Optimal Placement of Shunt Capacitor Banks on a Sub-Transmission Network

prepared by: M. Ntusi

Student Number: 205518033

Dissertation submitted in partial fulfilment of the requirements for the degree of Master of Science in Power System Economics

Department of Electrical Engineering

University of KwaZulu-Natal

November, 2009

DECLARATION

As the candidate's co-supervisor I agree / ~~do not agree~~ to the submission of this dissertation.
Supervisor Name: Dr N Mbuli
Signature : [illegible] Date: 27 November 2009

I Marathon Ntusi (*205518033*) hereby declare that this is my own unaided work.

Signed [illegible] on the 27 of November 2009

(i) The research reported in this thesis, except where otherwise indicated, is my original work.

(ii) This thesis has not been submitted for any degree or examination at any other university.

(iii) This thesis does not contain other persons' data, pictures, graphs or other information, unless specifically acknowledged as being sourced from other persons.

(iv) This thesis does not contain other persons' writing, unless specifically acknowledged as being sourced from other researchers. Where other written sources have been quoted, then:

a) their words have been re-written but the general information attributed to them has been referenced;

b) where their exact words have been used, their writing has been placed inside quotation marks, and referenced.

(v) Where I have reproduced a publication of which I am an author, co-author or editor, I have indicated in detail which part of the publication was actually written by myself alone and have fully referenced such publications.

(vi) This thesis does not contain text, graphics or tables copied and pasted from the Internet, unless specifically acknowledged, and the source being detailed in the thesis and in the References section.

ACKNOWLEDGEMENTS

First and foremost I would like to thank the Almighty Lord who granted me wisdom, strength and tenacity to complete this research project

My sincere gratitude is extended to Dr N Mbuli, my project co-supervisor for his consistent guidance and tremendous support that He gave me during the course of the research.

My special thanks also go out to my adorable wife for affording me the space and opportunity to focus on the research when it was not always easy.

My current employer, Eskom, for the great privilege I was afforded to further my academic studies.

I am also indebted to Dr Azeem Khan for his assistance with Genetic Algorithm Matlab codes and course notes

My appreciation is also extended to Anne Pipe for proof reading the dissertation and enhancing the legibility of the report

RESEARCH ABSTRACT

The optimal capacitor placement problem is the determination of the optimal location of the shunt capacitors on the sub-transmission networks such that energy losses are minimised, the power factor and the network voltage profile are improved. During this period when Eskom is experiencing an unacceptably low generation reserve margin, it's quite critical that the electrical Transmission and Distribution network losses be kept to a minimum to optimise on the scarce generation that is available to supply South Africa's current and future power demand.

One of the ways of minimising technical losses is through the optimal placement or installation of capacitor banks on the network. The placement of shunt capacitors on a bulk Transmission network is essentially to improve the voltage profile on the network, increase system security and reduce transmission losses. The optimal placement of shunt capacitors with the above objectives would assist in minimising the cost of the investment whilst maximising the return on investment to the utility. This research subject is treated as an optimization problem and hence optimization solutions were considered to address the "Optimal capacitor placement problem". This optimisation problem is solved for all loading levels i.e. peak, standard and off peak periods and for different seasons in a given typical year.

This thesis investigates the capability of Genetic Algorithms technique in solving this optimisation problem. Genetic algorithms utilize a guided search principle to develop a robust solution to this research problem. Given their capability to traverse the complicated search space with a multivariate objective function, Genetic Algorithm are versatile and robust to locate the global optimum of the objective function. These Genetic Algorithms (GA's) were implemented on real sub transmission networks modelled on DigSilent/ Powerfactory. The modelled GA's on DigSilent were then tested on different network types i.e. commercial, mining, residential and industrial load mixes. The solutions determined by the different GA's are then compared in terms of time taken to locate the solution, reliability and robustness. The most reliable GA is then identified and recommended as the preferred optimisation approach. A methodology of using GA's to solve the above mentioned problem is therefore proposed.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AMBA	ADAPTIVE MUTATION BREEDER GENETIC ALGORITHM
BGA	BREEDER GENETIC ALGORITHM
CAP	CAPACITOR
CU	COPPER
DPL	DIGSILENT PROGRAMMING LANGUAGE
EA	EVOLUTIONARY ALGORITHM
E_{LOSS}	TECHNICAL (LINE CU & TRFM FE) ENERGY LOSSES
FE	IRON
GA	GENETIC ALGORITHM
CT	CURRENT TRANSFORMER
VT	VOLTAGE TRANSFORMER
HV	HIGH VOLTAGE
NERSA	NATIONAL ENERGY REGULATOR OF SOUTH AFRICA
PBIL	POPULATION BASED INCREMENTAL LEARNING
pf	POWER FACTOR
PPBIL	PARALLEL POPULATION BASED INCREMENTAL LEARNING
pu	PER UNIT
SC	SYNCHRONOUS CONDENSOR
SIL	SURGE IMPEDANCE LOADING
STATCOM	STATIC SYNCHRONOUS COMPENSATOR
SVC	STATIC VAR COMPENSATOR
TRFM	TRANSFORMER
V_{MAX}	MAXIMUM VOLTAGE ON A NETWORK
V_{MIN}	MINIMUM VOLTAGE ON A NETWORK

LIST OF APPENDICES

Page No

1. INTRODUCTION

1.1 RESEARCH SUBJECT

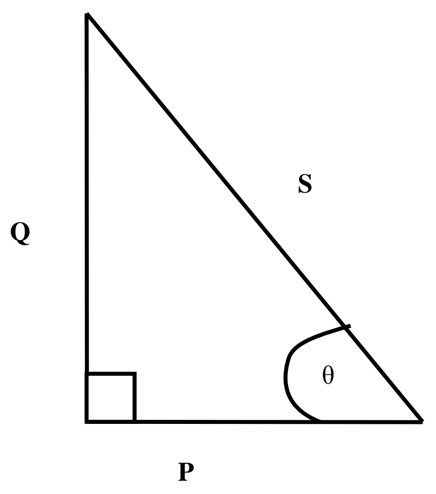
This report discusses different approaches to the optimal placement of capacitor banks on a sub transmission network. This is treated as an optimization problem and hence optimization solutions were considered to address the "optimal capacitor placement problem". Genetic algorithms, which utilize a "guided search" principle, were employed to develop a robust solution to this research problem.

1.2 BACKGROUND TO CAPACITOR PLACEMENT PROBLEM

The growing electrical demand and a subsequent shrinking in generation capacity is necessary for electrical networks be operated as efficiently as possible. A high demand in reactive power (i.e. MVArs) on an electrical network leads to depressed voltages, poor power factor (pf) and possibly high technical (copper and transformer iron) losses. One of the ways of efficiently operating the network is through optimal placement of capacitor banks to improve the network's voltage profile, release network capacity and minimize technical losses. At this critical time when South Africa is experiencing relatively low generation reserve margins, it is of utmost importance to maximize efficiency on Eskom's networks in order to ensure that maximum power is transmitted through the networks. This implies that network technical losses must be kept to a minimum for all network loading conditions. One of the strategies to be used in the minimization of losses is achieved through optimal placement of capacitor banks on an electrical network. These capacitor banks also provide reactive power support and improve the network voltage profile. This is to ensure that maximum efficiency from the electrical transmission network is achieved.

