**Demo for grad project**

**Table of contents**

[1. Introduction 1](#_Toc98254945)

[1.1. Distribution systems optimization 1](#_Toc98254946)

[1.1.1. Complexity of Distribution systems optimization problems 1](#_Toc98254947)

[1.2. What is metaheuristics? 2](#_Toc98254948)

[1.2.1. Pseudo code of metaheuristics 2](#_Toc98254949)

[1.3. Archimedes Optimization Algorithm (AOA) 2](#_Toc98254950)

[1.3.1. Archimedes’ principle 2](#_Toc98254951)

[1.3.2. Theory 3](#_Toc98254952)

[1.3.3. Algorithmic steps 3](#_Toc98254953)

[1.3.4. Pseudo code of AOA 4](#_Toc98254954)

[2. Problem formulation for power system networks 5](#_Toc98254955)

[2.1. Backward/forward sweep (BFS) algorithm 5](#_Toc98254956)

[3. Results 6](#_Toc98254957)

[4. conclusion 6](#_Toc98254958)

# Introduction

## Distribution systems optimization

Shunt capacitors and DGs installation on radial distribution feeders is commonly used to improve power flow control, system stability enhancement and power factor correction, managing voltage profile, and reducing active power and energy loss in distribution network.

### Complexity of Distribution systems optimization problems

Distribution systems optimizations problems are too complex (comparable to NP problems) that it is computationally impossible to solve using brute-force search.

To get a sense of the complexity of this type of problems, consider a 34-bus distribution system like that in figure 1 and we are required to place 3 capacitors on those systems and the capacitors sizes must be within the boundaries 500: 1200 KVAR. If we considered bus 1 as the slack bus then for the first capacitor there is 33 possible location and the second there are 32 possible location and 31 possible location for the third. So in total we need to try possible locations.

If we considered only the integer values of the capacitors sizes (500 , 501,502,……,1198,1199,1200), then for each capacitor there are 700 possible value of its size. So there are (. So the search space with these assumptions would have candidate solutions.

We can get the mean time required to run the BFS algorithm in years and multiply it with the number of the candidate solutions to find the number of years of computations required to find the best solution (under the mentioned assumptions). The result is shockingly above 200 years.

If we considered values to 1 decimal number instead of integer sizes of the capacitors ( 500.1, 500.2,500.3,……, 1199.8,1199.9,1200). Then the result will be above 2000 years and in general the result would be .

Chart, diagram

Description automatically generated

Figure

## What is metaheuristics?

**Metaheuristics** is a subfield of **stochastic optimization**.

Metaheuristics are independent of the problem which means that can implemented to solve different types of problems.

***Stochastic optimization*** *is the general class of algorithms and techniques which employ some degree of randomness to find optimal solutions to hard problems. Metaheuristics are the most general kind of these algorithms and applied to a wide range of problems.*

***Brute-force search*** *means trying each possible solution and choosing the best of them*

In our case, we are trying to optimally place DGs and capacitors in a 34-bus network to improve power flow and voltage regulation. It’s impossible computational to solve this problem using brute-force search as the space we search is too enormous. We can always check the candidate solution and assess that it improves the power flow and voltage regulation of our network. A metaheuristic algorithm will be suitable for our problem.

### Pseudo code of metaheuristics

Table

Description automatically generated

## Archimedes Optimization Algorithm (AOA)

Archimedes optimization algorithm (AOA) generally emulates what happens when objects of different weights and volumes are immersed in a fluid.

### Archimedes’ principle

Archimedes’ principle states that “Any object, totally or partially immersed in a fluid or liquid, is buoyed up by a force equal to the weight of the fluid displaced by the object.”

Chart, diagram, box and whisker chart

Description automatically generated

Figure

### Theory

Objects in a fluid tries to reach equilibrium state. For an object to be in equilibrium state, the buoyant force must equal the object’s weight

Where is the density, is the volume, and is the acceleration. The subscripts and are referring to the fluid and immersed object, respectively.

If there is a collision between to objects the equilibrium state is reached when

### Algorithmic steps

AOA is a population-based algorithm. In the proposed approach, the population individuals are the immersed objects. Like other population-based metaheuristic algorithms, AOA also commences search process with initial population of objects(candidate solutions)with random volumes, densities, and accelerations. At this stage, each object is also initialized with its random position in fluid. After evaluating the fitness of initial population, AOA works in iterations until termination condition meets. In every iteration, AOA updates the density and volume of every object. The acceleration of object is updated based on condition of its collision with any other neighboring object. The updated density, volume, acceleration determines the new position of an object.

