.9436	133.74	163.8746
x_7 (in)	422	418.8	424.4821	406.12	434.023
y_7 (in)	100	97.4	108.5779	92.63	105.2616
x_9 (in)	205	204.8	208.4691	196.69	213.3768
y_3 (in)	32	26.7	37.6349	37.06	29.85221
Weight (lb.)	**4574.28**	**4547.9**	**4530.68**	**4554.29**	**4214.65**
Function Evaluation	-	-	8000	8025	6765

The optimal weight obtained in this study is $4214.65(lb.)$ which is achieved after 200 iterations. The algorithm proposed in this study not only obtained a lower minimum weight when compared to previous studies but also achieved the solution in fewer iterations. In other words, the

convergence rate is increased when compared with the previous studies. The best results in the aforementioned studies was obtained by Ho-Huu et al. (2015) which found the optimum weight of 4554.29 ($lb.$) after 250 iterations. The convergence history of this example is shown in Figure 9. The final shape of truss is depicted in Figure 10 where the optimum shape of truss evolves dramatically from the initial shape.

At the beginning of the truss, the optimal result has larger overall section because degree of freedom is constrained while section becomes smaller at the end of the truss. This is consistent with the previous result found in the literature for this example and is based on structural principals of loading resistance. Additionally, the optimum cross-sectional areas show that bars with greater size are subjected to greater loads.

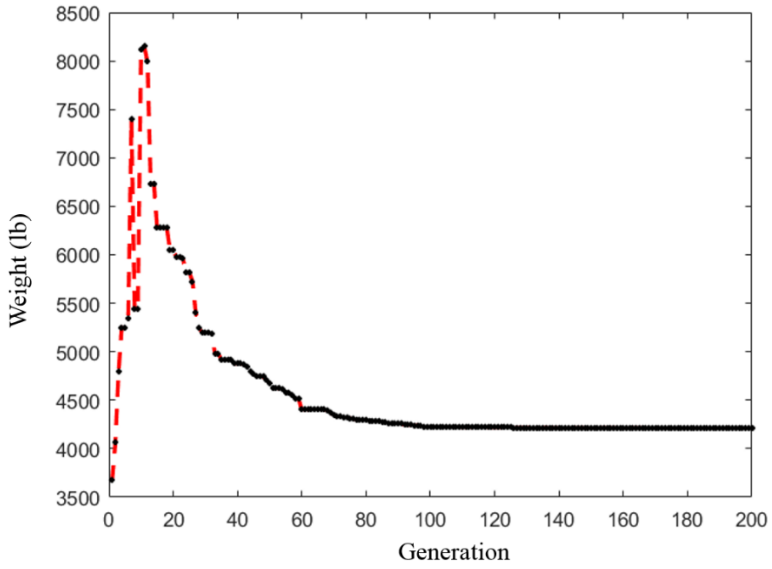

Figure 9. Convergence history for size and shape optimization of 18 bar truss.

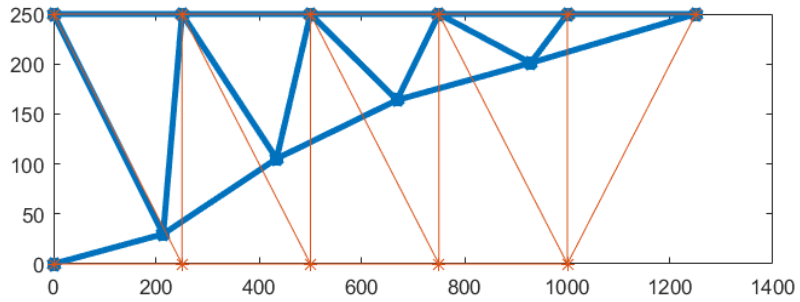

Figure 10. Geometry and optimal shape for size and shape optimization of 18 bar truss.

2.5.3 TWENTY FIVE BAR SPACE TRUSS

In this section, a space truss structure with 25 bar elements is optimized using the proposed algorithm. The initial geometry of a space truss together with the nodal numbering is shown in Figure 11. Members of this structure are categorized into 8 groups resulting in 8 discrete size variables:

1) A_1
2) $A_2 = A_3 = A_4 = A_5$
3) $A_6 = A_7 = A_8 = A_9$
4) $A_{10} = A_{11}$
5) $A_{12} = A_{13}$
6) $A_{14} = A_{15} = A_{16} = A_{17}$
7) $A_{18} = A_{19} = A_{20} = A_{21}$
8) $A_{22} = A_{23} = A_{24} = A_{25}$

Five geometry variables are defined for this problem as follows:

1) $x_4 = x_5 = -x_3 = -x_6$,
2) $x_8 = x_9 = -x_7 = -x_{10}$,
3) $y_3 = y_4 = -y_5 = -y_6$,
4) $y_7 = y_8 = -y_9 = -y_{10}$,
5) $z_3 = z_4 = z_5 = z_6$.

The stress limit (σ) is 40,000 (psi) for both tensile and compressive stresses for all members.

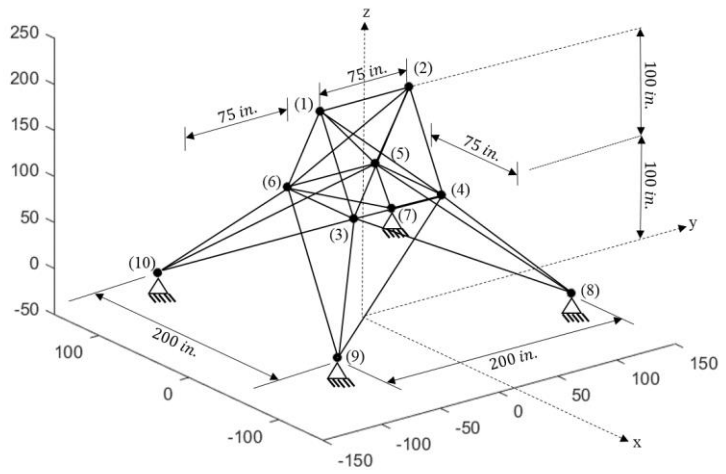

Figure 11. 25 bar space truss.

Young's modulus (E) is specified as 1.0×10^7 (psi). The material density (ρ) is 0.1 $(\frac{lb}{in^3})$. The displacements of all nodes are limited within 0.3 $(in.)$ range in all x, y, z directions. Seven loads are imposed on nodes 1, 2, 3, and 6. The details of the loading is provided in Table 7.

Table 7. Imposed nodal loads on 25 bar space truss.

Loads Node	$\boldsymbol{F_x(kips)}$	$\boldsymbol{F_y(kips)}$	$\boldsymbol{F_z(kips)}$
1	1.0	-10	-10
2	0.0	-10	-10
3	0.5	0	0
6	0.6	0	0

The sections are taken from a profile list of 30 sections, $D = (0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.8, 3.0, 3.2, 3.4)(in.^2)$. Side constraints for geometry variables are:

- $20\ (in.) \leq x_4 \leq 60\ (in.)$
- $40\ (in.) \leq x_8 \leq 80\ (in.)$
- $40\ (in.) \leq y_4 \leq 80\ (in.)$
- $100\ (in.) \leq y_8 \leq 40(in.)$
- $90(in.) \leq z_4 \leq 130(in.)$

The presented results are extracted from 100 independent runs of the algorithm similar to the other examples.

This size and shape optimization problem is previously solved using a Genetic Algorithm (GA) (Rajeev & Krishnamoorthy, 1992; Wu & Chow, 1995; Tang et al. , 2005; Rahami et al., 2008), as well as a Firefly Algorithm (Miguel et al., 2013), and a Differential Evolution (DE) model (Ho-Huu et al., 2015).

Since the cross-sectional areas are taken from a set of 30 discrete variables and the nodal coordinates are continuous, this problem is also a mixed variable optimization problem which deals simultaneously with integer and continuous design variables.

The GA proposed by Rahami et al. (2008) resulting in 10,000 objective function evaluations with the optimum result equal to $120.115\ (lb.)$. Miguel et al. (2013) found the optimum weight of $118.83(lb.)$ for this example after 6000 objective function evaluations. The optimum solution found by Ho-Huu et al. is $118.76\ (lb.)$ after 6000 objective function evaluations. The proposed algorithm found the optimum solution of $117.40\ (lb)$ which is lower than previous studies after only 50 iterations and the total number of objective function evaluations was almost 2310. The numerical details of this comparison is presented in Table 8.

Table 8. Optimum size, and shape solution for 25 bar space truss.

Design Variables	**Kaveh & Kalatjari (2004)**	**Tang et al. (2005)**	**Rahami et al. (2008)**	**Miguel et al. (2013)**	**Ho-Huu et al. (2015)**	**This work**
$A_1(in^2)$	0.1	0.1	0.1	0.1	0.1	0.1
$A_2(in^2)$	0.1	0.1	0.1	0.1	0.1	0.1
$A_3(in^2)$	1.1	1.1	1.1	0.9	0.9	1.0
$A_4(in^2)$	0.1	0.1	0.1	0.1	0.1	0.1
$A_5(in^2)$	0.1	0.1	0.1	0.1	0.1	0.1
$A_6(in^2)$	0.1	0.2	0.1	0.1	0.1	0.1
$A_7(in^2)$	0.1	0.2	0.2	0.1	0.1	0.1
$A_8(in^2)$	1.0	0.7	0.8	1	1.0	0.9
x_4(in)	36.23	35.47	33.0487	37.32	36.83	37.39
y_4(in)	58.56	60.37	53.5663	55.74	58.53	56.413
z_4(in)	115.59	129.07	129.9092	126.62	122.67	127.457
x_8(in)	46.46	45.06	43.7826	50.14	49.21	51.198
y_8 (in)	127.95	137.04	136.8381	136.40	136.74	139.49
Weight (lb.)	**124.0**	**124.943**	**120.115**	**118.83**	**118.76**	**117.40**
Function Evaluation		6000	6000	6000	6000	2310

Again, this example demonstrates the capability of the proposed algorithm in finding improved solution with a lower computational cost. The convergence rate is significantly increased when compared the previous studies while obtaining an improved optimal solution. The convergence history of the best solution is shown in Figure 12 and the geometry and optimal topology of the truss is shown in Figure 13.

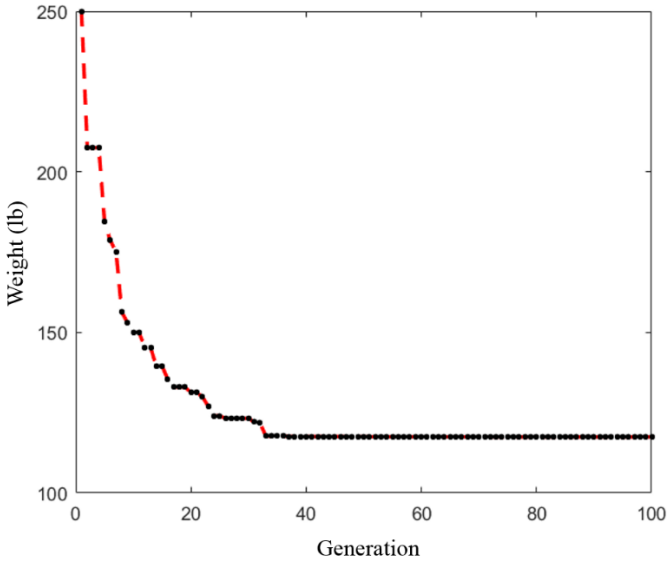

Figure 12. Convergence history for size and shape optimization of 25 bar truss.

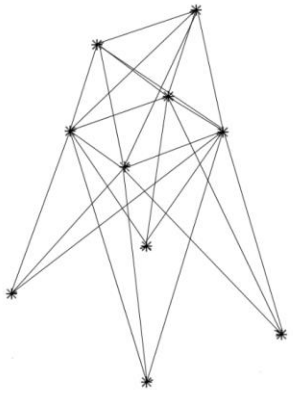

Figure 13. Geometry and optimal shape for size and shape optimization of 25 bar truss.

2.5.4 THIRTY NINE BAR SPACE TRUSS

This example is a size and shape optimization problem for a tower truss structure with 39 bar elements as shown in Figure 14. The truss is subjected to three vertical loads of 10 (kN) at three top nodes. The nodal coordinates are provided in Table 9. Members of this structure are categorized into 5 groups resulting in 5 discrete size variables as follows:

1) $A_1 = [(1,4), (2,5), (3,6)]$
2) $A_2 = [(4,7), (5,8), (6,9)]$
3) $A_3 = [(7,10), (8,11), (9,12)]$
4) $A_4[(10,13), (11,14), (12,15)]$
5) A_5 for the remaining elements

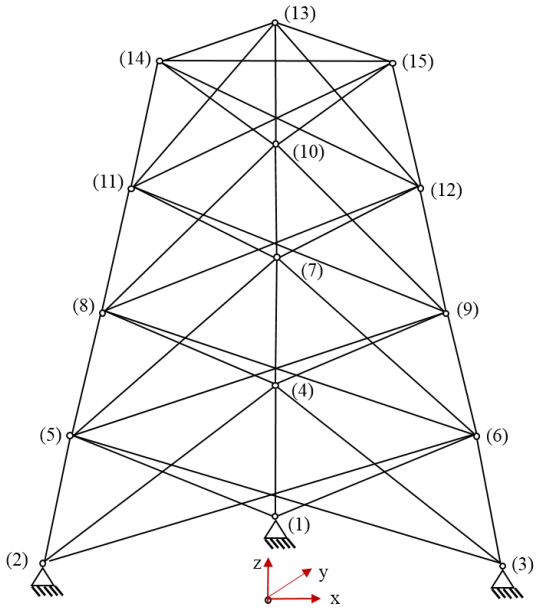

Figure 14. 39 bar space truss.

There are 6 geometry variables: $z_4, y_4, y_7, z_7, y_{10}$, and z_{10}. Displacement of node 13 is limited within 4 (mm) in y directions. The tensile stresses are constrained to remain under 240 (MPa) for all elements. Young's modulus (E) is specified as 210 (MPa). The material density (ρ) is 7800 $(\frac{kg}{m^3})$. The sections are taken from a profile list of sections, D =(0.1, 0.2, …, 13)(cm^2). Side constraints for geometry variables are defined below:

- $0.28\, m \leq\ z_4 \leq 1\, m$
- $0.0\, m \leq\ y_4 \leq 2\, m$
- $0.28\, m \leq\ y_7 \leq 1m$

- $1\,m \leq z_7 \leq 3\,m$
- $0.28\,m \leq\ y_{10} \leq 1\,m$
- $2\,m \leq\ z_{10} \leq 4\,m$

Table 9. Nodal coordinates of bottom and top nodes of 39 bar space truss.

Bottom Nodes				**Top Nodes**			
Number	$x(m)$	$y(m)$	$z(m)$	Number	$x(m)$	$y(m)$	$z(m)$
1	0	1	0	**13**	0	0.28	4
2	$-\frac{\sqrt{3}}{2}$	0.5	0	**14**	$\frac{-0.42}{\sqrt{3}}$	-0.14	4
3	$\frac{\sqrt{3}}{2}$	0.5	0	**15**	$\frac{0.42}{\sqrt{3}}$	-0.14	4

The comparison of the optimal designs with those of other references is provided in Table 10. The comparison of the results confirms that the proposed DE algorithm for mixed discrete–continuous 39-bar space truss outperforms previous methods in finding the minimal structure weight. Ho-Huu et al. (2015) found optimum objective value of 140.35(kg) after 1140 objective function evaluation while the proposed algorithm reached 138.9853(kg) after 825 objective function evaluations. The proposed algorithm continued its search and reached the optimum weight of 130.7979(kg). This is show that the proposed algorithm has a good search capability while maintain a good convergence rate. The result shows considerable changes of the topology compared with original design. The Convergence history for size and shape optimization is depicted in Figure 15.