1.3 BASIC FUNDAMENTALS

Any power system can be represented by the following power triangles:



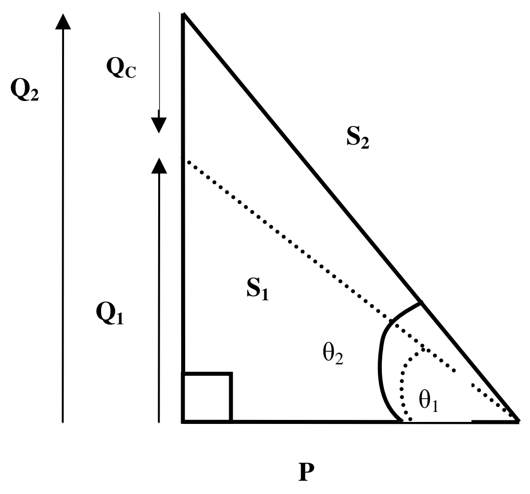


Figure 1.3(a): Uncompensated power system

Figure 1.3(b): Compensated power system

P → Active Power, Q → Reactive Power & S → Apparent Power

Figure 1.3(a) shows a power system without any reactive power compensation devices.

The power factor of the system pf = cos (θ) = $\frac{P}{S}$

Figure 1.3(b) is a representation of a network compensated by a capacitor C rated at reactive power Q_c. The diagram depicts that Capacitor C reduces the overall reactive demand of the power system. This clearly improves the power factor of the system by reducing the angle from θ_2 to θ_1, which in turn causes the cosine of the angle to increase. This does not change the active power demand of the system, but only compensates the reactive power supply.

It is important that this pf improvement does not only happen for peak loading conditions, but be applicable for all network loading conditions as the load varies throughout the day, week and season. The voltage limits also need to be kept within acceptable limits[1].

1. *Acceptable limits as per NERSA's NRS048 power quality directive voltage defined limits*

If the load on the compensated network increased, consider the following:

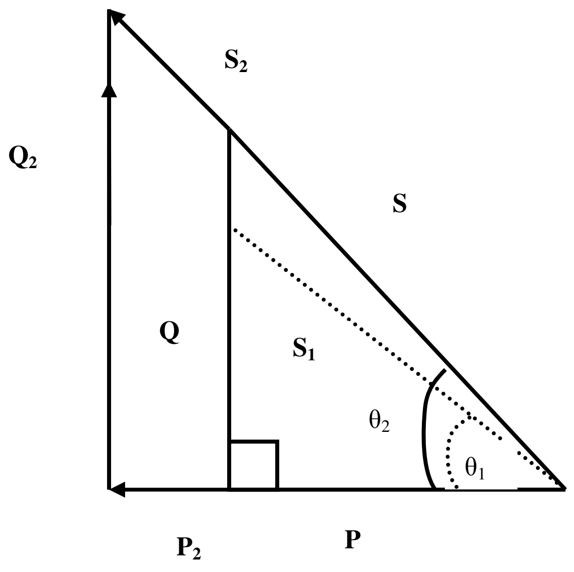


Figure 1.3(c): Compensated power system with increased load

Due to increased load on the network, the power factor improvement in Fig 1.3(c) would not be as much as in Figure 1.3(b). This is because the ratio of the increase in active power (from P to P_2) is less than the ratio of increase in reactive power (from Q to Q_2) [5]

1.4 NEED FOR REACTIVE POWER

The concept of reactive power is not as easy to grasp compared to active power, which delivers energy that does the real work, i.e. turns electric motors, illuminates lights, switches geysers on etc. Reactive power on the other hand oscillates between the source of power and the load, but never really gets consumed. Thus Reactive power never gets transferred to be consumed by the load unlike active power. During one half cycle, it flows from the source to the load and during the next half cycle, it returns to the source. Alternating Current (AC) systems are reliant on magnetic and electric fields for successful transmission of power from one end to the other [6], i.e. mainly from the source right down to the end user.

Reactive power can be defined as the control of the power system devices that regulate supply voltages in a manner that best achieves **reduction of losses and optimal voltage control** [7]

Reactive power management is subdivided into the following categories [7]:

- Reactive power long term planning
    - As network expansion plans are developed to cater for future power needs, it is critical that futuristic reactive power needs be also factored into the planning framework. This will ensure that in future there are no operational shortcomings associated with insufficient reactive power compensation, e.g. poor voltages or unacceptably low power factors.
- Operational planning
    - Reactive power requirements for different networks need to be determined in order to optimally deploy reactive power compensation devices where they are most needed, as in some areas reactive power needs may be easily catered for by generators.
- Reactive power dispatch as well as control.
    - Operational procedures and standards will have to be complied with to ensure that reactive power needs are catered for as quickly and efficiently as possible. This is to mainly ascertain that customer supply contractual obligations are strictly adhered to.

The objectives for each management category are for the security and economics of the grid. Reactive power is mainly needed to achieve the following [8]:

- In order to successfully transmit maximum active power from one area to another, healthy voltages on the interconnecting networks are required. Sufficient reactive power is needed to be able to sustain these healthy voltages.
- Inductive loads (e.g. motor-driven) rely on reactive power to create the magnetic and electric fields needed to produce the required work that drives the loads.
- Power transformers need reactive power flow to create the magnetic and electric fields needed to induce the Electro Motive Force/ Voltage across secondary windings, which will ensure that there is continuity of active energy flow from a higher voltage level to a lower voltage level.
- Reactive power is primarily needed to maintain a secure and stable transmission system for various levels of loading and network configurations.

1.5 NEED FOR GOOD VOLTAGE PROFILES

- Good voltages are essential to maintaining a stable and reliable electricity grid. Since the load on the grid varies with time, it is very crucial to maintain the voltages within required limits at all times.
- Electrical equipment is built to withstand certain voltage deviations, thus to avoid damage to network equipment and customer sensitive supplies. It is of utmost importance to maintain supply voltages within contractual obligations.
- With increased interconnections between grids of different sizes, different connection rules, different characteristics and unique sensitivities, it has become critical to maintain good and dependable network voltage profiles [9].
- Optimal voltage levels will also ensure that loads that are voltage-dependant do not unnecessarily consume large amounts of power, thereby saving on the cost of generation resources.
- When voltages drop lower than they should remain, rotating plant (motors) and transformers draw higher currents at the same power consumption. This may possibly lead to heating problems and cause technical losses to increase.
- Good voltage profiles provide for a good voltage regulation on the grid, thereby optimizing the cost of technical losses (copper and iron) as these are dependant on the voltage regulation limits.