### Pseudo code of AOA

A picture containing text

Description automatically generated

# Problem formulation for power system networks

## Backward/forward sweep (BFS) algorithm

Diagram

Description automatically generated

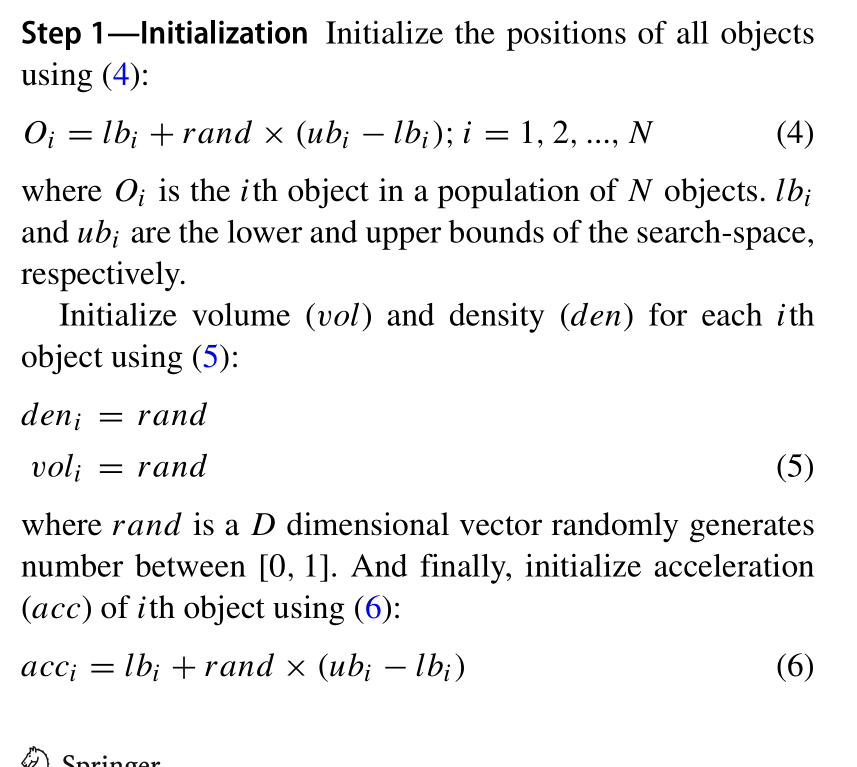
# Results

# conclusion

|  |  |
| --- | --- |
|  | (1) |

|  |  |
| --- | --- |
|  | (2) |

# Code



%==========================================================================

% Initial population and Initialization

%==========================================================================

%create the population

% for each device there are two objects: one for the bus and the other for

% the capacity

solutions{solutions\_no,2\*N\_caps+2\*N\_DGs}=objecte(2,3);

for i=1:solutions\_no

    for j=1:N\_caps

        solutions{i,j} =objecte(2,Nb);

    end

    for j=(N\_caps+1):(2\*N\_caps)

        solutions{i,j}=objecte(cap\_min,cap\_max);

    end

    for j=(2\*N\_caps+1):(2\*N\_caps+N\_DGs)

        solutions{i,j} =objecte(2,Nb);

    end

    for j=(2\*N\_caps+N\_DGs+1):(2\*N\_caps+2\*N\_DGs)

        solutions{i,j} =objecte(DG\_min,DG\_max);

    end

    for j=(2\*N\_caps+2\*N\_DGs+1)

        solutions{i,j} =result(Nb);

    end

end

classdef objecte

    %OBJECTE Summary of this class goes here

    %   Detailed explanation goes here

    properties

        x;

        den;

        vol;

        acc;

        acc\_norm;

        acc\_temp;

    end

    methods

        function m= objecte(lb,ub)