Table 10. Optimum size, and shape solution for 39 bar space truss.

Design Variables	Wang et al. (2002)	Shojaee et al. (2013)	Ho-Huu et al. (2015)	This work
$A_1(cm^2)$	11.01	10.12	13	12.7
$A_2(cm^2)$	8.63	9.91	12.9	9.5
$A_3(cm^2)$	6.69	8.56	9	6.3
$A_4(cm^2)$	4.11	3.92	2.7	2.1
$A_5(cm^2)$	4.37	3.44	1.6	1.5
y_4(m)	0.805	0.6683	0.9549	0.8129
z_4(m)	1.186	1.9	0.8589	1.6079
y_7(m)	0.654	0.4732	0.9258	0.6429
z_7(m)	2.204	2.8734	2.0154	2.6727
y_{10} (m)	0.466	0.3002	0.7160	0.3991
z_{10} (m)	3.092	3.4415	3.1011	3.5327
Weight (kg)	**203.18**	**176.834**	**140.35**	**130.7979**

2.5.5 ELEVEN BAR PLANAR TRUSS

The 11-bar, 6-node truss shown in Figure 16 has been widely published in the optimization literature. The optimization of its size, shape and topology is considered in this study. In this example, applied load on nodes 5 and 6 is equal to 100 (kip). Stress is constrained to be less than 25 (ksi). Vertical displacements at joints 2 and 4 were constrained to less than 2(in). The modulus of elasticity and the material density are 10,000 (ksi) and 0.1(lb/in^3), respectively. The sizing variables, cross-sectional areas are discrete and are taken from a set of 32 discrete values $D =$ (1.62, 1.80, 2.38, 2.62, 2.88, 3.09, 3.13, 3.38, 3.63, 3.84, 3.87, 4.18, 4.49, 4.80, 4.97, 5.12, 5.74, 7.22, 7.97, 11.50, 13.50, 13.90, 14.20, 15.50, 16.00, 18.80, 19.90, 22.00, 22.90, 26.50, 30.00, 33.50) (in^2) (Rajan, 1995).

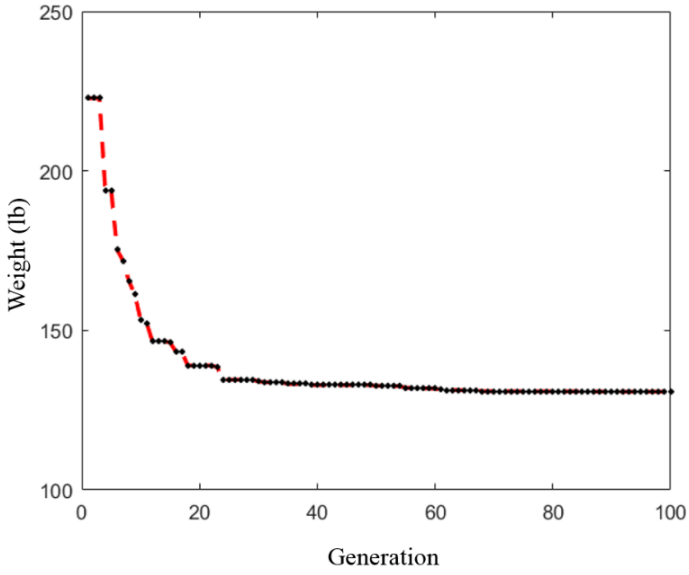

Figure 15. Convergence history for size and shape optimization of 39 bar truss.

Shape is optimized by allowing the vertical coordinates of joints 1, 3, and 5 to move between 180 $(in.)$ and 1000 $(in.)$. Topology is optimized by allowing all members to be removed except member 3 (between nodes 4 and 5) and member 4 (between nodes 5 and 6), see Figure 16.

The optimum result found by proposed DE algorithm is 2733.5$(lb.)$ after almost 7665 objective function evaluations. The best result in literature is found by Miguel et al. (2013) where the optimum structure weight is found 2705.16$(lb.)$ after 50,000 objective function evaluations. The comparison of the numerical results of this work and the previous studies are presented in Table 11.

The convergence rate of the proposed DE algorithm is significantly greater than other methods although the objective function value is not better than the best result by Miguel et al. (2013). Note that this study has reduced the required number of objective function evaluations

considerably (i.e., number of structural analysis) which translate into considerably lower computational cost.

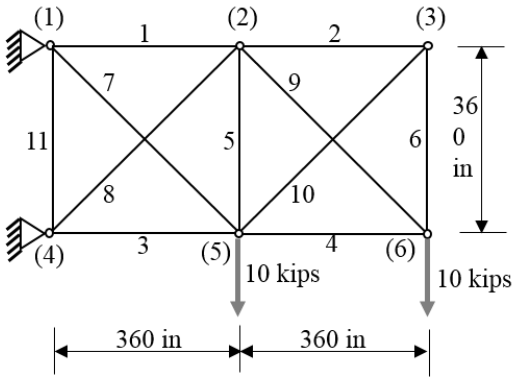

Figure 16. 11 bar planar truss.

Table 11. Optimum size, and shape solution for11 bar planar truss.

Design Variables	**Rajan (1995)**	**Balling et al. (2006)**	**Martini (2011)**	**Miguel et al.(2013)**	**This work**
$A_1(in^2)$	9.9	-	-	11.5	13.9
$A_2(in^2)$	9.4	-	-	0	0
$A_3(in^2)$	11.5	-	-	11.5	11.5
$A_4(in^2)$	1.5	-	-	7.22	7.22
$A_5(in^2)$	0	-	-	0	0
$A_6(in^2)$	12	-	-	0	0
$A_7(in^2)$	11.5	-	-	5.74	5.74
$A_8(in^2)$	3.6	-	-	2.88	2.13
$A_9(in^2)$	0	-	-	13.5	13.5
$A_{10}(in^2)$	10.4	-	-	0	0
$A_{11}(in^2)$	0	-	-	0	0
y_1(in)	186.5	-	-	-	598.1636
y_3(in)	554.5	-	-	-	460.2370
y_5(in)	786.9	-	-	-	768.7939
Weight (lb.)	**3254.0**	**2736**	**2900**	**2705**	**2733.5**
Function Evaluation	-	500,000	4075	50,000	7665

The objective function value presented by Martini (2011) is greater than this work, but their number of objective function evaluations is smaller. The convergence history of this work is presented in Figure 17 and the optimal shape and geometry of the 11 bar truss is shown in Figure 18.

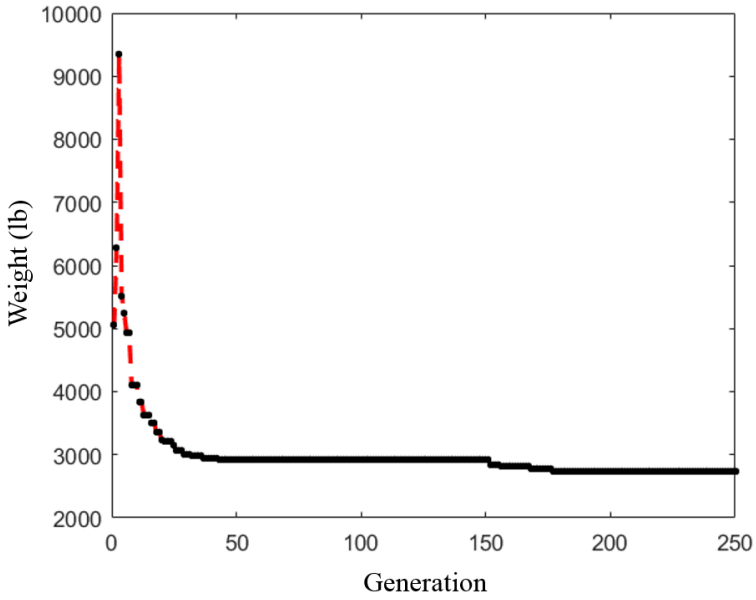

Figure 17. Convergence history for size, shape, and topology optimization of 11 bar truss.

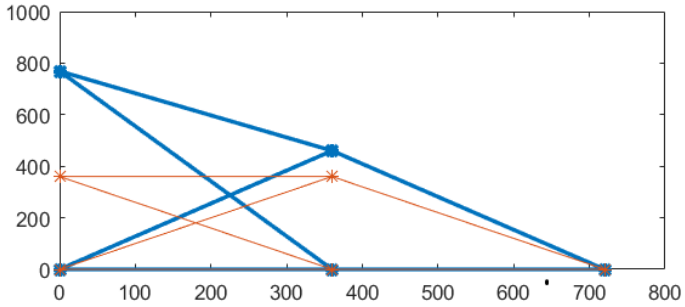

Figure 18. Geometry and optimal shape for size, shape, and topology optimization of 11 bar truss.

2.6 CONCLUSION

A new variant of differential evolution algorithm for solving discrete-continuous design variables by incorporation a modified version of Improved $(\mu + \lambda)$Differential Evolution and Improve Adaptive Trade of Method (IATM) is proposed. The effectiveness and robustness of presented algorithm were examined by solving several benchmark problems from the literature. Both shape and size, and shape, size, and topology optimization problems are considered for evaluation of this study. In numerical examples, both cross sectional areas and structural shapes were adjusted simultaneously to find the minimum structural weight under specified conditions. The benchmark problems are subjected to various combination of stress, displacement, and buckling constraints.

The numerical analysis confirmed that the proposed algorithm is capable of finding the optimum solution with a considerably improved convergence rate when compared to the existing methods. The improved convergence rate directly effects the required number of structural analysis to find the optimum solution of the truss structure. Structural analysis is one of the most time consuming and computationally expensive part of the truss optimization. Not only the proposed algorithm finds the global minimum faster but also in all but one of the studied cases it outperformed other methods in finding the minimum structural weight. The rate of the convergence was significantly higher for the case that obtained optimum weight was slightly higher than the best found in literature. The proposed algorithm was converged 84.67 percent faster and the objective function was only higher by one percent. The promising results founded in this work is a motivation for further development of the proposed algorithm and expansion of its application to more complicated real-life engineering problems.

CHAPTER 3

BACKWARD DIJKSTRA ALGORITHMS FOR FINDING THE DEPARTURE TIME BASED ON THE SPECIFIED ARRIVAL TIME FOR REAL-LIFE TIME-DEPENDENT NETWORKS

3.1 INTRODUCTION

Most people must commute from home to work, and wonder if they leave their homes at a specific time when they will arrive at work. They may also wonder what time they should depart home to arrive at work at a specific time. Similar questions have been asked by long distance travelers.

The vast majority of the literature found on Shortest Path Problem (SPP) has dealt with static (i.e., non-time-dependent) networks that have fixed topology and constant link costs. In recent years, there has been a renewed interest in the study of Time-Dependent Shortest Path Problems (TDSPP). One of the fundamental network problems in TDSPP is the computation of the shortest paths from all departure nodes to a set of destination nodes, for all possible departure times. Obviously, this problem should be solved in a given time-dependent network.

TDSPP first developed by Cooke and Halsey (1966) to find the shortest travel time from a given source node at a certain time to a given destination node. Orda and Rom (1990) presented an algorithm for finding the shortest path and minimum delay under various waiting constraints, and for all instances of time. They also investigated the properties of the derived paths under arbitrary functions for link delays with possible non-FIFO behavior. The FIFO stands for “First In

First Out" and is also called the *non-overtaking property* (Nannicini & Liberti, 2008), because it states that if T_1 leaves node i at time t_1 and T_2 leaves the same node at time $t_2 > t_1$, then the T_2 cannot arrive at node *j* before T_1. A time dependent vehicle routing problem proposed with non-FIFO property proposed by Malandraki and Daskin (2001). Daganzo (2002) solved the backward SPP on a network with FIFO links. Chabini and Ganugapati (2002) proposed an efficient dynamic solution algorithm, (DOT), and prove that no sequential algorithm with a superior worst-case computational complexity can be developed. The also developed a time-based parallel version of DOT for the case of minimum time paths in FIFO networks. Wuming and Pingyang (2007) introduced an algorithm to solve the shortest paths in time-dependent network by converting non-FIFO network to a FIFO network and solved the problem using the traditional SPP algorithms. Ding et al. (2008) proposed a new Dijkstra-based algorithm by decoupling path-selection and time-refinement in the starting-time interval *T*. Their algorithm can handle both FIFO and non-FIFO time-dependent graphs. They also established the time complexity and space complexity based on their proposed 2 steps approached. Through extensive numerical studies, they also concluded that their dynamic algorithm outperforms existing solution algorithms in terms of efficiency. Computational strategies for families of Frank-Wolfe (FW), Conjugate FW, Bi-conjugate FW, Deterministic User Equilibrium (DUE) algorithms for static networks were also developed by Allen (2013).

The focus of this study is to find the departure time at the source node(s) for a specified arrival time at the destination node(s) in FIFO, and non-FIFO networks. This present work consists of development of a sparse matrix storage scheme for efficiently storing large scale sparse network's connectivity. In addition, the concept of Time Delay Factor (TDF) is combined with a general piece-wise linear function to describe the non-FIFO link's costs as a function of time.

Furthermore, in this study backward Dijkstra SP algorithm with simple heuristic rules is presented for rejecting unwanted solutions during the search. Note that this work was published in the Journal of Applied Mathematics and Physics (Bakhtyar et al., 2016).

The remaining of this Chapter is organized as follows. Dynamic networks are discussed in Section 3.2, where the concept of TDF in conjunction with piece-wise linear time function for the links' costs are introduced. A simple but meaningful numerical example is solved in Section 3.3. The solution details of this numerical example facilitate the discussions of the Polynomial LCA and Forward Dijkstra algorithms for finding the arrival time at the destination node for a given departure time. Furthermore, this same example will also be used in Section 3.3 for finding the departure time at the source node in order to arrive at the destination node at a given time. The possibility of finding multiple or a single solution for this problem is discussed in Section 3.3. Real-life, large-scale dynamic networks are investigated using the proposed time-dependent Backward Dijkstra algorithm, and the numerical results are presented in Section 3.4 to validate the proposed dynamic algorithm. Finally, the conclusion is presented in Section 3.5.

3.2 TIME DELAY FACTOR AND PIECE-WISE LINEAR FUNCTION IN DYNAMIC NETWORKS

Unlike static networks, in a dynamic network the time spent to travel from a node to another is not constant. The actual travel time depends on the departure time. In this work, the following formulas are employed for a typical link k, connecting node i to node j.

$$AT = DT + CST_{ij} \times TDF(DT) \tag{55}$$

where AT is Arrival Time at node j , DT is the Departure Time at node i, CST_{ij} is the constant static time for link k, and $TDF(DT)$ stands for Time Delay Factor (TDF) defined by Equation (56).

$$TDF(DT) = 1 + y(DT) \tag{56}$$

Note that *TDF* is a function of time function y therefore depends on the *DT* as described by Equation (56). In this work, the function y for a typical link is defined as a time dependent, piece-wise linear function which is depicted in Figure 19. In real dynamic networks, the travel time will be increased during certain hours of the day. For instance, during the morning and afternoon rush hours, within 6 hours-8 hours (6am-8am) and 16 hours-18 hours (4:00pm-6:00pm) time frames.