1.6 CONSTRAINTS TO REALISING BENEFITS

There are a number of operational interventions that could be employed to improve the network voltage profile one of which is the optimal placement of capacitors. Although there is no doubt that capacitors have a technical capability of injecting the much-needed reactive power (MVArs) into the power system which helps to improve the voltage profile and minimize losses, it requires a well-thought-out strategy to carefully determine the optimal position(s) at which to connect these capacitor(s). When deciding on the placement of the capacitor banks, careful consideration will have to be given to a number of limitations and constraints. Among many of the considerations to be borne in mind are the following:

- Availability of adequate installation space (in the substation yard) for the capacitor bank.
    - Although it may be optimal to install a capacitor (as determined by a tried and tested methodology), it is crucial to confirm that there is enough room at the substation to erect a capacitor bank and also to ensure provision in the substation control room is made for a cap bank bay, capacitor breaker and control panel for its metering and tele-control circuits.

- Cost of installation and maintenance of the capacitor banks.
  - A benefit/ cost analysis will be necessary in order to ensure that indeed the cost of installing and maintaining the cap bank is outweighed by the cost of technical losses and perhaps poor voltage complaints from customers.
  - It must also be determined if there is a lucrative return on investment to be derived by pursuing the proposal to install the cap bank(s) at the identified position(s).

- Question on the operational mode of the banks to be considered
  - Switchable i.e. will the capacitor(s) ONLY be put in service when mostly needed to compensate MVArs on the network?
  - Fixed i.e. will the capacitor(s) be permanently put in service for all loading conditions and only switched out during maintenance (planned or unplanned)?
- Positioning capacitors on the network
  - Of utmost importance is a robust methodology on selecting positions to place the capacitors such that operational benefits are realized, not just for one loading condition (e.g. peaking time) and yet no operational benefits are experienced at other times. At the worst, the capacitor may cause over-voltage conditions or even result in network with a leading power factor.

1.7 VOLTAGE CONTROL DEVICES

The following are the different types of voltage control equipment [7]:

- Synchronous Condensers
  - Machines designed exclusively to provide reactive power support. It consumes approximately 3% of the reactive power rating (e.g. a 50MVar SC consumes approximately 1.5 MW).
- Capacitors and Reactors
  - Capacitors/ reactors are passive network elements that generate/ absorb reactive power. Their output is proportional to the square of the voltage e.g. for a capacitor bank rated at 100 MVar.

    If V = 0.95 p.u., the cap output will be ~ 90% = 90 MVar.

    If V = 1.05 p.u., the cap output will be ~ 110% = 110 MVar.
- Static Var Compensators (SVCs)
  - Similar to a synchronous condenser (i.e. can supply/ absorb reactive power), but it does not have rotating parts.
  - SVCs are composed of shunt reactors, shunt capacitors and thyristor switches.

- The degradation in reactive capability as the voltage drops requires harmonic filters to reduce the amount of harmonics injected into the power system.

- Static Synchronous Compensators (STATCOMs)
  - It is a FACTS device
  - Is similar to an SVC in response, speed and control capabilities
  - Instead of capacitors/ inductors, it uses power electronics to synthesize the reactive power.

- Transformers
  - Transformers provide the capability to raise/ lower voltages
  - By tapping the primary or secondary coil at various points, the ratio between the primary and secondary voltage can be adjusted. Fixed or variable taps provide +/- 10% voltage selection, with fixed taps typically in 5 taps and variable taps in 32 steps.

The following table contains characteristics of voltage control equipment [7]:

Table 1.7: Characteristics of voltage support equipment

EQUIPMENT TYPE	SPEED OF RESPONSE	ABILITY TO SUPPORT VOLTAGE	COSTS	
			CAPITAL PER KVAR	OPERATING
Generator	Fast	Excellent, Additional Short Term Capacity	Difficult To Separate	High
Synchronous Condenser	Fast	Excellent, Additional Short Term Capacity	Medium	High
Capacitor	Slow/ Stepped	Poor, Drops With V^2	Low	None
SVC	Fast	Poor, Drops With V^2	Medium – High	Moderate
STATCOM	Fast	Fair Drops With V	High	Moderate
Distributed Generation	Fast	Fair Drops With V	Difficult To Separate	High

The table above indicates the different types of voltage equipment in terms of speed of response and ability to control voltage and costs (capital and operating). It can be seen from the table that the capacitor has the least cost (low capital and no operating costs) compared to the other voltage support mechanisms. It, however, has the main drawback that its ability to control voltage is proportional to $\frac{1}{V^2}$. This implies that the capacitor needs to be switched into service before peak loading period, i.e. before the network voltages sag. This will ensure that maximum compensation is derived from the capacitor. An example of this is explained in 3.5, 2nd bullet point above. The other drawback is that the capacitor has a low response compared to the other devices. For the purposes of the scope of this research, only the capacitor voltage compensation device will be investigated.

1.8 SOURCES OF REACTIVE POWER

Most electrical equipment connected to the network either absorbs from or contributes reactive power into the electrical network. It is not always very economical to utilize some of the equipment to do voltage control. Reactive power can give rise to substantial voltage differences across different parts of the network, which requires a careful balance of reactive power flows between generators and load consumption points at various zones across the grid. The following are some of the sources of reactive power [10]:

- Synchronous Generators – Depending on the excitation applied on the generator, it can be run in a reactive power generating mode or reactive power absorption mode. Its reactive output will be determined by its operating range. Automatic voltage regulators govern the output MVArs by controlling the output voltage.

- Synchronous Compensators – Some small generators are run up to synchronous speed, and once synchronized, they are declutched from the turbines in order to only produce reactive power without producing real power.

- Capacitive and Inductive Compensators – These assist with keeping the network voltage levels within acceptable voltage limits. A capacitor creates an electric field which in turn generates reactive power. An inductor creates a magnetic field which absorbs reactive power. Compensation devices are available either as purely capacitive, purely inductive or even a combination of both capacitive and inductive (i.e. provision of either generation or absorption of reactive power).

- Overhead Lines and Underground Cables – Lines and cables generate electric fields which produce reactive power. A heavily-loaded overhead line creates magnetic fields, which cause the line to absorb reactive power. This is attributed to the dominant inductive properties of the line. On the other hand, a heavily-loaded cable is a net generator of reactive power due to the inherent capacitive properties of the cable.

- Transformers – Due to the windings of the transformer, the inductive properties of the transformer far outweigh its capacitive properties. This causes the transformer to be a net absorber of reactive power due to the magnetic fields it generates across its windings. The higher the loading on the transformer, the more reactive power it absorbs which in turn diminishes the spare capacity of the transformer to accommodate more active power that may need to be supplied to the connected loads.