            %equ 4:6

            m.x= lb+rand\*(ub-lb);

            m.den=rand;

            m.vol=rand;

            m.acc=lb+rand\*(ub-lb);

            m.acc\_norm=m.acc;

            m.acc\_temp=m.acc;

        end

    end

end

this part of code constructs the “solutions cell array” like in fig

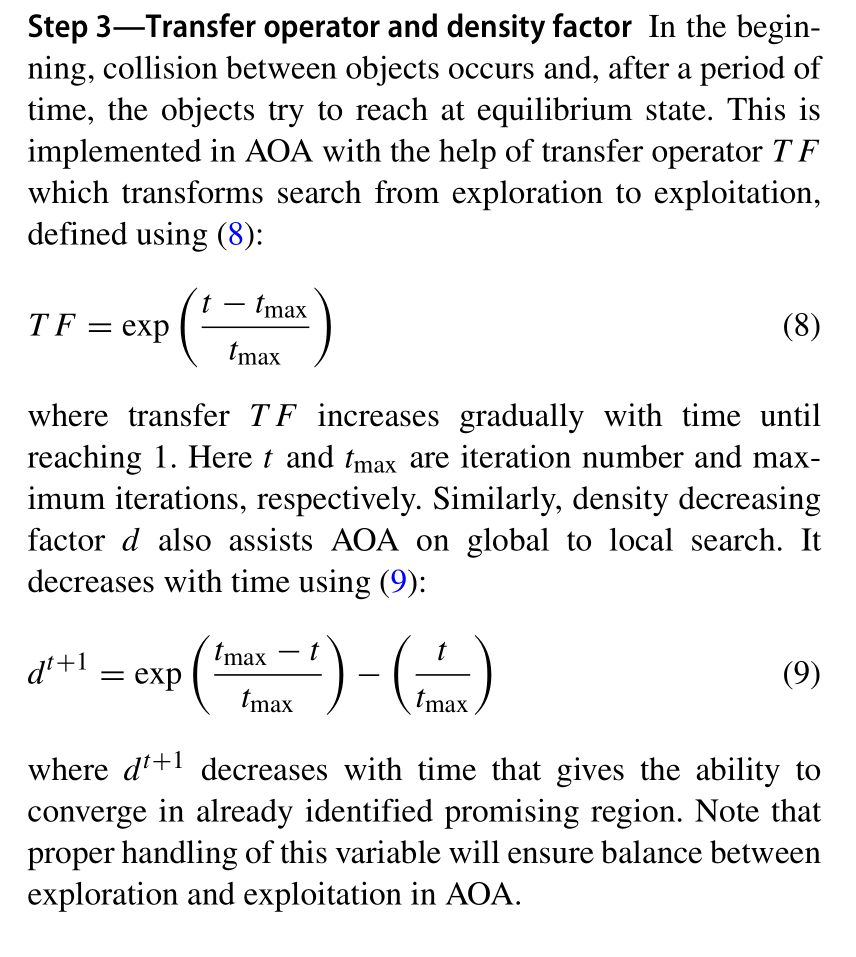
Graphical user interface, text, application

Description automatically generated

Mm1= [ 1 1 1]

V= [ .95 .94 .93]

Tvd = (1- .95)^2 +(1-.94)^2 +(1-.93)^2



%==========================================================================

%              Start the iterations -- AOA

%==========================================================================

for t=1:Max\_iter

    %======================================================================

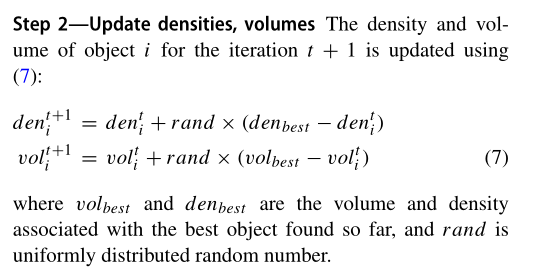
    TF=exp(((t-Max\_iter)/(Max\_iter)));                            % Eq. (8)

    if TF>1

        TF=1;

    end

    d=exp((Max\_iter-t)/Max\_iter)-(t/Max\_iter);



    for kk=1:solutions\_no

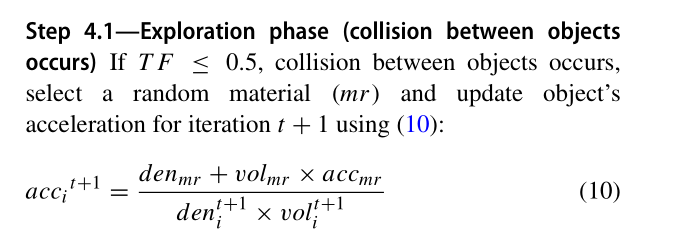
        for mm=1:(2\*N\_caps+2\*N\_DGs)

            %eq 7

            solutions{kk,mm}.den=(solutions{kk,mm}.den)+rand\*((best\_obj{mm}.den)-(solutions{kk,mm}.den));

            solutions{kk,mm}.vol =solutions{kk,mm}.vol +rand\*(best\_obj{mm}.vol- solutions{kk,mm}.vol);