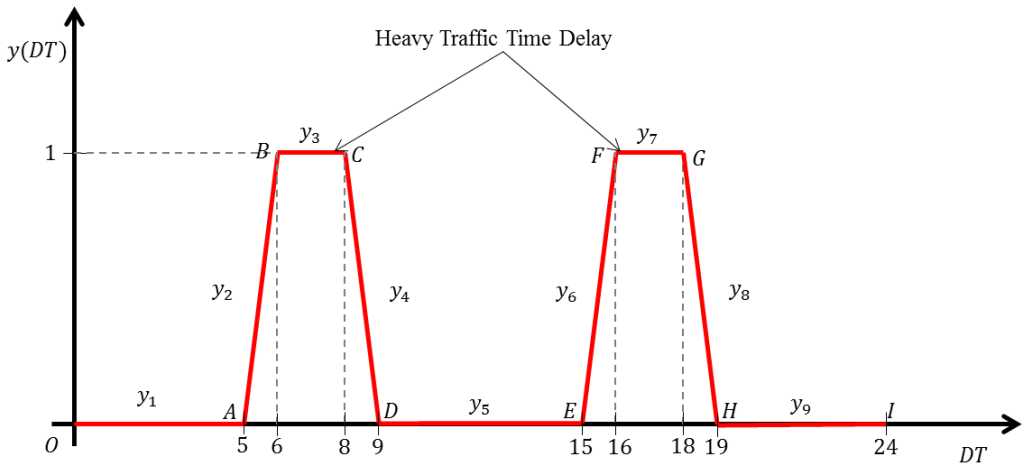

Figure 19. Piece-wise linear time function for a typical link *k*.

In Figure 19, the function $y(DT)$ is defined as follow:

- $y(DT) = y_1(DT)$ when $DT \in [0.00, 5.00]$ hours.
- $y(DT) = y_2(DT)$ when $DT \in [5.00, 6.00]$ hours.

- $y(DT) = y_3(DT)$ when $DT \in [6.00, 8.00]$ hours.
- $y(DT) = y_4(DT)$ when $DT \in [8.00, 9.00]$ hours.
- $y(DT) = y_5(DT)$ when $DT \in [9.00, 15.00]$ hours.
- $y(DT) = y_6(DT)$when $DT \in [15.00, 16.00]$ hours.
- $y(DT) = y_7(DT)$when $DT \in [16.00, 18.00]$ hours.
- $y(DT) = y_8(DT)$ when $DT \in [18.00, 19.00]$ hours.
- $y(DT) = y_9(DT)$when $DT \in [19.00, 24.00]$ hours.

This piece-wise linear time function can be conveniently provided by the end-user to consider the variations of congested traffic hours. Thus, the coordinates $(DT,\ y(DT))$ of defining points in Figure 19 such as O, A, B, C, D, E, F, G, H, and I are considered input parameters provided by the end-user.

It is possible and might be necessary to define a separate $y(DT)$ function for each link. However, in this study it is assumed that all links that exist in the network have the travel behavior presented in Figure 19.

The value of $y(DT)$ is zero in a static network while it varies between 0.00 to 1.00 in a dynamic network as indicated in Figure 19. Thus, for static networks, the TDF defined in Equation (56) is equal to 1, while in a dynamic network, the value of TDF could vary within the range of [1.00 – 2.00]. The following two important observations can be made:

1) On a typical link, if the **departure time** at starting node is known, then the arrival time at ending node can be uniquely and easily computed using Equation (55), Equation (56), and Figure 19.

2) On a typical link, if the **arrival time** at ending node is known, then departure time at starting node can be computed using Equation (55), Equation (56), and Figure 19.

The computed departure time in the second observation may not be unique. Therefore, it is necessary to develop a heuristic elimination rule to obtain an acceptable single solution.

3.3 FINDING THE DEPARTURE TIME BASED ON THE SPECIFIED ARRIVAL TIME

In order to study the performance of the proposed method, three problems are considered in this section. All three problems are developed to study a dynamic network with 5 nodes and 9 links as shown in Figure 20. It is also assumed that all links have the time function illustrated in Figure 19.

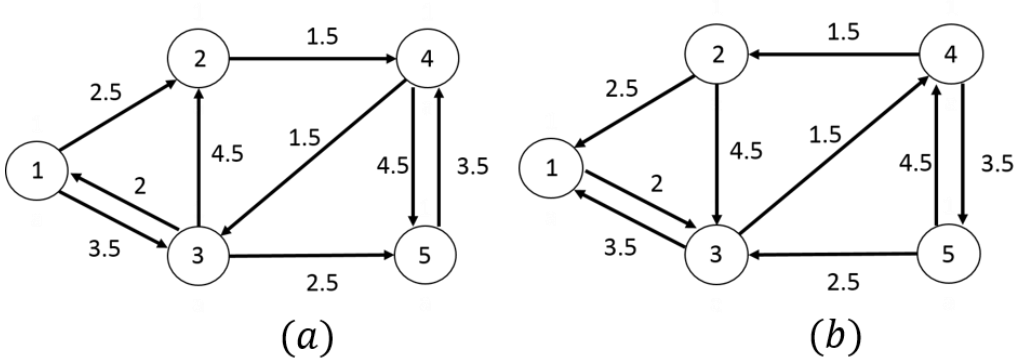

Figure 20. (a) A dynamic network topology, (b) A dynamic reversed network topology

Problem 3.1 Use the polynomial LCA method to find the time dependent shortest path from any source node, say $s = 5$ to any destination node, say $t = 2$ at the following three possible departure time:

Case (a): 9 hours = 9:00 am (to simulate right after rush hours)

Case (b): 15 hours = 3:00 pm (to simulate right before rush hours)

Case (c): 16.75 hours = 4:45 pm (to simulate during rush hours)

This problem is rather straight forward, since the departure time (DT) is known at any source node 5. For any subsequent link:

- The function $y(DT)$ is uniquely defined using the data presented in Figure 19.
- The Time Delay Factor (TDF) is uniquely determined using Equation (56).
- The arrival time (AT) at the targeted destination node is uniquely determined using Equation (55).

Following the above mentioned process the values of the arrival time (AT) at node 2 was calculated for all cases a, b, and c which are presented in Table 12.

Table 12. Numerical Results for Dynamic Network in Figure 20.

Case	Source Node	Destination Node	Departure Time	Arrival Time	Shortest Time (Cost)	Path	Number of Explored Nodes
Polynomial LCA & Forward Dijkstra							
a	5	2	9	16	7	5→3→2	5
b	5	2	15	24	9	5→3→1→2	5
c	5	2	16.75	26.25	9.5	5→3→2	5
Backward Dijkstra							
a	2	5	9	16	7	2→3→5	4
b	2	5	15.5714	24	8.4286	2→3→5	4
c	2	5	19.25	26.25	7	2→3→5	4

Problem 3.2. Re-do problem 1 for all cases a, b, and c, but using the time dependent regular forward Dijkstra algorithm.

The numerical results obtained using time-dependent regular forward Dijkstra algorithm were identical to those obtained in Problem 1 using time-dependent Polynomial LCA algorithm. The result for this problem also presented in Table 12.

Problem 3.3. Find the departure time for the *known arrival time* using dynamic backward Dijkstra algorithm for all three cases of the previous problem.

Case (a): 16 hours = 4:00pm

Case (b): 24 hours = 0:00 am (midnight)

Case (c): 26.25 hours= 2:15 pm

It is necessary to use the proposed modified dynamic backward Dijkstra algorithm to solve this problem. This algorithm can be utilized in two major steps:

Step 1. Revised the links' direction of the given network, as shown in Figure 20b.

Step 2. Find the departure time to *arrive* at the *destination* at a specified time.

The arrival times found in Problems 3.1 and 3.2 can be used as the known departure time at the source node 2. To solve this problem, one can still use Equation (55). However, the known variables in this problem are AT and CST_{ij}, and the unknown variable is DT. This is completely different from the defined problems 1 and 2, where the known variables are DT and CST_{ij}, and the unknown variable is AT. A unique value for the unknown variable AT can be easily found from Equation (55) in Problems 3.1 and 3.2. However, in Problem 3.3, it is challenging to find a unique value for variable DT using Equation (55). Combining Equation (55) and Equation (56), one obtains the following equation:

$$DT = AT - CST_{ij} \times \big(1 + y_r(DT)\big), r = 1, \dots, 9 \quad (57)$$

Note that the only unknown in Equation (57) is the departure time (DT).

To further clarify the application of Equation (57), the case (b) of the Problem 3.3 is solved next. The arrival time (AT) at node 2 is 24 hours (midnight).

The first iteration starts by initializing the distance vector, the predecessor vector, and the array of explored nodes, S, for the starting node 2 .

$$distance = \{\ Inf \quad 0 \quad Inf \quad Inf \quad Inf\ \} \tag{58}$$

$$predecesor = \{\ 0 \quad 0 \quad 0 \quad 0 \quad 0\} \tag{59}$$

$$S = \{2\} \tag{60}$$

Next, all the out-going links from node 2 are analyzed based on Figure 20b.

- For outgoing link 2-1 set:

$$AT = 24.00 \tag{61}$$

$$CST_{21} = 2.5 \tag{62}$$

Using Equation (57), the value of each departure time (DT) corresponding to the time function shown in Figure 19 can be calculated as follows:

$$\begin{aligned}&\{DT_1 \quad DT_2 \quad DT_3 \quad DT_4 \quad DT_5 \quad DT_6 \quad DT_7 \quad DT_8 \quad DT_9\}\\&= \{21.5 \quad 9.71 \quad 19 \quad 0.67 \quad 21.5 \quad 16.9 \quad 19 \quad 17.3 \quad 21.5\}\end{aligned} \tag{63}$$

Since $y_r(DT), r = 1, \dots, 9$, should be in a specified range (see explanation of Figure 19), eight of nine computed Departure Time (DT) must be rejected. The only acceptable Departure Time is $DT = DT_9 = 21.5$ hours, with the value $y(DT_9) = 0.00$, which correspond to the $TDF = 1.0$. Then, the travel information is updated:

$$distance\ (1) = distance\ (S\ (end)) + CST_{21} \times TDF \tag{64}$$

$$distance = \{\ 2.5 \quad 0 \quad Inf \quad Inf \quad Inf\ \} \tag{65}$$

$$predecesor = \{\ 2 \quad 0 \quad 0 \quad 0 \quad 0\} \tag{66}$$

- For outgoing link 2-3 set:

$$AT = 24.00 \tag{67}$$

$$CST = 4.5 \tag{68}$$

Then, Equation (57) is used to obtain the departure time (DT) values as follows:

$$\{DT_1 \quad DT_2 \quad DT_3 \quad DT_4 \quad DT_5 \quad DT_6 \quad DT_7 \quad DT_8 \quad DT_9\} = \{19.5 \quad 7.6 \quad 15 \quad 6 \quad 19.5 \quad 15.8 \quad 15 \quad 18.9 \quad 19.5\} \tag{69}$$

Only three out of nine computed Departure Time values can satisfy the requirements: $DT_6 = 15.8$, $DT_8 = 18.9$, and $DT_9 = 19.5$.

The corresponding values for the time function and time delay factor are:

$$\{y_6 \quad y_8 \quad y_9\} = \{0.8182 \quad 0.1429 \quad 0.00\} \tag{70}$$

$$\{TDF_6 \quad TDF_8 \quad TDF_9\} = \{1.82 \quad 1.14 \quad 1.00\} \tag{71}$$

Among these three possible solutions the one with the *largest* value ($DT = DT_9 = 19.5$) is selected. This choice also corresponds to the *smallest* value of time delay factor ($TDF = TDF_9 = 1.00$).

By selecting the *smallest* value of time delay factor (TDF) the *smallest* travel cost is picked for this particular link.

The problem data are updated accordingly:

$$DT = 19.5 \tag{72}$$

$$distance(3) = distance\big(S(end)\big) + CST_{23} \times TDF \tag{73}$$

$$distance = \{\ 2.5 \quad 0 \quad 4.5 \quad Inf \quad Inf\ \} \tag{74}$$

$$predecesor = \{\ 2 \quad 0 \quad 2 \quad 0 \quad 0\} \tag{75}$$

The next node to explore is node 1 so the second iteration can start by searching toward all the outgoing links from node 1 in which the arrival time at node 1 is 21.5 ($AT = 21.5$) , and $S = \{2 \quad 1\}$. The algorithm will stop when the next node to explore is the destination node.

The arrival time (AT) at node 5 for all cases a, b, and c of the Problem 3.3 were found, and presented in Table 12. Thus, for certain dynamic networks, there may be more than one solution for the departure time at source node which still yields the same specified arrival time at a

destination node. By using the suggested criterion to select the value of DT, the resulted path will also often correspond to the shortest path.

3.4 NUMERICAL RESULTS AND DISCUSSIONS

In this section, 12 large-scale examples based on real-life networks data were solved using the regular forward Dijkstra, and the heuristic backward Dijkstra (the proposed algorithm) algorithms. The regular forward Dijkstra algorithm was employed to find the arrival time at the destination node, based on the known departure time at the source node. The heuristic backward Dijkstra algorithm was employed to find the departure time at the source node, based on the known (specified) arrival time at the destination node.

For cases where multiple solutions for DT exists, the departure time (DT) which produces the smallest value of time function ($y(DT) = y_{min}$) was selected. This selection also corresponds to the smallest value of time delay factor ($TDF = TDF_{min}$). This is the criterion which has been used in Section 3.3.

To make the process more convenient, the arrival time at the destination node of the Forward Dijkstra algorithm was used as the departure time for the destination node of the Backward Dijkstra algorithm, for the same network with reversed links' directions. All numerical results are tabulated in Table 13.

Table 13. Comparisons of Forward and Backward Dijkstra Results for Real Networks.

Example	Network Name	Source w.r.t. Forward Search	Destination w.r.t. Forward Search	Forward Search			Backward Search (Y_{min})	
				Departure Time (hours)	Arrival Time (hours)	Cost	Back Calculated Departure Time (hours)	Cost
1	Winnipeg	5	100	6	16.494	10.494	6	10.494
2	Winnipeg	25	110	6	21.764	15.764	7.236	14.528
2	Winnipeg	25	110	7.236	21.764	14.528		
3	Barcelona	5	400	6	10.587	4.5876	6.0002	4.587
4	Barcelona	15	400	5	11.954	6.954	5	6.954
5	Austin	56	1800	1	22.855	21.855	1	21.855
6	Austin	156	1500	6	18.735	12.735	6.0007	12.734
7	Austin	5	6100	23	53.041	30.041	23	30.041
8	Austin	1	7388	6	22.797	16.797	5.9993	16.797
9	Philadelphia	6	560	1	13.481	12.481	1	12.481
10	Philadelphia	36	510	7	22.7	15.7	6.9996	15.700
11	Philadelphia	48	1415	1	63.352	62.352	1.5262	61.826
11	Philadelphia	48	1415	1.526	63.352	61.826		
12	Philadelphia	100	1429	6	57.165	51.165	6.0001	51.165
13*	Winnipeg	25	110	6	25.020	19.020	6	19.020
14*	Philadelphia	48	1415	1	199.32	198.32	1	198.32

* y=1 (for example 1 through 12, y is found by using function introduced in Figure 19)

For the problem of finding the departure time at the source node(s) based on the specified arrival time at the destination node(s), and based on the numerical results presented in Table 13, the following major observations can be made:

a) Unique solutions were found in all examples except examples 2 and 11.

b) Multiple solutions were found in examples 2, and 11 which is not surprising. In example 2, if the driver departs at the source node 25 at either 6.00 hours, or at 7.236 hours, he/she still arrives at the destination node 110 at the specified time (21.7647 hours). Therefore, this situation is called non-overtaking based on the definition of FIFO property. FIFO property, also called non-overtaking property, means if T_1 leaves node i at time t_1 and T_2 leaves the

same node at time $t_2 > t_1$, then T_2 cannot arrive at node *j* before T_1. Non-overtaking situation also is observed for example 11 (e.g., if the driver departs at the source node 48 at either 1.00 hours, or at 1.5262 hours, he/she still arrives at the destination node 1415 at the specified time 63.3532 hours.)