- Customer Loads – Depending on the type of customer loads, some loads will absorb reactive power (e.g. motor loads, which create magnetic fields) and in some other instances, reactive power will be generated (e.g. fluorescent lights) by the loads.

1.9 BENEFITS OF REACTIVE POWER

The application of reactive power management brings the following benefits [7]:

- cost savings due to reduced system losses
- improved voltage control (not just locally but system wide)
- improved system security, greater reserve margins/ reserves available for contingencies and
- improved interchange transfer capabilities

1.10 PROBLEMS TO BE CONSIDERED

This research problem is aimed at mainly addressing the following key challenges:

1.10.1 The “optimal capacitor placement” problem must be solved for all loading conditions.

- The solution to the capacitor placement problem needs to be optimal for all loading conditions i.e. peak, standard and off peak loading periods. This is to ensure that the installed capacitors not only improve system efficiency through MVar compensation during peak and perhaps standard periods, but compromise system efficiency during off peak periods by causing over-voltages and possibly a leading power factor as well.

1.10.2 A method that could be applied on a real sub-transmission system needs to be developed

- The solution to this research problem must not be too complex to implement by System Planning and Optimisation Engineers in their current work environments i.e. preferably the solution must be developed for implementation on DigSilent Powerfactory Simulation Programme for Power engineers as the above mentioned Engineers already use the DigSilent simulation package for system studies (both network expansion/ strengthening and operational contingency studies).

1.11 OBJECTIVES OF THE RESEARCH PROJECT

In light of the above background, the main objective of the dissertation is to develop a realistic, easy-to-implement methodology on optimal placement of capacitor banks. This will include:

- Background literature study into the methods and algorithms used for optimal placement of capacitors on transmission networks, highlighting each method's advantages and disadvantages.
- Proposal and development of a method that could be used for a realistic transmission system.
- Development and implementation of the model on a realistic transmission system and run the model with realistic data.
- Discussion of the results and comparison to other methods to indicate the quality of the solution and robustness of the model.
- A write-up of the dissertation and submission for evaluation.

1.12 INFORMATION-GATHERING PROCEDURE

The information on which this dissertation is based was gathered by means of analysis of relevant Eskom Standards, IEEE journal papers, other journals related to optimal placement of capacitor banks and various relevant PhD theses. Eskom's real network models were used on this research as the solution is intended to be applicable to these networks.
Customer load profiles were sourced from Eskom's customer consumption management systems and used in the network models to accurately perform realistic power flow simulations.

1.13 SCOPE AND LIMITATIONS

The scope of this research is limited to Sub-transmission networks (44 – 132kV) as per Eskom's standard voltages definition. Although there are many ways of solving this research problem, this research investigates the use of genetic algorithms to solve the above mentioned optimization problem. Genetic algorithms (GA) have been specifically chosen due to their versatility, robustness and reliability of consistently finding solutions to multi-objective optimization problems. In this case an optimal capacitor position (network busbar or terminal) needs to be located for various levels of network dynamics (i.e. different loading conditions and different network configurations).

1.14 PLAN OF DEVELOPMENT

Chapter 1 gives an introduction to the whole research. The background to the research problem is looked at in order to clearly define the goal of the report. The objectives of the research are mentioned; the scope, limitations and plan of development of the research are stated in order to ensure the research is focused and bears fruit.

Chapter 2 reviews the relevant literature in order to do an assessment of similar methodologies or approach that already exists in industry. GA-based methods are considered. Advantages and disadvantages of GAs are carefully considered.

Chapter 3 defines the optimisation problem broadly and later narrows it down to the formulation of the specific research problem aimed at being addressed by this dissertation.

Chapter 4 outlines the application considerations for Shunt Capacitors in terms of different network characteristics, quality of supply, system stability and voltage regulation.

Chapter 5 gives an overview of genetic algorithms in terms of their operational strategy, attributes and main strengths as to why they are the preferred solution. GA's implementation on DigSilent is also discussed in terms of the main features developed.

Chapter 6 outlines the implementation of chosen Genetic Algorithms. Simulation results are presented and considered algorithms are ranked by their reliability and robustness.

Chapter 7 outlines the main conclusions and key recommendations of this research.

Chapter 8 lists the references used in the research, and, finally,

Chapter 9 tables the appendices which includes the converted DPL scripts that were coded on DigSilent.

2. LITERATURE REVIEW ON OPTIMAL PLACEMENT OF CAPACITOR BANKS ON A SUB-TRANSMISSION NETWORK

2.1 INTRODUCTION

For the main purpose of ensuring that a viable methodology for optimal placement of capacitor banks is developed, it is crucial to do a thorough literature scan to determine, compare and interrogate the various approaches already developed. It is also important to narrow the scope of the literature survey to the following:

- "Genetic algorithm"- based methodologies, as this research explores a genetic algorithm driven solution.
- Solutions that would be implemented on a power system simulation package already used by engineers in the system operational and planning departments.

In order to be able to formulate a robust and balanced solution, the following methods in literature are reviewed:

- exhaustive search;
- linear deterministic optimisation model;
- simulated Annealing;
- tabu search; and
- evolutionary approach

2.2 ADVANTAGES OF USING GENETIC ALGORITHMS

The following are the advantages of a genetic algorithm-based solution to the optimal placement problem of capacitors:

- Instead of using traditional exhaustive search methodologies (which take a long time to locate a solution for multivariate objective functions), genetic algorithms randomly search a pool of potential solutions based on well-guided search techniques. Genetic algorithms are therefore able to efficiently find a solution to sophisticated engineering problems in the specific search space.
- Industrial problems are usually multi-objective in nature e.g. optimise power flow to minimise technical losses subject to certain voltage constraints. Genetic algorithms are best suited to model engineering problems that are usually "multi-objective in nature".
- Genetic algorithms are versatile when traversing the search space for an optimal solution, i.e. if a solution to an engineering objective function is optimal locally, the GA will adapt the search in pursuit of a globally optimum solution subject to predefined search constraints.

2.3 DISADVANTAGES OF USING GENETIC ALGORITHMS

- Implementation of GAs on an existing power system simulation tool requires the GA code to be converted to a language recognized by the simulation programme (e.g. DigSilent Programming Language for DigSilent Power Simulation Package).
- The GA code can be very long or can have many subroutines, thereby becoming complicated to implement.
- Due to the non-linearity of most objective functions, Genetic Algorithms are difficult to solve as it requires the calibration of complex and arbitrary parameters that depend on operation conditions and distribution systems characteristics

2.4 OPTIMISATION METHODS

The methods discussed in this section are among the many relevant methods to the research topic. The methods are then reviewed to be able to identify methods which efficiently yield optimal results.