          %collision

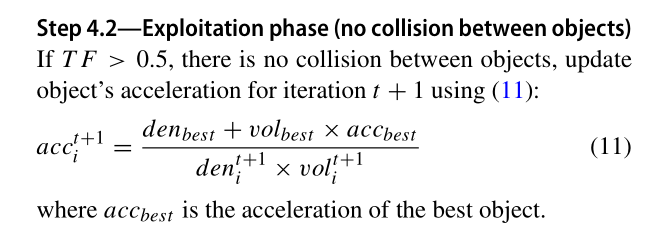


            if TF<.55

                mr=randi(solutions\_no);

                % Eq. (10)

                solutions{kk,mm}.acc\_temp=((solutions{mr,mm}.den+solutions{mr,mm}.vol\*solutions{mr,mm}.acc)/(rand\*solutions{kk,mm}.vol\*solutions{kk,mm}.den));



  else

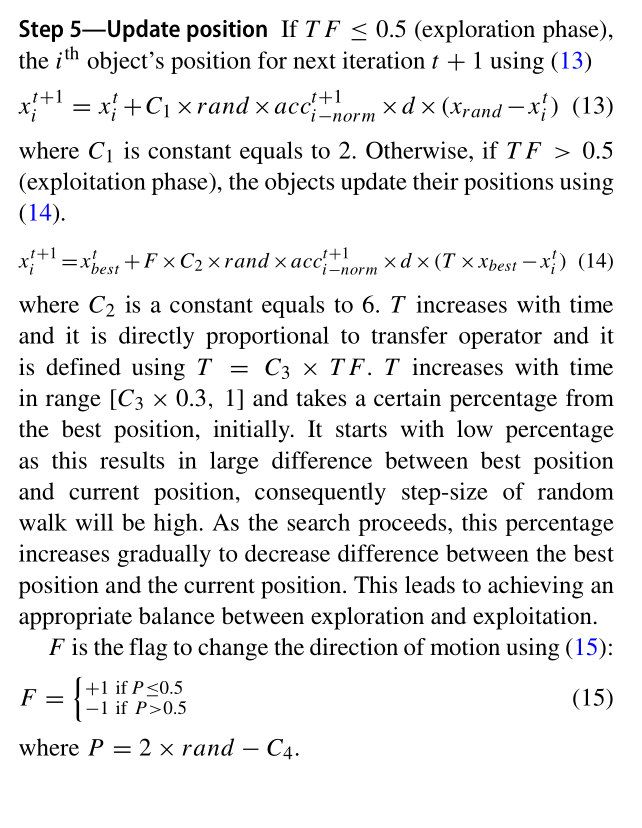
                %eq 11

                solutions{kk,mm}.acc\_temp=((best\_obj{mm}.den+best\_obj{mm}.vol\*best\_obj{mm}.acc)/(rand\*solutions{kk,mm}.vol\*solutions{kk,mm}.den));   % Eq. (10)

            end

        end

    end



%==========================================================================

% update position

%==========================================================================

    for v3=1:solutions\_no

        for vv3=1:(2\*N\_caps+2\*N\_DGs)

            if TF<.5

                mr=randi(solutions\_no);

                % Eq. (13)

                solutions{v3,vv3}.x=solutions{v3,vv3}.x+C1\*rand\*solutions{v3,vv3}.acc\_norm\*(solutions{mr,vv3}.x-solutions{v3,vv3}.x)\*d;

            else

                p=2\*rand-C4;                                    % Eq. (15)

                T=C3\*TF;

                if T>1

                    T=1;

                end

                if p<.5

                    % Eq. (14)

                    solutions{v3,vv3}.x=best\_obj{vv3}.x+C2\*rand\*solutions{v3,vv3}.acc\_norm\*(T\*best\_obj{vv3}.x-solutions{v3,vv3}.x)\*d;

                else

                    solutions{v3,vv3}.x=best\_obj{vv3}.x-C2\*rand\*solutions{v3,vv3}.acc\_norm\*(T\*best\_obj{vv3}.x-solutions{v3,vv3}.x)\*d;

                end

            end

        end

    end

fitness

Text

Description automatically generated

This outputs

The following arrays

Caps\_locations= [ 3 5 ]

Caps\_sizes = [ 800 900 ]

DGs\_locations= [ 6 8 ]

DGs\_sizes= [ 200 300 ]

Then check the required conditions and run BFS algorithm to get the total power loss or tvd