An example was solved in a study published in 2007 by Wuming and Pingyang to show the difference between FIFO and non-FIFO properties in a small network. The different departure time (i.e., 0, 1, 2, and 3) from a specified source node was examined in the mentioned example. For the non-FIFO case, when the departure time at the source node was 0, the arrival time at the destination node was 8 ($DT_1 = 0, AT_1 = 8$). In addition, when the departure time was 1, the arrival time was $23/4$ ($DT_2 = 1, AT_2 = 23/4$). This situation is called overtaking (i.e., when one departed from the source node sooner ($Dt_1 = 0 < DT_2 = 1$) arrived at the destination later ($AT_1 = 8 > AT_2 = 23/4$).) which happens in non-FIFO network. They also proposed a method to convert non-FIFO into FIFO property. For the FIFO case, when the departure time at the destination node was 0 or 1, the arrival time was $23/4$, ($DT_1 = 0$ or $1, AT_1 = 23/4$), so unlike the non-FIFO case there is no overtaking situation. Also, when the departure time was 2, the arrival time was $27/4$, ($DT_2 = 2, AT_2 = 27/4$), which is less than $23/4$ ($DT_1 = 0\ or\ 1 < DT_2 = 2, AT_1 = 23/4 < AT_2 = 27/4$) which is expected in a FIFO network.

Since in examples 2 and 11, there were no overtaking situation therefore the FIFO property was satisfied. Consequently, Dijkstra algorithm still can be implemented efficiently to find the optimum solution in a time dependent algorithm using the piece-wise linear function introduced in Figure 19.

3.5 CONCLUSION

In this study, the well-known polynomial LCA, and the Regular Forward Dijkstra algorithms have been applied to dynamic networks, through the concept of piece-wise linear function and Time Delay Factor (TDF) which is a function of the departure time (DT) at the source node for a typical link.

The practical problems of finding the departure time at the source node(s) based on the specified arrival time at the destination node(s) can be efficiently solved by using the proposed Backward Dijkstra algorithm, which basically employs the Forward Dijkstra algorithm on the same dynamic network with all links' direction are reversed.

Extensive numerical results based on a small-scale (academic) dynamic network (with 5 nodes, and 9 links), as well as using 12 real-life (large-scale) dynamic networks, seem to indicate that:

I. The proposed time dependent Backward Dijkstra algorithm always find the correct departure time at the source node and guarantees to arrive at the destination node at the specified arrival time.

II. Most of the time, the computed paths correspond to the shortest paths, and the solution is unique.

III. The computed paths often correspond to the shortest paths, although SP is not a requirement for the type of time-dependent problems considered in this work.

IV. The computed solution(s) might be unique or non-unique where multiple solutions exist

CHAPTER 4

BIDIRECTIONAL DIJKSTRA ALGORITHM USING PIECE-WISE LINEAR FUNCTION

4.1 INTRODUCTION

Time dependent shortest path problems (TDSPP) have recently attracted considerable interest and utilized extensively to study network problems. TDSPP is the Shortest Path Problem (SPP) in which the cost of edges can vary as a function of time. Since, in a road network, the shortest path from a source node to a destination node during rush over is different from other time of the day, TDSPP is considered as a fundamental optimization problem. TDSPP was first developed by Cooke and Halsey (1966) via a recursive formula to find the shortest travel time from a given source node at a certain time to a given destination node. Bidirectional Dijkstra search is a standard technique to speed up computations on static networks. However, since the arrival time at the destination is unknown, the cost of time-dependent links around the target node cannot be evaluated. Thus, bidirectional search cannot be directly applied on time-dependent networks. Nannicini (2009) proposed a solution to the above mentioned problem by using a time-independent lower bounding function in the backward search.

Nannicini et al. (2012) introduced a bidirectional A^*algorithm for solving shortest path problem and their algorithms is shown to be faster than Dijkstra's algorithm while finding only slightly sub-optimal solutions. Another bidirectional A^*algorithm was proposed by Pijls and Post (2009; 2010) to find the shortest path. Geisberger et al. (2008) presented a hierarchical query algorithm using bidirectional shortest path search. Their algorithm is found to be faster than

hierarchical Dijkstra. Abraham and Shukla (2015) used bidirectional strategy and genetic algorithm for computing the shortest path. Nazemi and Omidi (2013) proposed a neural network model to solve SSP.

In this study, an attempt is made to solve a point to point shortest path problem using a bidirectional algorithm while decreasing computational cost. The proposed heuristic bidirectional Dijkstra based algorithm looks for the shortest path from node s to node t in a graph or network $G(V, A)$ in which V represents a set of nodes and A stands for a set of links. The backward search works on the reversed graph in which every original arc $(u, v) \in A$ is replaced with arc (v, u) having the same cost. The Time Delay Factor (TDF) method combined with A piece-wise linear function, used in Chapter 3, is used to make the links' cost time dependent. Reducing the consumed computational time in finding the shortest path on a large network is far from trivial. In order to achieve this goal, a special procedure is used in the bidirectional algorithm to reduce the number of the explored nodes. In a bidirectional algorithm, for the forward search the source node and the destination nodes are denoted with s, and t , respectively. In the backward search, their roles are interchanged. In the forward search, the departure time is known while in the backward search the departure time (i.e., the arrival time for the forward search) is unknown. In this work, to start the backward search, two methods are examined: in the first method, an arbitrary guessed arrival time is used while in the second method an extrapolated guessed arrival time is used to estimate the arrival time. Both small and large scale (real size) networks are used to evaluate the proposed algorithm. The results of this study show the advantage of the proposed algorithm in term of computational cost. The shortest path and arrival time are slightly different from the optimum solution in some cases. The proposed algorithm is explained in Section 4.2. The numerical implementation and conclusion are provided in Sections 4.3 and 4.4, respectively.

4.2 PROPOSED TIME DEPENDENT BIDIRECTION DIJKSTRAL ALGORITHM

Suppose a network of N nodes and M links is described by a graph $G(V, A)$ where V and A are sets containing all the existing nodes and links in the network, respectively. The cost of each link connecting node v to node w is denoted by $c(v, w)$. The goal is to find the shortest path from the source node s to the destination node t.

The piece-wise linear function that was introduced in Section 3.2 is also used to generate time-dependent cost function for the links in this section. The travel time from node v to node w of a link k is not constant in dynamic networks and depends on the departure time at the starting node v. The following formulas are employed for finding the arrival time at ending node of a typical link $(v \rightarrow w)$.

$$AT = DT + CST_{vw} \times TDF(DT) \tag{78}$$

where AT represents arrival time at the ending node w, DT is the departure time at the starting node v, CST_{vw} stands for Constant Static Time of the link, and TDF is the Time Delay Factor which is dependent on departure time, DT, and can be defined by Equation (79).

$$TDF(DT) = 1 + y(DT) \tag{79}$$

where $y(DT)$ is the time function for the link. The piece-wise linear time function is depicted in Figure 21. Usually in a dynamic network, travel time is increased during rush hours (i.e., during $6am - 8am$ and during $4{:}00\text{pm} - 6{:}00\text{pm}$). In Figure 21, the coordinates $(DT,\ y(DT))$ of such points O, A, B, C, D, E, F, G, H, and I are defined as the input parameter provided by the end-user. Thus, this piece-wise linear time function can be adjusted to take into account the variations of local traffic congestion time.

The proposed algorithm is based on the Dijkstra's algorithm and similarly starts by exploring the source node and finding distance array (d) and predecessor array ($pred$) at each iteration for outgoing link(s) and then, updates them. The distance array, d, contains the distance of each node from the source node(s). The distance array d is found using the following formula:

$$d\,(v) = d\,(w) + CST_{vw} \times TDF \tag{80}$$

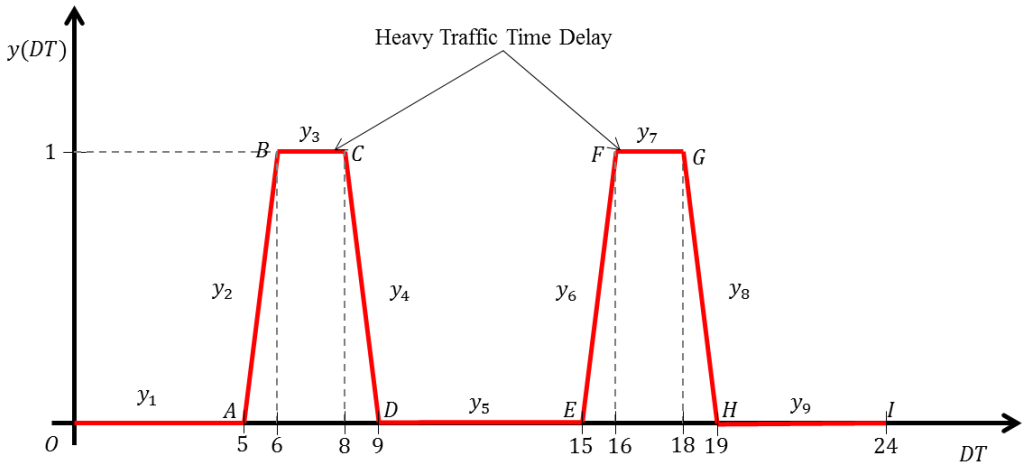

Figure 21. Piece-wise linear time function for a typical link *k*.

where v is the starting node and w is the ending node of the corresponding link, k.The static cost of travel from node v to node w for link k is denoted by CST_{vw}. Also, the corresponding time delay factor, TDF, is calculated by Equation (79).

The static bidirectional algorithm starts its forward and backward search simultaneously. When both searches collide at a node, the algorithm stops. For time dependent networks, this method cannot be used because the arrival time is unknown to start the backward search. In a case

where the arrival time for backward search is guessed, the arrival time at the collision node of both searches cannot be the same.

To overcome the above mentioned difficulties a new bidirectional algorithm is proposed in this work. The proposed algorithm starts the forward and backward search until the collision node is found. Then, the backward search stops and forward search continues to explore nodes which was previously explored through backward search process. The backward search is only implemented to limit the number of nodes required to be explored by forward search. This idea was first presented by Nannicini et al. (2012). They started the backward search using lower bounds on arc costs to analyze a dynamic network using a bidirectional A* algorithm. In a different manner, in this study, a guessed arrival time is used to start the backward search. Note that the departure time for the backward search is found using Equation (81).

$$DT = AT - CST_{ij} \times \big(1 + y_r(DT)\big), r = 1, \dots, 9 \quad (81)$$

where the value of each departure time, DT, can be calculated. Note that $y_r(DT)$, $r = 1, \dots, 9$, is only valid for a certain time range as indicated in Figure 21. For cases where multiple solutions for DT exists, the departure time which produces the smallest value of time function ($y(DT) = y_{min}$) is selected. This selection also corresponds to the smallest value of time delay factor($TDF = TDF_{min}$). Then, the backward search is performed similar to Section 3.2.

The step by step procedure for the proposed algorithm is summarized below:

Phase 1. In this phase, the algorithm starts the forward and backward search simultaneously until the collision node is found. Note that the forward search uses the network with the reversed link's direction. The phase 1 is performed in the following steps:

Step1. Initialize:

- the distance array, d, with an element for each node in the graph. Set the element value corresponding to the source node to zero and all the other elements to a large value (i.e., inf): *Set* $d(s) = 0$ *and for all* $v \in V - \{s\}$, $set\ d(v) = \infty$
- predecessor array, $pred$, with a length equal to the number of nodes in the graph. Then assign zero or null to all the elements of $pred$.
- an array for the exploring nodes, denoted as S. Increase the size of S at each iteration by adding a new node. Note that this array starts either with the source node or the destination node for forward and backward searches, respectively.

Step2.

- For a typical link, $v \rightarrow w$, explore all the outgoing nodes of the last coefficient of array S:

if $d(w) > d(v) + CST_{vw} \times TDF$ *then,*
$d(w) = d(v) + CST_{vw} \times TDF$, *and*
$pred(w) = v$,
else if $d(w) \not> d(v) + CST_{vw} \times TDF$ *then,*
apply no update to that link (i.e., for d *and* $pred$ *arrays).*
end if

- Find the next node to search which is the node with the minimum value of distance array, d.
- If the next exploring node for backward search and forward search is the same go to phase 2, else go to step 2.

Phase 2. Continue the forward search to explore nodes which are settled in the array S found by backward search, $S_{backward}$, in phase 1.

- Explore all the outgoing nodes of $S_{backward}$ array from the collision node to the destination node and update the information:

If $d(w) > d(v) + CST_{vw} \times TDF$ *then,*
$d(w) = d(v) + CST_{vw} \times TDF$ *and*

$pred\ (w) = v,$
else if $d\ (w) \not> \ d\ (v) + \ CST_{vw} \ \times TDF$ *then,*
apply no update to the link (i.e., for d *and* $pred$ *arrays).*
end if

- If the next exploring node is the destination node then stop.
- Specify the Shortest Path cost= $d(t)$, then, find the shortest path by back-tracking the destination node t using $pred$ array.

In a dynamic network that utilize a piece-wise function, an estimate of the arrival time can be obtained using the following extrapolation (Talbot, private communication).

$$t_E = \left[|\Delta N\#|^{\left(\frac{\#N}{\#L}\right)} \times Geo\ Mean\ S\&D\ L.C.\ \times (\bar{y} + 1)\right] + DT \tag{82}$$

where t_E is the extrapolated guessed arrival time, $\Delta N\#$ is the difference in node number for the source and destination nodes, $\#\ N$ is the number of nodes in network, $\#L$ is the number of links in network, $Geo\ Mean\ S\&D\ L.C.$ is the geometric mean of source and destination link cost, $\bar{y}$ is the arithmetic mean of y, and DT is departure time.

Equation (82) is the product of three main components: average number of links between two given points in a given network, average link cost at source and destination nodes, and average piecewise function time penalty plus the departure time.

The first component of the equation uses a power function to estimate the number of links between any two points in a given network. The second component of the equation uses the geometric mean of the source node and destination node link costs to estimate the average link cost throughout the entire network. A geometric mean is used to obtain the "average" since the central tendency it delivers is less sensitive to wide variability in the data. An arithmetic mean is

used to determine the time-based penalty of the network since an input value of zero gives a geometric mean of zero. More details on this formula is presented in Appendix B.

4.3 NUMERICAL IMPLEMENTATION

To facilitate the elaboration of the proposed bidirectional algorithm, a small network is solved first. Then, the proposed algorithm is used to solve multiple real-road network problems.

Problem 4.1 Use the proposed algorithm to find the shortest path from the source node 2 to the destination node 8 when departure time is 6 hours (6 am) for the network depicted in Figure 22. Assume that the guessed arrival time is 13 hours (1 pm). The network with the reversed link's cost is shown in Figure 23.