2.4.1 EXHAUSTIVE SEARCH METHOD

This approach exhaustively searches for an optimal solution in the search space. The search is not guided and this may lead to the method taking a long time to locate the solution. Consider the following example:

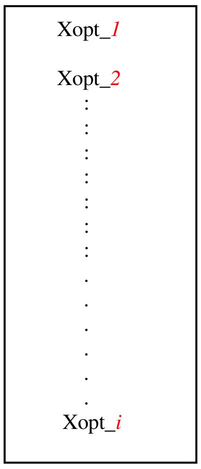


Figure 2.4.1: Depicts a vector of optimal solutions

In Figure 2.4.1, 1 is a vector with *i* optimal solutions. The method searches from optimum solution 1 to optimum solution *i* regardless of whether the solution is in the 10th position, 100th position or right at the end of the vector. This method thoroughly searches the search space (vector of optimal solutions in this case) to locate the optimum solution.

The main disadvantage is that if the search space is large (e.g. in the case of multi objective functions), the computation time increases exponentially.
In the case of power systems, the method works well for a system with few buses. Typical sub-transmission networks have many buses and as a result this method becomes less favourable.

2.4.2 LINEAR DETERMINISTIC OPTIMISATION METHOD

This method considers reducing investment costs and energy loss as the optimisation problems. These optimisation problems are addressed through a linear approximation and variable representation strategies [1]. This method transforms classical problems into mixed linear integer optimization problems. Lower and upper voltage limits may need to be relaxed if cost saving is significantly higher than voltage improvement [1]. This method considers minimising the following optimisation parameters as its objectives:

- investment costs of installing capacitors; and
- the total energy loss costs over the planned installation period

The following figure represents the loss linear approximation theorem.

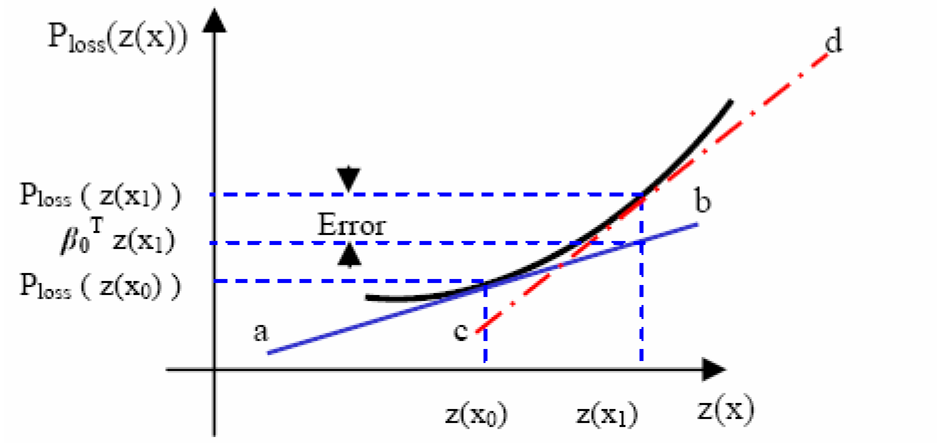


Figure 2.4.2: Linear approximation of active power losses (Picture adapted from [1])

The dark nonlinear curve above represents actual active power losses in a particular branch. Let $\mathbf{z_0} = \mathbf{z(x_0)}$ be a unique vector of possible solutions corresponding to the operating point $\mathbf{x_0}$ in branch i. e.g. **z** can be considered as $\mathbf{z} = [\mathbf{V}_i^2\, \mathbf{V}^2_{i+1}\, \mathbf{P}_i]^{\mathbf{T}}$ [1].

For each value of z, there exists a linearly approximated value of P_{loss}. As can be seen in Figure 2.4.2, line segment a - b would introduce errors when estimating $P_{loss}(\mathbf{x_1})$. As a result this method uses a better linearization approximation, i.e. line segment c - d. The method generates several line segments to approximate or mimic the non-linear loss function as the one depicted in Figure 2.4.4. This method also considers installing fixed size and switched capacitor banks. With fixed size capacitors, different load levels are considered.

Advantages of the linear deterministic optimisation method [1] are:

- it converts a non linear optimisation problem into a linear one which is less complex to solve;
- the optimisation problem is solved by a complete deterministic procedure with no need of applying random optimisation methods;
- very few load flow simulations are done due to the linearization approach that this method employs; and
- calibration of complex and arbitrary parameters is avoided.

Disadvantages of the linear deterministic optimisation method [1] are:

- the main disadvantage of this method is the errors that are introduced due to using linear approximations instead of solving the real non-linear function; and
- some real engineering problems are difficult to get a linear approximation for, thus this method may lead to very inaccurate solutions.

2.4.3 SIMULATED ANNEALING

This method uses a hybrid algorithm based on simulated annealing and greedy search techniques [2]. The method analyses constant power load customers by breaking up a year as follows:

Peak period = 1000 hours, **Nominal** period = 6760 hours and **Off peak** period = 1000 hours.

Constant current and constant impedance loads are not considered. The objectives this method considers minimising are the price of capacitors and the total energy loss costs.

Operational constraints are also considered, i.e. voltages to be kept within certain defined limits and transformer tap changers to be optimised to ensure that supply voltage are within limits. Two types of simulated annealing algorithms are considered; homogeneous and inhomogeneous [2]. For the homogeneous one, Markovian chain has constant control parameter for all its solutions. There are several Markovian chains for which the control parameter decreases gradually. “For the inhomogeneous one, there is one Markovian chain with different control parameters for the successive pairs of solutions” [2].

Branco and Milos express Simulated Annealing by the following scheme in Pseudo-Pascal [2]:

- start with any initial solution,
- perturb from current (*j*) solution to the next one (*i*) ,
- find ΔCij ,
- if $\Delta Cij < 0$ replace the solution j by solution i,
- if not find exp ($-\Delta Cij / c$) ,
- if exp ($-\Delta Cij / c$) is greater than random number uniformly distributed in segment [0,1), replace the solution *j* by solution *i* (Metropolis criterion),
- if not, retain current solution *j*.
- stop when system is frozen i.e. there is no noticeable improvement in the solution.

The algorithm follows the following sequence [2]:
Step 1 - Input data.
Step 2 - Starting with initial feasible configuration (″bare″ network).
Step 3 - Cooling schedule.
Step 4 - Generate new feasible configuration.
Step 5 - Metropolis criterion.
Step 6 - Checking of stop criterion.
Step 7 - Output file

Advantages of the Simulated Annealing optimisation method [2]

- The speed of convergence of this algorithm is fast.
- The algorithm is versatile to be applied for different network configurations.
- The algorithm runs load flow simulations for different load levels and takes voltage profiles for each load level into account.

Disadvantages of the Simulated Annealing optimisation method [2]

- In the control loops, the greedy search technique (instead of simulated annealing) is used to minimise CPU computation time.
- More restrictive input parameters drastically increase the computation time.
- Sometimes the algorithm is prone to terminating with worse solutions and hence needs monitoring during execution.

2.4.4 TABU SEARCH

"Many computational experiments have shown that Tabu search has now become an established optimization technique" [3]. This method can now compete with almost all known techniques and by its flexibility can beat many classical procedures [3]. The essential feature of the Tabu search methodology is that it keeps track, not only of local information, but information related to the entire exploration process. This method keeps record of good solutions and seeks to improve on the local optimal solutions. The following algorithm is employed by the Tabu search method [3]:

Step 1. Choose an initial solution i in S. Set i* = i and k = 0.

Step 2. Set k = k + 1 and generate a subset V* of solution in N(i,k) such that either one of the Tabu conditions tr(i,m) $\in$ Tr is violated (r = 1,...,t) or at least one of the aspiration conditions ar(i,m) $\in$ Ar(i,m) holds (r = 1,...,a).

Step 3. Choose a best j = i, m in V* (with respect to f or to the function f~) and set i = j.

Step 4. If f(i) < f(i*) then set i* = i.

Step 5. Update Tabu and aspiration conditions.

Step 6. If a stopping condition is met, then stop or go to Step 2.

Advantages of the Tabu search optimisation method [3]

- This method ensures effective computing as for each iterative step it evaluates the solutions.
- This method efficiently uses the memory to help intensify its search for optimal solutions.
- The search is diversified to ensure that the "less frequently" visited regions of the search space are also considered.

Disadvantages of the Tabu search optimisation method [3]

- The search for an optimal solution is very intensive and hence the method is complex to implement.
- The mean value of Tabu solutions grows with the increase of the Tabu list size.

2.4.5 EVOLUTIONARY APPROACH

An evolutionary algorithm creates a population of solutions to evolve through the application of recombination, mutation and natural selection operators. Fitter individuals are able to survive longer, thus perpetuating their genetic information [4]. After several generations, the population is expected to be composed of high-quality individuals, which will be a representation of good solutions.

After trial parent solutions are selected, they become utilised as input parameters to the recombination operator. The recombination returns a new individual or offspring [4]. The mutation operator then adds diversity to the population of individuals. The mutation operator is mainly divided into two parts i.e. "The first one modifies the binary portion of the chromosome by choosing a position of the individual at random and changing the allele's value (*bit-swap*). The second part acts on the integer values by adding or subtracting a unity from its value. The choice to add or subtract is also decided at random" [4].

Below is a simplified pseudo-code of an implemented Evolutionary Algorithm [4]:

```
a. Create the initial population.
b. Select individuals for recombination.
c. Recombine selected individuals and create new ones through crossover mechanisms.
d. Mutate the new individuals.
e. Apply a local search operator on the new individuals.
f. Insert the new individuals into the population, replacing the worse ones. Go back to b.
```

Advantages of the Evolutionary Algorithm optimisation method [4]

- This method does not restrict the number of times a given individual takes part in the crossover within the same generation. This increases the chances of finding a globally-optimum solution.
- The mutation operator adds diversity to the population which enhances the algorithm to locate robust solutions.

Disadvantages of the Evolutionary Algorithm optimisation method [4]

- This method is a bit complex to implement.

3. DEFINITION OF THE RESEARCH PROBLEM

3.1 INTRODUCTION

It is crucial that an optimal position for capacitors be determined on a sub-transmission network in order to ensure that network voltages are improved, the power factor is improved and system losses minimized. This leads to a realization that the problem to be solved is of an optimization nature; hence optimization approaches that will give realizable, robust and reliable results are required to solve this optimization problem. This then leads to formulating the multi-objective optimization problem as it applies to the placement of capacitor banks on a network and to defining what the problem is that is to be solved in this dissertation.

3.2 OVERVIEW OF THE OPTIMISATION PROBLEM

In engineering terms optimization can be defined as a process of finding the most suitable parameters of a system that will improve the performance of that particular system towards the optimum performance [11]. The optimization process is then faced with the problem of tuning the design parameters such that the desired result is attained. To illustrate this consider the following examples:

3.2.1 OPTIMISATION EXAMPLE

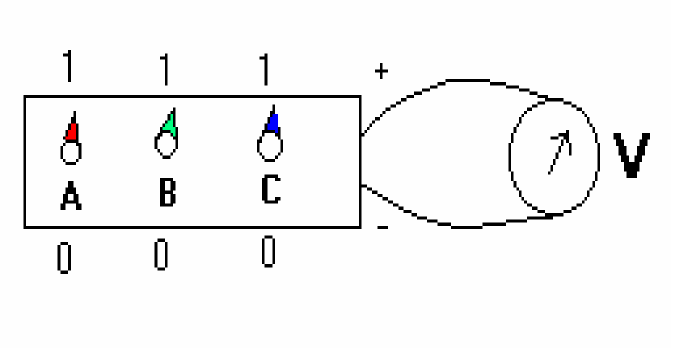


Figure 3.2.1: Shows 3 switches A, B, C toggled between 1 and 0

Consider the problem of maximizing the output voltage of some system as depicted in Figure 3.2.1. The system comprises three switches; A, B and C which, when flipped up, turn on (represented as logic 1) and when flipped down, turn off (represented as logic 0). A certain binary string combination of input A, B and C will maximize the output voltage of the system measured by the voltmeter V [17].

The optimization problem then, is to find a combination of these switch states of switches A, B and C that will maximize the output voltage. Different optimization techniques have been developed in literature, a few of which will be considered in this dissertation project.

- One intuitive approach would be to search for the solution by considering all the possible combinations of the switch states. This approach works very well if the number of decision variables are not many (in this case the decision variables would be A, B and C); however in real world problems this does not often hold to be true, many problems are usually characterized by far more than three variables. The downfall of this approach is that the number of possibilities that should be considered increases exponentially with the number of decision variables, i.e. if x is the variable, the number of possibilities equals 2^x.
- Another approach would be to randomly search for optimum combinations that would yield the maximum value of the output. However, in this approach, the search is guided towards finding the best solution. Referring to Figure 3.2.1 above, this approach means that, for instance, if the combination A = 1, B = 0, C = 1 improved the output (in either maximizing or minimizing sense depending on the application), then, when the search continues to search for better solutions, it would have to bias its search by favouring the states A = 1 and B = 1, since these seem to improve on the output result and a combination that degraded the output would not be favoured. Although in a very large search space there are less chances of finding a solution by mere random searching, the random search can be effective if it is biased towards the solution by only considering solutions that better or improve the output. This is the principle on which evolutionary search methods and stochastic hill climbing search methods work, e.g. PBIL, BGA, etc.