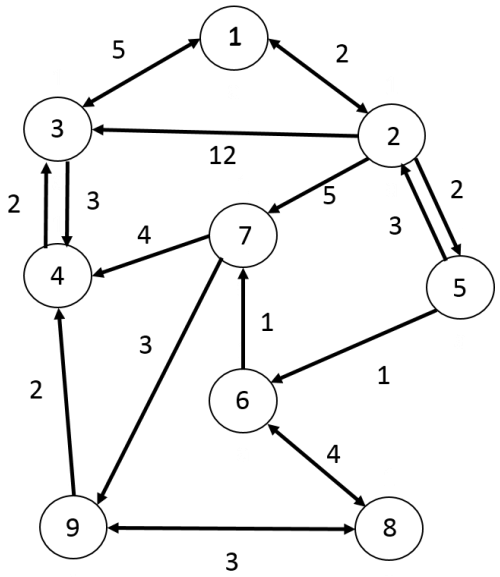

Figure 22. A network topology with 9 nodes and 15 links.

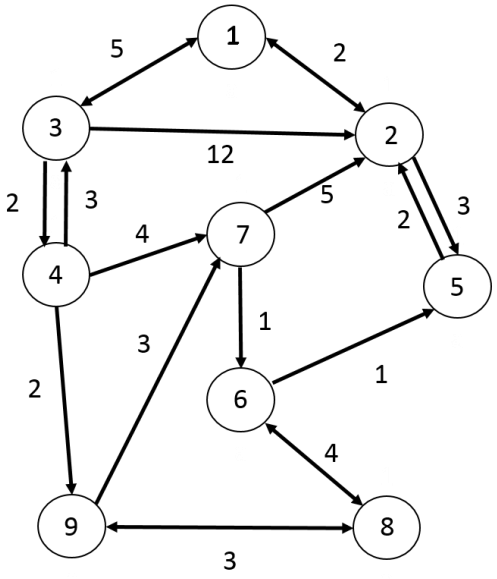

Figure 23. The reversed network topology with 9 nodes and 15 links.

Phase1:
Forward search:

Step1. Initialize:

$Source\ node = 2$
$Destination\ node = 8$
$Departure\ time = 6$
$d = \{Inf \quad 0 \quad Inf \quad Inf \quad Inf \quad Inf \quad Inf \quad Inf \quad Inf\}$
$pred = \{0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0\}$
$S_{Forward} = 2$

Step 2. Explore all the outgoing nodes of the last coefficient of array $S_{Forward}$:

First iteration: Explore all the outgoing links for node 2, when departure time is 6 hours and $TDF = 1 + y = 2$. This information is shown in Table 14.

The next node to explore is node 1 since the value of distance array for both nodes 1 and 5 are the same (i.e., $d(1) = 4$ and $d(4) = 4$), the smallest node is selected to be explored next. Update the information: $next\ node = 1$, $S_{Forward} = \{2 \quad 1\}$

Table 14. Information for exploring all the outgoing links of node 2.

Link 2→ 1
$Link\ cost = \quad 2$
$d = \{4 \quad 0 \quad Inf \quad Inf \quad Inf \quad Inf \quad Inf \quad Inf \quad Inf\}$
$pred = \{\ 2 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0\}$
$Arrival\ time = 10$
Link 2→ 3
$Link\ cost = \quad 12$
$d = \{\ 4 \quad 0 \quad 24 \quad Inf \quad Inf \quad Inf \quad Inf \quad Inf \quad Inf\}$
$pred = \{\ 2 \quad 0 \quad 2 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0\}$
$Arrival\ time = 30$
Link 2→ 5
$Link\ cost = \quad 2$
$d = \{4 \quad 0 \quad 24 \quad Inf \quad 4 \quad Inf \quad Inf \quad Inf \quad Inf\}$
$pred = \{\ 2 \quad 0 \quad 2 \quad 0 \quad 2 \quad 0 \quad 0 \quad 0 \quad 0\}$
$Arrival\ time = 10$
Link 2→ 7
$Link\ cost = \quad 5$
$d = \{4 \quad 0 \quad 24 \quad Inf \quad 4 \quad Inf \quad 10 \quad Inf \quad Inf\}$
$pred = \{\ 2 \quad 0 \quad 2 \quad 0 \quad 2 \quad 0 \quad 2 \quad 0 \quad 0\}$
$Arrival\ time = 16$

Second iteration: Explore all the outgoing links for node 1, when departure time is 10 hours and $TDF = 1 + y = 1$. This information is shown in Table 15.

Table 15. Information for exploring all the outgoing links of node 1.

Link 1→ 2
$Link\ cost = \quad 2$
No update for d and $pred$ arrays.
Link 1→ 3
$Link\ cost = \quad 5$
$d = \{4 \quad 0 \quad 9 \quad Inf \quad 4 \quad Inf \quad 10 \quad Inf \quad Inf\}$
$pred = \{2 \quad 0 \quad 1 \quad 0 \quad 2 \quad 0 \quad 2 \quad 0 \quad 0\}$
$Arrival\ time = 15$

The next node to explore is node 5, update the data to start next iteration:

$S_{Forward} = \{\ 2 \quad 1 \quad 5\}$.

Third iteration: Explore all the outgoing links for node 5, when departure time is 10 hours and $TDF = 1 + y = \ 1$. This information is shown in Table 16.

Table 16. Information for exploring all the outgoing links of node 5.

Link 5→ 2
$Link\ cost = \ \ 3$
No update for d and $pred$ arrays.
Link 5→ 6
$Link\ cost \ = \ \ 1$
$d = \{ 4 \quad 0 \quad 9 \ \ Inf \quad 4 \quad 5 \quad 10 \ \ Inf \quad Inf\}$
$Arrival\ time = 11$
$pred = \{ 2 \quad 0 \quad 1 \quad 0 \quad 2 \quad 5 \quad 2 \quad 0 \quad 0\}$

The next node to explore is node 6, update the data to start next iteration:

$S_{Forward} = \{ \ 2 \quad 1 \quad 5 \quad 6\}$

Backward search:

Step1. Initialize:

$Source\ node = \ 8$
$Destination\ node = \ 2$
$Arrival\ time = 13$
$d = \{Inf \quad Inf \quad Inf \quad Inf \quad Inf \quad Inf \quad Inf \quad 0 \quad Inf\}$
$pred = \{ 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0\}$
$S_{Backward} = \ 8$

Step 2. Explore all the outgoing nodes of the last coefficient of array $S_{Backward}$:

First iteration: Explore all the outgoing links for node 8, when arrival time is 13 hours. Find departure time using Equation (81) and select the proper departure time for each outgoing link. The information obtained in the first iteration is shown in Table 17.

Table 17. Information for exploring all the outgoing links of node 8 (backward search).

Link 8→ 6
$Departure\ time = \{\ Inf \quad 5.8 \quad Inf \quad 9.0 \quad 9.0 \quad Inf \quad Inf \quad Inf \quad Inf\}$
$y = \{\ Inf \quad 0.8 \quad Inf \quad 0 \quad 0 \quad Inf \quad Inf \quad Inf \quad Inf\}$
$y_{min} = y(4) = 0 \Rightarrow\ Departure\ time = 9$
$TDF\ =\ \ 1$
$d = \{\ Inf \quad Inf \quad Inf \quad Inf \quad Inf \quad 4 \quad Inf \quad 0 \quad Inf\}$
$pred = \{\ 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 8 \quad 0 \quad 0 \quad 0\}$
Link 8→ 9
$Departure\ time = \{Inf \quad Inf \quad 7.0 \quad 8.5 \quad 10.0 \quad Inf \quad Inf \quad Inf \quad Inf\}$
$y = \{\ Inf \quad Inf \quad 1.0 \quad 0.5 \quad 0 \quad Inf \quad Inf \quad Inf \quad Inf\}$
$y_{min} = y(5) = 1 \Rightarrow\ Departure\ time = 10$
$TDF\ =\ \ 1$
$d = \{\ Inf \quad Inf \quad Inf \quad Inf \quad Inf \quad 4 \quad Inf \quad 0 \quad 3\}$
$pred = \{0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 8 \quad 0 \quad 0 \quad 8\}$

The next node to explore is node 9, update the data to start the next iteration:

$Arrival\ time\ \ =\ \ 10$
$S_{Backward} = \{\ 8 \quad 9\}$

Second iteration: Explore all the outgoing links for node 9, when arrival time is 10 hours. This iteration is summarized in Table 18.

The next node to explore is node 6, update the data: $S_{Backward} = \{\ 8 \quad 9 \quad 6\}$.

The forward search and backward search met at node 6, so backward search stops here. Forward search continues as follows.

Phase 2:

The forward search continues to explore nodes which are settled in the array S found by backward search, $S_{backward}$.
$S_{Forward} = \{2 \quad 1 \quad 5 \quad 6\}$
$S_{Backward} = \{\ 8 \quad 9 \quad 6\}$

First iteration: Explore all the outgoing links for node 6, when departure time is 11 hours and $TDF = 1 + y =\ \ 1$. This information is shown in Table 19.

Table 18. Information for exploring all the outgoing links of node 9 (backward search).

Link 9→ 7
$Departure\ time = \{Inf \quad 5.5 \quad Inf \quad Inf \quad Inf \quad Inf \quad Inf \quad Inf \quad Inf\}$
$y = \{\ Inf \quad 0.5 \quad Inf \quad Inf \quad Inf \quad Inf \quad Inf \quad Inf \quad Inf\}$
$y_{min} = y(2) = 0.5 \Rightarrow\ Departure\ time = 5.5$
$TDF\ =\ \ 1.5$
$d = \{\ Inf \quad Inf \quad Inf \quad Inf \quad Inf \quad 4.0 \quad 7.5 \quad 0 \quad 3.0\}$
$pred = \{0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 8 \quad 9 \quad 0 \quad 8\}$
Link 9→ 8
$Departure\ time = \{Inf \quad 5.5 \quad Inf \quad Inf \quad Inf \quad Inf \quad Inf \quad Inf \quad Inf\}$
$y\_check\ =\ \ \{\ Inf \quad 0.5 \quad Inf \quad Inf \quad Inf \quad Inf \quad Inf \quad Inf \quad Inf\}$
$y_{min} = y(2) = 0.5 \Rightarrow\ Departure\ time = 5.5$
$TDF = 1.5$
No update for d and $pred$ arrays.

Table 19. Information for exploring all the outgoing links of node 6.

Link 6→ 7
$d = \{4 \quad 0 \quad 9 \quad Inf \quad 4 \quad 5 \quad 6 \quad Inf \quad Inf\}$
$Arrival\ time\ \ = 12$
$pred = \{\ 2 \quad 0 \quad 1 \quad 0 \quad 2 \quad 5 \quad 6 \quad 0 \quad 0\}$
Link 6→ 8
$d = \{4 \quad 0 \quad 9 \quad Inf \quad 4 \quad 5 \quad 6 \quad 9 \quad Inf\}$
$Arrival\ time\ \ = 15$
$pred = \{2 \quad 0 \quad 1 \quad 0 \quad 2 \quad 5 \quad 6 \quad 6 \quad 0\}$

The next node to explore is node 9, update the data to start next iteration:

$S_{Forward} = \{\ 2 \quad 1 \quad 5 \quad 6 \quad 9\}$

Second iteration: Explore all the outgoing links for node 9, when departure time is 11 hours and $TDF = 1 + y =\ \ 1$. There is no update for array d and $pred$ of both outgoing links from node 9 (i.e., link 9→ 4 and link 9→ 8). Therefore, the next node to explore is node 8 which is the destination node.

$S_{Forward} = \{\ 2 \quad 1 \quad 5 \quad 6 \quad 9 \quad 8\}$

Third iteration: The search stops here because the next node to explore is the destination node. The shortest path cost is 9 ($d(destination\ ) = d(8) = 9$) , the shortest path is found by back-

tracking the destination node, $t = 8$, using $pred$ array: $path = \{2 \rightarrow 5 \rightarrow 6 \rightarrow 8\}$, and the arrival time is 15 hours (3 pm).

Problem 4.2 Use the proposed algorithm to find the shortest path from the source node 5 to the destination node 6100 when departure time is 23 hours (11 pm) for Austin road network. This network has 7388 nodes and 18961 links. Since the arrival time is unknown, consider the following 6 values for the arrival time to start the backward search. Compare the results of all the 6 cases with the forward Dijkstra algorithm.

1) Guessed arrival time is 30
2) Guessed arrival time is 40
3) Guessed arrival time is 50
4) Guessed arrival time is 60
5) Guessed arrival time is 70
6) Guessed arrival time is 31.4261 hours when the extrapolated formula, Equation (81), used.

First, the problem was solved using time dependent Dijkstra algorithm and the arrival time at node 6100 was 53.041 hours. Then, problem was solved for all the 6 cases using the proposed algorithm obtaining identical results to those of Dijkstra algorithm.

The proposed heuristic bidirectional algorithm explored fewer nodes in comparison with the Dijkstra algorithm. In other words, the proposed algorithm reduces the computational cost by reducing the number of explored nodes without compromising the accuracy of the results. The Dijkstra algorithm explored 6036 nodes while the proposed algorithm explored less nodes for each case of guessed arrival time. The results are shown in Table 20.

Table 20. Number of explored nodes for all the cases in Problem 4.2.

Dijkstra Algorithm	**The Proposed Algorithm**					
	$AT = 30$	$AT = 40$	$AT = 50$	$AT = 60$	$AT = 70$	$A = 31.4261$
6036	2605	2689	2711	2597	2703	2667

Based on Table 20, the guessed arrival time has direct effect on the number of explored nodes. A proper value for guessed arrival time can help to reduce computational cost of finding the shortest path. In this example, the minimum number of nodes were explored when the arrival time was 60 (hours).

In order to further evaluate the proposed algorithm, 10 more examples were selected from four road networks: Winnipeg, Barcelona, Philadelphia, and Austin. The properties of these examples are given in Table 21. The destination and source nodes of all the examples were selected arbitrary.

The above mentioned examples were solved both by forward Dijkstra and the proposed bidirectional algorithms. The piece-wise linear function, explained in Section 4.2, was used to make both algorithms time dependent. The requirement to start the backward search within the bidirectional algorithm is to guess the arrival time. So, two distinct values for the guessed arrival time were considered in which one of them was selected arbitrary by user and the other one was calculated using the extrapolated guessed arrival time formula given by Equation (82). The results are shown in Table 22 and Table 23.

Table 21. Properties of the large scale examples.

Example	**Network**	**Number of Links**	**Number of Nodes**	**Departure Time** (hours)	**Source Node**	**Destination Node**	**Arbitrary Guessed Arrival Time** (hours)	**Extrapolated Guessed Arrival Time** (hours)
1	Winnipeg	2836	1052	6	5	100	10	15.1562
2	Barcelona	2522	1020	6	5	400	18	12.2964
3	Austin	18961	7388	1	56	1800	22	11.3923
4	Austin	18961	7388	23	5	6100	40	31.4261
5	Philadelphia	40003	13389	1	6	560	12	3.2907
6	Philadelphia	40003	13389	1	48	1415	59	1.6266
7	Philadelphia	40003	13389	6	100	1429	54	6.9814
8	Philadelphia	40003	13389	1	253	1415	59	2.4835
9	Austin	18961	7388	6	1	7388	24	21.1258
10	Barcelona	2522	1020	7	50	1003	22	19.4493

As seen in Table 22, in all examples but example 6, identical values were found for shortest path and arrival time using Dijkstra and the proposed algorithm with arbitrary guessed arrival time. When the extrapolated guessed arrival time was used with the proposed algorithm, the calculated shortest path and the arrival time values were identical to those of forward Dijkstra algorithm for all the examples but examples 6 and 8.