3.2.2 OPTIMISATION PROBLEM REVISITED

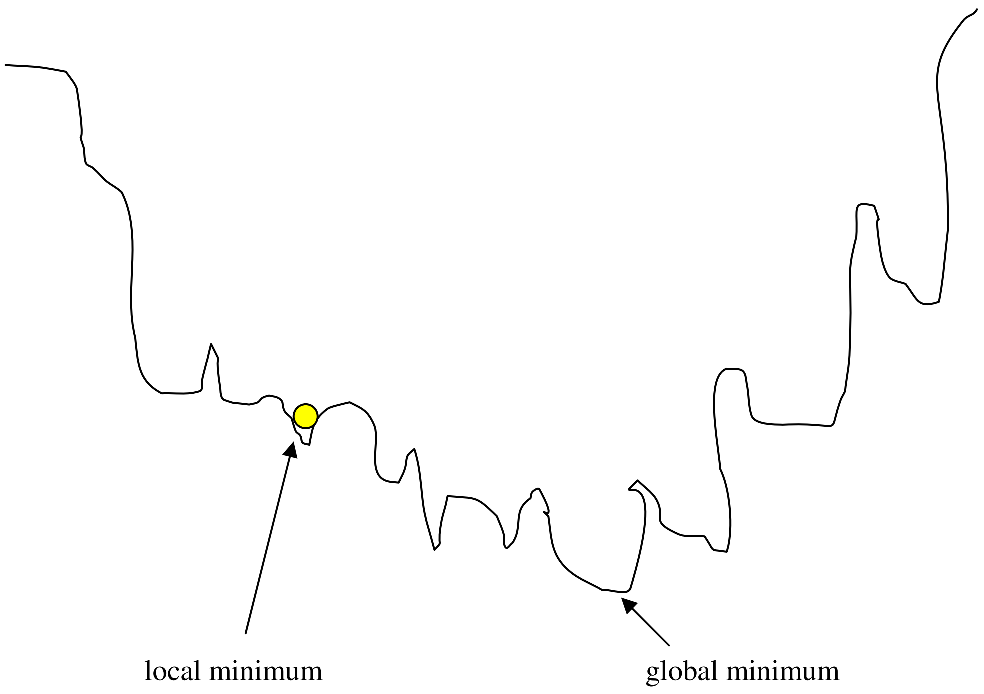


Figure 3.2.2: Some arbitrary multivariable function to be minimized

Secondly, let us consider some arbitrary multivariable function to be optimized (in this case to be minimized). In real engineering problems it is quite rare to have smooth, regular functions that only have one minimum that would be both local and global minimum. Due to the nonlinear nature of the engineering problems, there appears to be some irregularities on the behaviour of the objective function that describes the problem.

The problem that the search techniques would suffer from is to get trapped in some local minimum as depicted in Figure 3.2.2, where the search algorithm is analogous to a tossed marble (the yellow ball in Figure 3.2.2) over some irregular surface. The chances that the marble (yellow ball) will roll over and end up in the global minimum position are very limited, due to the irregularity of the surface. Due to this undesirable shape of the objective function, search algorithms that are very robust and diverse need to be employed to ensure that the whole search space is explored to avoid local optimum points.

3.3 OPTIMISATION PROBLEM FORMULATION

The optimization problem is often one that requires several objectives to be optimized; for instance, consider the following:

- Optimal power flow to minimize system losses and optimize energy flow.
- Optimal Capacitor placement for improved system power factor.
- Optimisation of generator parameters to increase its efficiency.
- Minimization of design costs and maximization of efficiency, etc.

Hence, the Optimization problem can be formulated as a general constrained problem. Generally the Optimization problem comprises of *n* decision variables (design parameters to be optimized), *k* objective functions and *m* constraints. Thus, mathematically, the goal we seek to achieve is:

Maximize / Minimize $F(\mathrm{x}) = \{f_1(\mathrm{x}), f_2(\mathrm{x}), f_3(\mathrm{x}) \ldots\ldots f_k(\mathrm{x})\}$…… (Objective function)

Subject to: $g_i(\mathrm{x}) \leq 0$ and $\mathrm{h_i}(\mathrm{x}) = 0$ …………………………… (Constraint # 1)

$\mathrm{x_{min}} \leq \mathrm{x} \leq \mathrm{x_{max}}$ ……………………………. …….. (Constraint # 2)

In the context of this research, the above mentioned optimisation problem definition can be applied as follows:

- *F*(x) represents the objective function which seeks to minimize technical losses on the sub-transmission network. The technical losses objective function is the summation of:
    - Copper losses of all overhead lines and underground cables on the sub-transmission network of interest.
    - Copper losses in the transformer's primary and secondary windings. These are for all transformers (two windings and three windings) on the sub-transmission network under consideration.
    - Transformer iron losses as a result of the magnetic flux that escapes from the core thereby failing to couple the primary and secondary windings of the transformer. This inefficiency is attributed to the quality of the transformer core magnet and insulation between the core laminations that determine the quantity of the undesirable eddy currents that flow in between the core laminations.
- x is a decision variable (s) which determines the position of the Capacitor(s) that will ensure that the technical losses *F*(x) alluded to above is minimized.
- *g*(x) is a vector of nonlinear inequality constraints e.g. a Planning or Operations Engineer might wish to install a Capacitor to ensure the voltage or pf remains between some desired range in order to cater for sensitive customer quality of power supply requirements.
- h(x) is a vector of equality constraints. x_{min} and x_{max} are vectors of lower and upper bounds on the design variables, e.g. it might not be feasible to place a Capacitor at some of the network busbars.

3.4 CAPACITOR CONTROL OPTIONS

A capacitor bank can be operated in two operating modes i.e. fixed or switched modes.
Fixed mode – the capacitor is permanently put in service and is only taken out of service during either planned or unplanned maintenance.
Switched mode – the capacitor is only put in service when needed or during predefined times. Switched banks require a switch and control circuit/ device, which makes it more costly compared to the fixed bank [21]. The main reason why Capacitor banks would be switched out of service is to avoid over-compensating the power system such that the power factor becomes leading and possibly even result in over-voltages.

3.4.1 CAPACITOR CONTROL OBJECTIVES

Another critical challenge of the Capacitor placement problem is to ensure that the effectiveness of Capacitor control is optimal in order to better respond to daily and seasonal demand profiles.

As stated in the introduction of this research, it is quite crucial to ensure that the power system is adequately compensated (through MVAr support) for different loading conditions and not just during the peak loading period. There are a number of ways in which Capacitor control can be achieved for switched type capacitor banks. Among many of these are:

- **Time based control**

- This type of control mechanism switches the Capacitor into service at pre-determined time intervals. The on/off times of the Capacitor are estimated from the known load profiles of the loads that are connected to the network to which the Capacitor is connected. This may of course have its drawbacks due to the fact that the networks often feed different load mixes with different loading characteristics, which causes the load profiles to be unpredictable at times. The new loads that are added to the networks also bring another complication to predicting an overall accurate load profile for the network of concern.

- **Temperature based control**

- This works on temperature limits that are linked to the reactive power. The Capacitor is switched on and off for temperature limits. This applies to loads whose reactive demand varies with temperature, e.g. air conditioning motor load. This control type mechanism assumes that the load characteristics are known.