The numerical results presented in Table 22 also show that the proposed algorithm can decrease the number of explored nodes by half when compared to the forward Dijkstra algorithm.

Table 22. Results for forward and bidirectional algorithms for 10 examples of Table 21.

Example	Network	Departure Time (hours)	Arrival Time (hours)			Number of Explored Nodes		
			Forward Dijkstra	Proposed Algorithm +Arbitrary Guessed Arrival Time	Proposed Algorithm + Extrapolated Guessed Arrival Time	Forward Dijkstra	Proposed Algorithm +Arbitrary Guessed Arrival Time	Proposed Algorithm + Extrapolated Guessed Arrival Time
1	Winnipeg	6	16.4940	16.4940	16.4940	563	255	255
2	Barcelona	6	10.5878	10.5878	10.5878	142	95	95
3	Austin	1	22.8558	22.8558	22.8558	4129	2082	2100
4	Austin	23	53.0410	53.0410	53.410	6036	2597	2667
5	Philadelphia	1	13.4817	13.4817	13.4817	5118	2538	2538
6	Philadelphia	1	63.3523	66.6138	63.6382	13386	5658	5800
7	Philadelphia	6	57.1655	57.1655	57.1655	13086	6728	6680
8	Philadelphia	1	67.8548	67.8548	68.7716	13387	6425	6599
9	Austin	6	22.7972	22.7972	22.7972	736	277	277
10	Barcelona	7	14.0048	14.0048	14.0048	566	214	214

The performance of the proposed method in solving the above examples with different values of guessed arrival time is summarized below:

- The number of exploded nodes in five examples (i.e., 1,2,5,9, and 10) was not affected by guessed arrival time.
- In four examples (i.e., 3, 4, 6, and 8), the algorithm explored less nodes with arbitrary arrival time when compared with the extrapolated guessed arrival time.
- In one example (i.e., 7), the algorithm explored less nodes using extrapolated guessed arrival time when compared with the arbitrary guessed arrival time.

Table 23. Comparison of consuming time for forward Dijkstra and the proposed algorithms for 10 examples of Table 21.

Example	Network	Departure Time (hours)	Arbitrary Guessed Arrival Time (hours)	Extrapolated Guessed Arrival Time (hours)	Consuming Time (seconds)		
					Forward Dijkstra	Proposed Algorithm + Arbitrary Guessed Arrival Time	Proposed Algorithm + Extrapolated Guessed Arrival Time
1	Winnipeg	6	10	15.1562	0.0088	0.0053	0.0053
2	Barcelona	6	18	12.2964	0.0048	0.0022	0.0020
3	Austin	1	22	11.3923	0.0879	0.0448	0.0451
4	Austin	23	40	31.4261	0.1169	0.0598	0.0600
5	Philadelphia	1	12	3.2907	0.1570	0.0691	0.0689
6	Philadelphia	1	59	1.6266	0.3640	0.1547	0.1609
7	Philadelphia	6	54	6.9814	0.3629	0.1832	0.1819
8	Philadelphia	1	59	2.4835	0.3656	0.1715	0.1732
9	Austin	6	24	21.1258	0.0312	0.0072	0.0073
10	Barcelona	7	22	19.4493	0.0072	0.0037	0.0037

Note that the number of explored nodes for Dijkstra algorithms is the same as the number of iterations. Since, the proposed algorithm needs less iterations to find the shortest path it also can reduces the computation time. Based on the numerical results presented in Table 23, the computational speed of the proposed algorithm is at least 60% and at most 430% higher than that of forward Dijkstra algorithm.

4.4 CONCLUSION

In the proposed heuristic bidirectional algorithm, the forward and backward searches start simultaneously. When both searches meet at a node, the backward search stops. Then, the forward

search continue to explore nodes which already settled by backward search. To generate a time dependent links cost, a Time Delay Factor method was combined with a piece-wise linear function.

The backward search is only used to restrict the search space of forward search to the nodes that have been previously explored by backward search. The backward search explores all the nodes on the shortest path which has not been explored by the forward search.

Based on the numerical results, the proposed bidirectional algorithm is able to find the shortest path while decreasing the computational cost. The speedup is significant even though in some cases the obtained solutions were slightly sub-optimal. This increased speed is significant when the objective is to simultaneously find the shortest paths for multiple routes on a road network.

Since a good guessed arrival time directly affects the number of explored nodes throughout the algorithm, the extrapolated guessed arrival time is the preferred method over the arbitrary guessed arrival time. The extrapolated guessed arrival time is calculated by proposed algorithm and not no longer considered input data.

REFERENCES

Abbass, H. A. (2002, 12-17 May 2002). *The self-adaptive Pareto differential evolution algorithm.* Paper presented at the Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on.

Abraham, S. C., & Shukla, G. D. (2015). Shortest path computation in large graphs using bidirectional strategy and genetic algorithms. *International Journal of Computer Applications, 109*(13) doi:http://dx.doi.org.proxy.lib.odu.edu/10.5120/19250-0932

Allen, S. E. (2013) *Parallel implementations of the Frank-Wolfe algorithms for the traffic assignment problem, M.Sc. Thesis, MSVE Department, Old Dominion University, Norfolk, USA.* http://dx.doi.org/10.1109/ivs.2005.1505180

Azad, M. A. K., & Fernandes, M. G. P. (2013). Modified constrained differential evolution for solving nonlinear global optimization problems. In K. Madani, A. Dourado, A. Rosa, & J. Filipe (Eds.), *Computational Intelligence: Revised and Selected Papers of the International Joint Conference, IJCCI 2011, Paris, France, October 24-26, 2011* (pp. 85-100). Berlin, Heidelberg: Springer Berlin Heidelberg.

Bakhtyar, G., Nguyen, V., Cetin, M., & Nguyen, D. (2016) Backward Dijkstra algorithms for finding the departure time based on the specified arrival time for real-life time-dependent networks. *Journal of Applied Mathematics and Physics*, 4, 1-7. doi: 10.4236/jamp.2016.41001.

Balling, R. J., Briggs, R. R., & Gillman, K. (2006). Multiple optimum size/shape/topology designs for skeletal structures using a genetic algorithm. *Journal of Structural Engineering, 132*(7), 1158-1165.

Brest, J., Greiner, S., Boskovic, B., Mernik, M., & Zumer, V. (2006). Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *IEEE Transactions on Evolutionary Computation, 10*(6), 646-657. doi:10.1109/TEVC.2006.872133

Camp, C., & Bichon, B. (2004). Design of space trusses using ant colony optimization. *Journal of Structural Engineering, 130*(5), 741-751.

Chabini, I. and Ganugapati, S. (2002) Parallel algorithms for dynamic shortest path problems, *International Transactions in Operational Research*, 9, 279-302. http://dx.doi.org/10.1111/1475-3995.00356

Chen, T. Y., & Chen, H. C. (2008).Mixed-discrete structural optimization using a rank-niche evolution strategy. *Engineering Optimization, 41*, 39-58.

Cooke, K. L., & Halsey, E. (1966). The shortest route through a network with time-dependent internodal transit times. *Journal of Mathematical Analysis and Applications, 14*(3), 493-498. doi:http://dx.doi.org/10.1016/0022-247X(66)90009-6

Cui, L., Li, G., Lin, Q., Chen, J., & Lu, N. (2016). Adaptive differential evolution algorithm with novel mutation strategies in multiple sub-populations. *Computers & Operations Research, 67*, 155-173. doi:http://dx.doi.org/10.1016/j.cor.2015.09.006

Daganzo, C. F. (2002) Reversibility of the time-dependent shortest path problem. *Transportation Research Part B: Methodological*, 36(7), 665-668. http://dx.doi.org/10.1016/s0191-2615(01)00012-1

Das, S., Konar, A., & Chakraborty, U. K. (2005). *Two improved differential evolution schemes for faster global search*. Paper presented at the Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation, Washington DC, USA.

Das, S., & Suganthan, P. (2011). Differential evolution: A survey of the state-of-the-art. *IEEE Transactions on Evolutionary Computation, 15*(1), 4-31.

Das, S., Mullick, S. S., & Suganthan, P. N. (2016). Recent advances in differential evolution – An updated survey. *Swarm and Evolutionary Computation, 27*, 1-30. doi:http://dx.doi.org/10.1016/j.swevo.2016.01.004

Deb, K. (2000). An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering, 186*(2), 311-338. doi:http://dx.doi.org/10.1016/S0045-7825(99)00389-8

Ding, B., Yu, J.X. ,& Qin, L. (2008) Finding time-dependent shortest paths over large graphs. *EDBT'2008 Proceedings of the 11th International Conference on Extending Database Technology: Advances in Database Technology*, 205-216. http://dx.doi.org/10.1145/1353343.1353371

Eiben, A. E., Hinterding, R., & Michalewicz, Z. (1999). Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation, 3*(2), 124-141. doi:10.1109/4235.771166

Fan, Q., & Yan, X. (2015). Self-adaptive differential evolution algorithm with discrete mutation control parameters. *Expert Systems with Applications, 42*(3), 1551-1572. doi:http://dx.doi.org/10.1016/j.eswa.2014.09.046

Feoktistov, V. (2006). *Differential evolution: In search of solutions (Springer optimization and its applications)*: Springer-Verlag New York, Inc.

Gamperle, R., Müller, S. D., & Koumoutsakos, P. (2002). A parameter study for differential evolution, *Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation*, A. Grmela and N. E. Mastorakis, Eds. Interlaken, Switzerland: WSEAS Press, 2002, pp. 293–298.

Gandomi, A.H., Yang, X., & Alavi, A.H. (2011). Mixed variable structural optimization using Firefly Algorithm. *Computers and Structures, 89*(23), 2325-2336.

Geisberger, R., Sanders, P., Schultes, D., & Delling, D. (2008). Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In C. C. McGeoch (Ed.), *Experimental Algorithms: 7th International Workshop, WEA 2008 Provincetown, MA, USA, May 30-June 1, 2008 Proceedings* (pp. 319-333). Berlin, Heidelberg: Springer Berlin Heidelberg.

Gholizadeh, S. (2013). Layout optimization of truss structures by hybridizing cellular automata and particle swarm optimization. *Computers & Structures, 125*, 86-99. doi:http://dx.doi.org/10.1016/j.compstruc.2013.04.024

Goldberg, D.E., Personal communication, September 1992.

Gonçalves, M. S., Lopez, R. H., & Miguel, L. F. F. (2015). Search group algorithm: A new metaheuristic method for the optimization of truss structures. *Computers & Structures, 153*, 165-184. doi:http://dx.doi.org/10.1016/j.compstruc.2015.03.003

Hasançebi, O., & Erbatur, F. (2002). Layout optimisation of trusses using simulated annealing. *Advances in Engineering Software, 33*(7), 681-696. doi:http://dx.doi.org/10.1016/S0965-9978(02)00049-2

Hernández, S., Leguizamón, G., & Mezura-Montes, E. (2013, 20-23 June 2013). *A hybrid version of differential evolution with two differential mutation operators applied by stages.* Paper presented at the 2013 IEEE Congress on Evolutionary Computation.

Ho-Huu, V., Nguyen-Thoi, T., Nguyen-Thoi, M. H., & Le-Anh, L. (2015). An improved constrained differential evolution using discrete variables (D-ICDE) for layout optimization of truss structures. *Expert Systems with Applications, 42*(20), 7057-7069. doi:http://dx.doi.org/10.1016/j.eswa.2015.04.072

Islam, S. M., Das, S., Ghosh, S., Roy, S., & Suganthan, P. N. (2012). An adaptive differential evolution algorithm with novel mutation and crossover strategies for global numerical optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 42*(2), 482-500. doi:10.1109/TSMCB.2011.2167966

Jia, G., Wang, Y., Cai, Z., & Jin, Y. (2013). An improved (μ+λ)-constrained differential evolution for constrained optimization. *Information Sciences, 222*, 302-322. doi:http://dx.doi.org/10.1016/j.ins.2012.01.017

Kaveh, A., & Kalatjari, V. (2004). Size/geometry optimization of trusses by the force method and genetic algorithm. *ZAMM - Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik, 84*(5), 347-357. doi:10.1002/zamm.200310106

Kaveh, & Talatahari. (2009). Particle swarm optimizer, ant colony strategy and harmony search scheme hybridized for optimization of truss structures. *Computers and Structures, 87*(5), 267-283.

Krempser, E., Bernardino, H. S., Barbosa, H. J. C., & Lemonge, A.C.C. (2012). Differential evolution assisted by surrogate models for structural optimization problems, *in B.H.V. Topping, (Editor), Proceedings of the Eighth International Conference on Engineering Computational Technology, Civil-Comp Press*, Stirlingshire, UK, Paper 49, 2012. doi:10.4203/ccp.100.49

Lampinen, J., & Zelinka, I. (1999). Mechanical engineering design optimization by differential evolution. In C. David, D. Marco, G. Fred, D. Dipankar, M. Pablo, P. Riccardo, & V. P. Kenneth (Eds.), *New ideas in optimization* (pp. 127-146): McGraw-Hill Ltd., UK.

Lee, K. S., Geem, Z. W., Lee, S.-h., & Bae, K.-w. (2005). The harmony search heuristic algorithm for discrete structural optimization. *Engineering Optimization, 37*(7), 663-684. doi:10.1080/03052150500211895

Liu, J., & Lampinen, J. (2005). A fuzzy adaptive differential evolution algorithm. *Soft Computing, 9*(6), 448-462. doi:10.1007/s00500-004-0363-x

Liao, T. W. (2010). Two hybrid differential evolution algorithms for engineering design optimization. *Applied Soft Computing, 10*(4), 1188-1199. doi:http://dx.doi.org/10.1016/j.asoc.2010.05.007

Malandraki, C., & Daskin, M. S. (1992). Time dependent vehicle routing problems: Formulations, properties and heuristic algorithms. *Transportation Science*, *26*(3), 185.

Mallipeddi, R., & Suganthan, P. N. (2010). Differential evolution algorithm with ensemble of parameters and mutation and crossover strategies. In B. K. Panigrahi, S. Das, P. N. Suganthan, & S. S. Dash (Eds.), *Swarm, Evolutionary, and Memetic Computing: First International Conference on Swarm, Evolutionary, and Memetic Computing, SEMCCO 2010, Chennai, India, December 16-18, 2010. Proceedings* (pp. 71-78). Berlin, Heidelberg: Springer Berlin Heidelberg.

Martini, K. (2011). Harmony search method for multimodal size, shape, and topology optimization of structural frameworks. *Journal of Structural Engineering, 137*(11), 1332-1339. doi:doi:10.1061/(ASCE)ST.1943-541X.0000378

Mezura-Montes, E., Velazquez-Reyes, J., & Coello, C. A. C. (2006). *Modified differential evolution for constrained optimization.* Paper presented at the 2006 IEEE International Conference on Evolutionary Computation.

Miguel, L. F. F., Lopez, R. H., & Miguel, L. F. F. (2013). Multimodal size, shape, and topology optimisation of truss structures using the Firefly algorithm. *Advances in Engineering Software, 56*, 23-37.Doi:http://dx.doi.org/10.1016/j.advengsoft .2012. 11.006

Mohamed, A. W., & Sabry, H. Z. (2012). Constrained optimization based on modified differential evolution algorithm. *Information Sciences, 194*, 171-208. doi:http://dx.doi.org/10.1016/j.ins.2012.01.008

Mohamed, A. W., Sabry, H. Z., & Khorshid, M. (2012). An alternative differential evolution algorithm for global optimization. *Journal of Advanced Research, 3*(2), 149-165. doi:http://dx.doi.org/10.1016/j.jare.2011.06.004

Mohrig, J. R., Alberg, D., Holifmeister, G., Schatz, P. F., & Hammond, C. N. (2014). *Laboratory techniques in organic chemistry*. WH Freeman & Co Ltd.

Nannicini, G., & Liberti, L. (2008). Shortest paths on dynamic graphs. *International Transactions in Operational Research, 15*(5), 551-563. doi:10.1111/j.1475-3995.2008.00649.x

Nannicini, G. (2009) Point-to-point shortest paths on dynamic time-dependent road networks, Ph.D. Dissertation, Ecole Polytechnique, France. http://dx.doi.org/10.1007/s10288-010-0121-0

Nannicini, G., Delling, D., Schultes, D., & Liberti, L. (2012). Bidirectional A* search on time-dependent road networks. *Networks, 59*(2), 240-251. doi:10.1002/net.20438

Nazemi, A., & Omidi, F. (2013). An efficient dynamic model for solving the shortest path problem. *Transportation Research Part C: Emerging Technologies, 26*, 1-19. doi:http://dx.doi.org/10.1016/j.trc.2012.07.005

Omran, M. G. H., Salman, A., & Engelbrecht, A. P. (2005). Self-adaptive differential evolution. In Y. Hao, J. Liu, Y. Wang, Y.-m. Cheung, H. Yin, L. Jiao, J. Ma, & Y.-C. Jiao (Eds.), *Computational Intelligence and Security: International Conference, CIS 2005, Xi'an, China, December 15-19, 2005, Proceedings Part I* (pp. 192-199). Berlin, Heidelberg: Springer Berlin Heidelberg.

Orda, A., and Rom R. (1990) Shortest path and minimum delay algorithms in networks with time-dependent edge length, *Journal of the Association for Computing Machinery*, 37 (3), 607-625. http://dx.doi.org/10.1145/79147.214078

Pham, A. H. (2016). Discrete optimal sizing of truss using adaptive directional differential evolution. *Advances in Computational Design, 1*(3), 275-296. doi:10.12989/acd.2016.1.3.275

Pijls, W., & Post, H. (2009). *Yet another bidirectional algorithm for shortest paths*. St. Louis: Federal Reserve Bank of St Louis. Retrieved from https://search-proquest-com.proxy.lib.odu.edu/docview/1698342477?accountid=12967

Pijls, W., & Post, H. (2010). Note on "A new bidirectional algorithm for shortest paths". *European Journal of Operational Research, 207*(2), 1140-1141. doi:http://dx.doi.org/10.1016/j.ejor.2010.06.003

Price, K. (1999). *New ideas in optimization*: McGraw-Hill Ltd., UK.

Mezura-Montes, E., Velazquez-Reyes, J., & Coello, C. A. C. (2006). *Modified differential evolution for constrained optimization.* Paper presented at the 2006 IEEE International Conference on Evolutionary Computation.

Price, K., Storn, R., & Lampinen, J. (2006). *Differential evolution: A practical approach to global optimization*. Berlin/Heidelberg: Springer-Verlag Berlin and Heidelberg GmbH & KG.

Qin, A. K., Huang, V. L., & Suganthan, P. N. (2009). Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE Transactions on Evolutionary Computation, 13*(2), 398-417. doi:10.1109/TEVC.2008.927706

Rahami, H., Kaveh, A., & Gholipour, Y. (2008). Sizing, geometry and topology optimization of trusses via force method and genetic algorithm. *Engineering Structures, 30*(9), 2360-2369. doi:http://dx.doi.org/10.1016/j.engstruct.2008. 01.012

Rajan, S. D. (1995). Sizing, shape, and topology design optimization of trusses using genetic algorithm. *Journal of Structural Engineering, 121*(10), 1480-1487. doi:doi:10.1061/(ASCE)0733-9445(1995)121:10(1480)

Rajeev, S., & Krishnamoorthy, C. S. (1992). Discrete optimization of structures using genetic algorithms. *Journal of Structural Engineering, 118*(5), 1233-1250. doi:doi:10.1061/(ASCE)0733-9445(1992)118:5(1233)

Ronkkonen, J., Kukkonen, S., & Price, K. V. (2005, 5-5 Sept. 2005). *Real-parameter optimization with differential evolution.* Paper presented at the 2005 IEEE Congress on Evolutionary Computation.

Ruiyi, S., Liangjin, G., & Zijie, F. (2009). Truss topology optimization using genetic algorithm with individual identification technique. *Lecture Notes in Engineering and Computer Science,2177*(1), 1089-1093.

Schmidt, H., & Thierauf, G. (2005). A combined heuristic optimization technique. *Advances in Engineering Software, 36*(1), 11-19. doi:http://dx.doi.org/10.1016/j.advengsoft.2003.12.001

Shojaee, S., Arjomand, M., & Khatibinia, M. (2013). A hybrid algorithm for sizing and layout optimization of truss structures combining discrete PSO and convex approximation. *International Journal of Optimization in Civil Engineering, 3*, 57–83.

Storn, R., & Price, K., "Differential evolution - A simple and efficient adaptive scheme for global optimization over continuous spaces" , Technical Report TR-95-012, ICSI, March 1995, ftp.icsi.berkeley.edu.

Storn, R., & Price, K. (1997). Differential evolution – A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization, 11*(4), 341-359. doi:10.1023/a:1008202821328

Talbot, C., Personal communication, June 2017

Tanabe, R., & Fukunaga, A. (2013, 20-23 June 2013). *Success-history based parameter adaptation for Differential Evolution.* Paper presented at the 2013 IEEE Congress on Evolutionary Computation.

Tang, W., Tong, L., & Gu, Y. (2005). Improved genetic algorithm for design optimization of truss structures with sizing, shape and topology variables. *International Journal for Numerical Methods in Engineering, 62*(13), 1737-1762. doi:10.1002/nme.1244

Tang, L., Dong, Y., & Liu, J. (2015). Differential evolution with an individual-dependent mechanism. *IEEE Transactions on Evolutionary Computation, 19*(4), 560-574. doi:10.1109/TEVC.2014.2360890

Teo, J. (2006). Exploring dynamic self-adaptive populations in differential evolution. *Soft Computing, 10*(8), 673-686. doi:10.1007/s00500-005-0537-1

Wang, D., Zhang, W. H., & Jiang, J. S. (2002). Combined shape and sizing optimization of truss structures. *Computational Mechanics, 29*(4), 307-312. doi:10.1007/s00466-002-0343-x

Wang, Y., Cai, Z., Guo, G., & Zhou, Y. (2007). Multiobjective optimization and hybrid evolutionary algorithm to solve constrained optimization problems. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 37*(3), 560-575. doi:10.1109/TSMCB.2006.886164

Wang, Y., Cai, Z., Zhou, Y., & Zeng, W. (2008). An adaptive tradeoff model for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation, 12*(1), 80-92. doi:10.1109/TEVC.2007.902851

Wang, Z., Tang, H., & Li, P. (2009, 19-20 Dec. 2009). *Optimum design of truss structures based on differential evolution strategy.* Paper presented at the 2009 International Conference on Information Engineering and Computer Science.

Wang, Y., & Cai, Z. (2011). Constrained evolutionary optimization by means of (μ + λ)-differential evolution and improved adaptive trade-off model. *Evolutionary Computation, 19*(2), 249-285. doi:10.1162/EVCO_a_00024

Wang, Y., Cai, Z., & Zhang, Q. (2012). Enhancing the search ability of differential evolution through orthogonal crossover. *Information Sciences, 185*(1), 153-177. doi:http://dx.doi.org/10.1016/j.ins.2011.09.001

Wu, S.-J., & Chow, P.-T. (1995). Integrated discrete and configuration optimization of trusses using genetic algorithms. *Computers & Structures, 55*(4), 695-702. doi:http://dx.doi.org/10.1016/0045-7949(94)00426-4

Wu, C.-Y., & Tseng, K.-Y. (2010). Truss structure optimization using adaptive multi-population differential evolution. *Structural and Multidisciplinary Optimization, 42*(4), 575-590. doi:10.1007/s00158-010-0507-9

Wu, C., & Tseng, K. (2009). Stress-based binary differential evolution for topology optimization of structures. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science, 224*(2), 443-457.

Wu, C., & Tseng, K. (2010). Topology optimization of structures using modified binary differential evolution. *Structural and Multidisciplinary Optimization, 42*(6), 939-953.

Wuming, L., & Pingyang, H. (2007) Study on non-FIFO arc in time-dependent networks. *Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing* http://dx.doi.org/10.1109/snpd.2007.445

Zamuda, A., & Brest, J. (2015). Self-adaptive control parameters ′randomization frequency and propagations in differential evolution. *Swarm and Evolutionary Computation, 25*, 72-99. doi:https://doi.org/10.1016/j.swevo.2015.10.007

Zhang, J., & Sanderson, A. C. (2009). JADE: Adaptive differential evolution with optional external archive. *IEEE Transactions on Evolutionary Computation, 13*(5), 945-958. doi:10.1109/TEVC.2009.2014613

APPENDICES

APPENDIX A: EVALUATION OF DEB'S CONSTRAINT HANDLING METHOD

This work compared the performance of the Differential Evolution optimizer with the performance of the Genetic Algorithm by incorporating Deb's constraint violation method. Each algorithm run 50 times and the computed optimal objective function values was recorder for each of the following test problems. Problems 1 to 4 are selected from Deb's paper (Deb, 2000):

1. Test problem 1:

Minimize

$$f_1(x) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 \tag{1a}$$

subjected to

$$g_1(x) = 4.84 - (x_1 - 0.05)^2 - (x_2 - 2.5)^2 \geq 0 \tag{2a}$$

$$g_2(x) = x_1^2 + (x_2 - 2.5)^2 - 4.84 \geq 0 \tag{3a}$$

$$0 \leq x_1 \leq 6, 0 \leq x_2 \leq 6 \tag{4a}$$

The problem has the optimum solution at (3, 2) with a function value equal to zero. The maximum number of objective function evaluations is considered as 2,000.

2. Test problem 3:

Minimize

$$f_3(\vec{x}) = 5\sum_{i=1}^{4} x_i - 5\sum_{i=1}^{4} {x_i}^2 - \sum_{i=5}^{13} x_i \tag{5a}$$

subjected to

$$g_1(\vec{x}) = 2x_1 + 2x_2 + x_{10} + x_{11} \leq 10 \tag{6a}$$

$$g_2(\vec{x}) = 2x_1 + 2x_3 + x_{10} + x_{12} \leq 10 \tag{7a}$$

$$g_3(\vec{x}) = 2x_2 + 2x_3 + x_{11} + x_{12} \leq 10 \quad (8a)$$

$$g_4(\vec{x}) = -8x_1 + x_{10} \leq 0 \quad (9a)$$

$$g_5(\vec{x}) = -8x_2 + x_{11} \leq 0 \quad (10a)$$

$$g_6(\vec{x}) = -8x_3 + x_{12} \leq 0 \quad (11a)$$

$$g_7(\vec{x}) = -2x_4 - x_5 + x_{10} \leq 0 \quad (12a)$$

$$g_8(\vec{x}) = -2x_6 - x_7 + x_{11} \leq 0 \quad (13a)$$

$$g_9(\vec{x}) = -2x_8 - x_9 + x_{12} \leq 0 \quad (14a)$$

$$0 \leq x_i \leq 1, i = 1, \dots, 9 \quad (15a)$$

$$0 \leq x_i \leq 100, i = 10, 11, 12 \quad (16a)$$

$$0 \leq x_{13} \leq 1 \quad (17a)$$

The optimum solution for this problem is $\vec{x} = (1, 1,1,1,1,1,1,1,1,3,3,3,1)$, $f_3(\vec{x}) = -15$. The maximum number of objective function evaluations is considered as 20,000.

3. Test problem 4:

Minimize

$$f_4(\vec{x}) = x_1 + x_2 + x_3 \quad (18a)$$

subjected to:

$$g_1(\vec{x}) = 1 - 0.0025\,(x_4 + x_6) \geq 0 \quad (19a)$$

$$g_2(\vec{x}) = 1 - 0.0025(x_5 + x_7 - x_4) \geq 0 \quad (20a)$$

$$g_3(\vec{x}) = 1 - 0.01(x_8 - x_5) \geq 0 \quad (21a)$$

$$g_4(\vec{x}) = x_1x_6 - 833.33252x_4 - 100x_1 + 83333.333) \geq 0 \quad (22a)$$

$$g_5(\vec{x}) = x_2x_7 - 1250x_5 - x_2x_4 + 1250x_4 \quad (23a)$$

$$g_6(\vec{x}) = x_3x_8 - x_3x_5 + 2500x_5 - 1250000 \geq 0 \quad (24a)$$

$$100 \leq x_1 \leq 10000 \quad (25a)$$

$$1000 \leq x_2, x_3 \leq 10000 \tag{26a}$$

$$100 \leq x_i \leq 1000, i = 4, \ldots, 8 \tag{27a}$$

The optimum solution is $\vec{x} = (579.3167, 1359.943, 5110.071, 182.0174, 295.5985, 217.9799, 286.4162, 395.5979)$, $f_4(\vec{x}) = 7049.330923$. The maximum number of objective function evaluations is considered as 32,000.

4. Test problem 5:

Minimize

$$f_5(\vec{x}) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11.0)^2 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7 \tag{28a}$$

subjected to:

$$g_1(\vec{x}) = 127 - 2x_1^2 - 3x_2^4 - x_3 - 4x_4^2 - 5x_5) \geq 0 \tag{29a}$$

$$g_2(\vec{x}) = 282 - 7x_1^2 - 3x_2 - 10x_3^2 - x_4 + x_5) \geq 0 \tag{30a}$$

$$g_3(\vec{x}) = 196 - 23x_1 - x_2^2 - 6x_6^2 + 8x_7) \geq 0 \tag{31a}$$

$$g_4(\vec{x}) = -4x_1^2 - x_2^2 + 3x_1x_2 - 2x_3^2 - 5x_6 + 11x_7) \geq 0 \tag{32a}$$

$$-10 \leq x_i \leq 10, i = 1, \ldots, 7 \tag{33a}$$

The optimum solution is $\vec{x} = (2.330499, 1.951372, -0.4775414, 4.365726, -0.6244870, 1.038131, 1.594227)$, $f_5(\vec{x}) = 680.6300573$. The maximum number of objective function evaluations is considered as 7,000).

5. The Ackley function with 5 variables (max objective function evaluations 20,000)

6. The Rastrigin function with 5 variables (max objective function evaluations 20,000)

Initial Information is summarized below:

For DE: $Cr \in [0, 1]$ =0.9, $F = 0.5$

For GA: Polynomial-based mutation parameters: η_m=100 (the distribution index for mutation), η_c=1(the distribution index for SBX)

Population Size=10× number of design variables

Since, (Number of iterations +1)× Population Size= Max Objective Function valuations, the maximum number of generation for all the problems are calculated as follows:

- problem 1: Number of design variables =2, Number of generations =99
- problem 2: Number of design variables=13, Number of generations=153
- problem 3: Number of design variables=8, Number of generations=399
- problem 4: Number of design variables=7, Number of generations=99
- problem 5: Number of design variables=5, Number of generations=399
- problem 6: Number of design variables=5, Number of generations =399

According to Table 24, when DE Method is applied ,48 runs out of 50 runs have found a solution within 50% of the optimal objective function value and this has been achieved with only a maximum of 2,000 function evaluation. However; 40 runs out of 50 have obtained a solution within 50% of the optimal objective function value when GA Method is applied.

For test problem 3 and test problem 4, DE optimizer is able to find optimal solution for all 50 runs within 2% of the optimum solution. GA optimizer found the optimal solution for 46 runs out of 50 runs within 2% of the optimum solution for test problem 3 and 1 run out of 50 runs within 2% of the optimum solution for test problem 4. These results are shown in Table 25 and Table 26.

Table 24. Number of runs (out of 50 runs) converged within ϵ % of the best-known solution using real-coded DE and GA with constraint handling scheme on test problem 1(True optimum solution =13.59085).

Method	ϵ							**Infeasible**	**Optimized $f_1(\vec{x})$**		
	≤ 1%	≤ 2%	≤ 5%	≤ 10%	≤ 20%	≤ 50%	≥ 50%		Best	Median	Worst
DE	47	47	47	47	48	48	2	0	13.5908	13.5908	26.7482
GA	36	36	36	38	39	40	10	0	13.9511	13.5953	268.2987

Table 25. Number of runs (out of 50 runs) converged within ϵ % of the best-known solution using real-coded DE and GA with constraint handling scheme on test problem 3(True optimum solution=-15).

Method	ϵ							**Infeasible**	**Optimized $f_3(\vec{x})$**		
	≤ 1%	≤ 2%	≤ 5%	≤ 10%	≤ 20%	≤ 50%	≥ 50%		Best	Median	Worst
DE	50	50	50	50	50	50	0	0	-14.9888	-14.9094	-14.9569
GA	46	46	47	49	50	50	0	0	-14.9902	-14.9626	-12.0370

Table 26. Number of runs (out of 50 runs) converged within ϵ % of the best-known solution using real-coded DE and GA with constraint handling scheme on test problem 4(True optimum solution=7049.330923).

Method	ϵ							**Infeasible**	**Optimized $f_4(\vec{x})$**		
	≤ 1%	≤ 2%	≤ 5%	≤ 10%	≤ 20%	≤ 50%	≥ 50%		Best	Median	Worst
DE	49	50	50	50	50	50	0	0	7056.7	7065.9	7171.6
GA	0	1	5	31	44	50	0	0	7153.6	7676.4	9858.6

For test problem 5, both optimizers found optimum solution for all 50 runs and all 50 runs have optimum solution within 1% of the true optimum solution. The value of objective function for this problem is 680.6300573. The best solution was found by DE is much closer to the true optimum solution than the best solution was found by GA (Table A4).

The Ackley and the Rastrigin functions results are shown in Table 27 and Table 28. The optimum solution is zero for both two problems. The best, worst, and median computed optimal values

of all 50 runs show that the DE optimizer can find better result than GA in terms of finding a solution closer to the true optimum.

Table 27. Number of runs (out of 50 runs) converged within ϵ % of the best-known solution using real-coded DE and GA with constraint handling scheme on test problem 5 (True optimum solution=680.630573).

Method	ϵ							**Infeasible**	**Optimized $f_5(\vec{x})$**		
	≤ 1%	≤ 2%	≤ 5%	≤ 10%	≤ 20%	≤ 50%	≥ 50%		Best	Median	Worst
DE	50	50	50	50	50	50	0	0	681.0933	681.7487	683.0978
GA	50	50	50	50	50	50	0	0	681.1196	681.9538	683.4061

Table 28. Number of runs (out of 50 runs) converged using real-coded DE and GA with constraint handling scheme on the Ackley function with 5 variables (True optimum solution=0).

Method	**Optimized $f_{Ackley}(\vec{x})$**		
	Best	Median	Worst
DE	8.8818e-16	4.4409e-15	4.4409e-15
GA	6.3671e-4	0.0030	0.0117

According to Table 29, just for test problem 3 the best computed optimal value that has been found by GA is closer to the true optimum solution. DE's distance from the true optimum solution (relative error) is 0.075% and GA's distance is 0.065%. For the other five test problems, DE has found better solutions. The consuming time for DE and GA for one run is compared in Table 30. The differences for our 6 test problems are negligible.

Table 29. Number of runs (out of 50 runs) converged using real-coded DE and GA with constraint handling scheme on the Rastrigin function with 5 variables (True optimum solution=0).

Method	**Optimized $f_{Rastrigin}(\vec{x})$**		
	Best	Median	Worst
DE	8.8818e-16	4.4409e-15	4.4409e-15
GA	8.8818e-15	4.4608e-4	2.6421

Table 30. Comparison of the best solution of DE and GA optimizer.

Test Problem	True Optimum	DE 's Best Solution	GA 's Best Solution	Distance from the True Optimum (%)	
				DE	GA
Problem 1	13.59085	13.5908	13.9511	0	2.651
Problem 3	-15	-14.9888	-14.9902	0.075	0.065
Problem 4	7049.330923	7056.7	7153.6	0.105	1.479
Problem 5	680.6300573	681.0933	681.1196	0.068	0.072
Ackley	0	8.8818E-16	6.37E-4	0	0.064
Rastrigin	0	8.8818E16	8.8818E-14	0	0

Table 31. Comparison of the time consuming for DE and GA optimizer.

	Time Consuming for a run (seconds)	
Test Problem	DE	GA
Problem 1	0.066596	0.090486
Problem 3	0.709190	0.829371
Problem 4	0.916993	0.982227
Problem 5	0.208502	0.243718
Ackley	0.751477	0.760782
Rastrigin	0.696611	0.722260

The performance of DE optimizer for three different values of crossover rate Cr is investigated. These results are displayed in Table 32 to Table 37 for all six test problems. According to the mentioned results, when Cr =0.9 the DE algorithm can find the best solution with smallest distance from the true optimum.

It can be concluded that DE optimizer performance is better than GA optimizer in term of finding a solution closer to the optimum solution. DE algorithm with constraint handling is a practical optimization technique which has the ability to handle non-differentiable, nonlinear and multimodal cost functions. It can be parallelized in order to save consuming time and memory requirement. It is easy to use with few control variables. DE is a very simple and straightforward strategy and converges so well would be of great interest.

Table 32. Number of runs (out of 50 runs) converged within ϵ % of the best-known solution using real-coded DE with constraint handling scheme on test problem 1(True optimum solution =13.59085).

Method					ϵ			**Infeasible**	**Optimized $f_1(\vec{x})$**		
DE	≤ 1%	≤ 2%	≤ 5%	≤ 10%	≤ 20%	≤ 50%	≥ 50%		Best	Median	Worst
CR=0.9	47	47	47	47	48	48	2	0	13.5908	13.5908	26.7482
CR=0.5	48	48	49	49	50	50	0	0	13.5908	13.5909	15.8523
CR=0.1	17	23	31	40	43	44	6	0	13.5957	13.9536	58.7986

Table 33. Number of runs (out of 50 runs) converged within ϵ % of the best-known solution using real-coded DE with constraint handling scheme on test problem 3(True optimum solution=-15).

Method					∈			**Infeasible**	**Optimized $f_3(\vec{x})$**		
DE	≤ 1%	≤ 2%	≤ 5%	≤ 10%	≤ 20%	≤ 50%	≥ 50%		Best	Median	Worst
CR=0.9	50	50	50	50	50	50	0	0	-14.9888	-14.9094	-14.9569
CR=0.5	50	50	50	50	50	50	0	0	-14.9985	-14.9952	-14.9921
CR=0.1	50	50	50	50	50	50	0	0	-14.9979	-14.9898	-14.9762

Table 34. Number of runs (out of 50 runs) converged within ϵ % of the best-known solution using real-coded DE with constraint handling scheme on test problem 4(True optimum solution=7049.330923).

Method					∈			**Infeasible**	**Optimized $f_4(\vec{x})$**		
DE	≤ 1%	≤ 2%	≤ 5%	≤ 10%	≤ 20%	≤ 50%	≥ 50%		Best	Median	Worst
CR=0.9	49	50	50	50	50	50	0	0	7056.7	7065.9	7171.6
CR=0.5	0	0	0	39	50	50	50	0	7534.3	7706.3	7952.1
CR=0.1	0	0	0	1	40	50	0	0	7632.2	8248.9	8725.6

Table 35. Number of runs (out of 50 runs) converged within ϵ % of the best-known solution using real-coded DE with constraint handling scheme on test problem 5(True optimum solution=680.630573).

Method					∈			**Infeasible**	**Optimized $f_5(\vec{x})$**		
DE	≤ 1%	≤ 2%	≤ 5%	≤ 10%	≤ 20%	≤ 50%	≥ 50%		Best	Median	Worst
CR=0.9	50	50	50	50	50	50	0	0	681.0933	681.7487	683.0978
CR=0.5	50	50	50	50	50	50	0	0	685.0433	685.1968	683.4601
CR=0.1	18	49	50	50	50	50	0	0	682.3544	688.4198	697.4731

Table 36. Number of runs (out of 50 runs) converged using real-coded DE with constraint handling scheme on the Ackley function with 5 variables (True optimum solution=0).

Method	**Optimized $f_{Ackley}(\vec{x})$**		
DE	Best	Median	Worst
CR=0.9	8.8818e-16	4.4409e-15	4.4409e-15
CR=0.5	8.8818E-16	4.4409E-15	4.4409E-15
CR=0.1	4.4409E-15	4.4409E-15	7.9936E-15

Table 37. Number of runs (out of 50 runs) converged using real-coded DE with constraint handling scheme on the Rastrigin function with 5 variables (True optimum solution=0).

Method	**Optimized $f_{Rastrigin}(\vec{x})$**		
DE	Best	Median	Worst
CR=0.9	8.8818e-16	4.4409e-15	4.4409e-15
CR=0.5	0	0	0
CR=0.1	0	0	0

APPENDIX B: EXTRAPOLATED GUESSED ARRIVAL TIME

To obtain an extrapolated guessed arrival time in a dynamic network that utilizes a piecewise function, the following equation was developed.

$$t_E = \left[|\Delta N\#|^{\left(\frac{\#N}{\#L}\right)} \times Geo\ Mean\ S\&D\ L.C.\ \times (\bar{y} + 1) \right] + DT \quad (1b)$$

where t_E is the extrapolated guessed arrival time, $\Delta N\#$ is the difference in node number for the source and destination nodes, # N is the number of nodes in network, #L is the number of links in network, $Geo\ Mean\ S\&D\ L.C.$ is the geometric mean of source and destination link cost, $\bar{y}$ the is arithmetic mean of y, and DT is Departure Time.

It is the product of three main components plus the departure time given in the Equation (1c). That means t_E =[Average number of links between two given points in a given network ×Average link cost at source and destination nodes ×Average piecewise function time penalty] +Departure Time.

The first component of the equation uses a power function to estimate the number of links between any two points in a given network. The second component of the equation uses the geometric mean of the source node and destination node link costs to estimate the average link cost throughout the entire network. A geometric mean is used to obtain the "average" since the central tendency it delivers is less sensitive to wide variability in the data. An arithmetic mean is used to determine the time-based penalty of the network since an input value of zero gives a geometric mean of zero.

The inspiration for using a power function to estimate the number of links along the pathway was taken from the equation used to determine the fraction of a solute remaining in the

original water solvent after a solvent extraction. This is given by the following equation taken from p.145 of *Laboratory Techniques in Organic Chemistry 4th Edition*.

$$fraction\ of\ solute\ remaing = \frac{(final\ mass\ of\ solute)water}{(initial\ mass\ of\ solute)water} = \left(\frac{V_2}{V_2 + V_1K}\right)^n \quad (2b)$$

where V_1 is equal to the volume of organic solvent in each extraction, V_2 is equal to the original volume of water, n is equal to the number of extractions, and K is equal to the distribution coefficient.

The representation of the number of extractions n by a power function demonstrates the ability of a power function to accurately model independent events that are part of a bigger network (Mohrig, 2014). In this case, the network of performing multiple extractions on a solute. Therefore, a reasonable connection to dynamic networks was made with the intention to use it to estimate the average number of links to be traversed in a network from a given point to another. Each link functions as an independent event, but the outcome of each link impacts the outcome of the next link, impacting the pathway choice in a given network.

The power function also was raised to the number of nodes in a given network divided by the number of links in the network. The logic being that there can be only one link between any two given nodes that are on the shortest path between the source and destination nodes. This ratio of nodes to available links seems to serve as a viable estimator of the number of links between two points. The extreme values of the function are the value one and the value of $|\Delta N\#|^2$. The upper extreme value is arrived at since the smallest imaginable network would be two points connected by a single link in which case the ratio of nodes to links would be two. Given these extreme values

there will be at least always one link between the source and destination node, or potentially as many as the square of the numerical difference between these two nodes represented in the extrapolation equation.

The base of the power function is the absolute value of the difference in the numerical labels of the source and destination nodes on the given network. It is assumed that the network is numbered from one end of the network to the other, therefore the absolute value of the difference between the numerical labels of the source and destination nodes serves as a sufficient base to raise to the number of nodes in the network divided by the number of links in the network to.

Some possible improvements to this equation would be to include all of the link costs in the network when taking the geometric mean to come up with an average link cost to traverse the network. For the piecewise function time penalty, a geometric mean should be able to be taken if the value of one is added into the piecewise values before taking the average. This would remove the value of zero from the piecewise function, allowing the geometric mean to be taken. Again, the advantage of the geometric mean is that it is less sensitive to wide variability in the data.

The following extrapolation equation is proposed for obtaining the estimated arrival time.

$$t_E = \left[\frac{|\Delta N\#|^{\left(\frac{\#N}{\#L}\right)} \times Geo\ Mean\ S\&D\ Distance}{(Arith.\ \ Mean\ Network\ Velocity)} \right] + DT \qquad (3b)$$

where $Geo\ Mean\ S\&D\ Distance$ is the geometric mean of source and destination distances to all immediate connecting nodes and $Arith.\ \ Mean\ Network\ Velocity$ is the arithmetic mean of the velocity of all links in the network that is updated every ten minutes. The arithmetic mean is again employed as a safeguard against a velocity of zero, which would make the entire equation undefined. Therefore, an assumption of this equation is that there is always some traffic flow in

the network at any given time. Again, the geometric mean of the distance could be further improved by taking the geometric mean of all the distances between nodes in the network. The flip side of this would be taking the mean of only the source and destination nodes for the arithmetic mean network velocity. This localized mean might perform better than an entire network average.

For the Bureau of Public Roads function, the following equation is proposed. It represents the average number of nodes between two given points multiplied by the average link cost; given by taking the average of all input values for the entire network or only the source and destination nodes and plugging them into the Bureau of Public Roads function.

$$t_E = \langle |\Delta N\#|^{\left(\frac{\#N}{\#L}\right)} \times \left\{ \overline{t_{a0}} \times \left[1 + \bar{\alpha} \times \left(\frac{\overline{x_a}}{\bar{c}_a} \right)^{\bar{\beta}} \right] \right\} \rangle + D.T. \tag{4b}$$

which is taken from the following Bureau of Public Roads function for estimating link cost.

$$t_a(x_a) = t_{a0} \times \left[1 + \alpha \times \left(\frac{x_a}{c_a} \right)^{\beta} \right] \tag{5b}$$

where t_a is the travel link cost on link a, which is a function of x_a, t_{a0} is the travel time on the link a under free flow conditions, c_a is the capacity of the link a, x_a is the flow on a link, and α and β are given parameters.

VITA

Gelareh Bakhtyar Sanjabi

Department of Civil and Environmental Engineering

Old Dominion University
Norfolk, VA 23529

B.S. August 2003, Razi University, Kermanshah, Iran

M.S. August 2013, Old Dominion University, Norfolk, VA, USA