- **Reactive Power/ VAR based**

- Limits of reactive power through a selected branch are determined. When the reactive power through the monitored network branch(es) exceeds the predefined limit, the Capacitor is switched into service for compensation. This type of control is also known as power factor control.
- An auxiliary transformer and a current transformer are needed for sensing the voltage and the current [21].

- **Network Voltage Control**

- This type of control is used to regulate network voltage profiles in order to ensure that the network voltage of interest is kept between predefined voltage limits. As a result, when the voltage deviates outside the desired range, the Capacitor is triggered to compensate for the voltage deviations, i.e. if the voltage drops below the low target setting, the Capacitor is switched into service and if the voltage increases above the high target setting, the Capacitor is switched out of service.

- The control mechanism monitors the voltage for various levels of network loading in order to ensure that the network voltage at any point in time remains within the desired regulation window. Consider the following figure below:

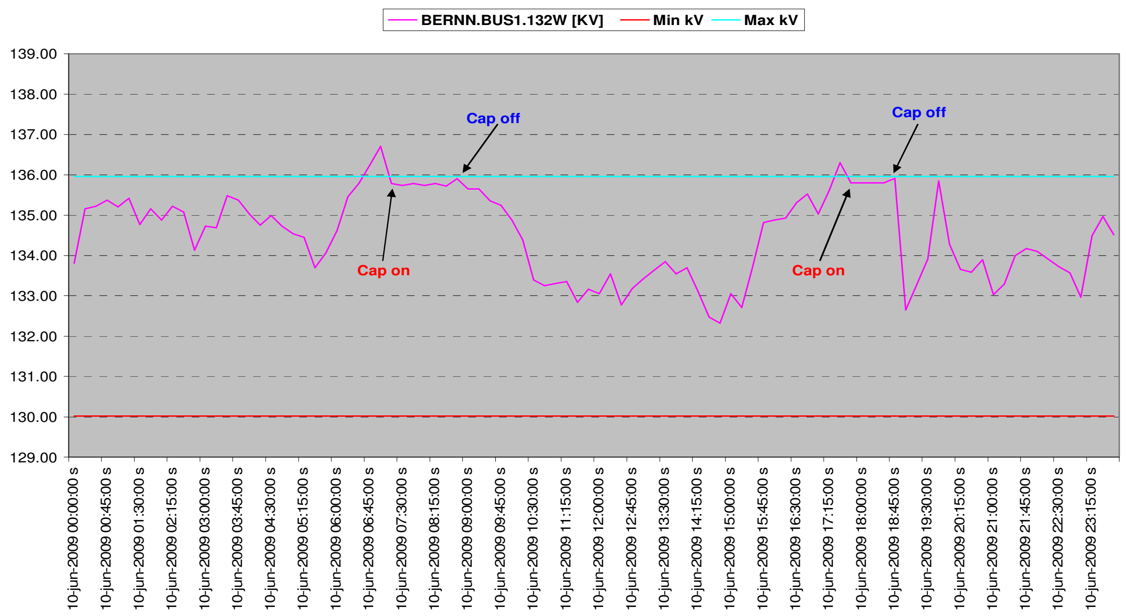


Figure 3.4.1: Shows a network voltage profile with a switchable cap

- The Figure above depicts a 132kV network voltage profile (for a day) with a voltage upper limit (blue line) and a low limit (red line). When the voltage exceeds the upper limit, the Cap is switched on and this can be seen through the sudden step change in voltage as shown on the figure above.

- **Current based Control**
- In a similar way to the reactive power control, this form of control uses current loadings to trigger when a Capacitor should be switched on/ off. It requires CT's to sense the current through the network corridor being monitored.

3.4.2 CAPACITOR VOLTAGE LIMITS

The size or rating of any Capacitor can be determined using the following expression:

$$kVAr_{\text{per phase}} = \frac{2\pi fCV^2}{1000} \quad [21]$$

Where: f is the grid frequency in Hertz (Hz)

C is the capacitance in micro Farads (μ F)

V is the network voltage where the cap is connected in (kV)

The above formula indicates the relationship between the voltage applied across the capacitor bank and the reactive power produced by the Capacitor. If we ignore the variation of the frequency on the grid and therefore assume that *f* and C are constant, it is clear that there is a parabolic relationship between the Capacitor MVArs and the system kV, i.e. as the voltage varies, the Var output of the capacitor varies by a voltage squared factor. It is because of this mathematical relationship that Capacitors need to be switched into service shortly before the network voltages are depressed due to the rising load profile. For example, a drop in 2% on the system voltages will cause an expected drop of approximately 4% in the Capacitor Var compensation. The increase in voltage, especially if prolonged, may sometimes lead to high currents which may cause excessive heating inside the bank and possibly cause damage to the bank [21]. Having stated the above, it is thus imperative to ensure that the Capacitor's over-voltage limits are adhered to in order to avoid the failure of the Cap bank. The limits are as follows:

Table 3.4.2: Capacitor over voltage limits (Table Adapted from [21])

DURATION	RATED VOLTAGE MULTIPLYING FACTOR
½ CYCLE	3.0
1 CYCLE	2.7
15 CYCLES	2.0
1 SECOND	1.75
15 SECONDS	1.4
1 MINUTE	1.3
5 MINUTES	1.2
30 MINUTES	1.15
CONTINUOUS	1.1

3.5. DISCUSSION AND REFOCUSSING

This chapter attempted to describe the optimization problem as it applies to engineering problems in general and ultimately narrowed it down to how it applies to the optimal placement of capacitors optimisation problem. The most crucial point to realize is that the optimisation problem can be represented by the objective function *F(x)*. In this particular research, the *F(x)* is technical losses on a Sub-transmission network and x is a vector representing the position(s) of Capacitor(s). Thus, the fundamental problem is to find an optimal combination of capacitor positions such that technical losses are minimized subject to voltage compliance constraints.

4. APPLICATION CONSIDERATIONS FOR SHUNT CAPACITORS

4.1 INTRODUCTION

Once the control objective of the Capacitor has been determined as discussed in 3.4 earlier, careful consideration must be given to other operational interventions that are intended to achieve similar objectives, e.g. If the Capacitor is installed (to achieve voltage support), it has to be coordinated with transformer tap-changer settings. Failure to do this may result in the two voltage support operational interventions working in contradiction to each other, i.e. if the Cap raises the voltage, the tap-changer may tap the voltage lower thereby causing unnecessary and undesirable operational conditions on the network that may adversely affect the quality of supply and system stability.

4.2 REACTANCE TO RESISTANCE (X/R) RATIO

The components (conductors, generators, transformers, lights, variable speed drives etc.) of a power system are made up of resistive (R) and reactive (X) network elements. Mainly, R and X are a function of distance and thus dependant on the length of conductors, spacing between conductors, type of cables and their size, spacing of conductors to ground, mutual coupling between conductors, etc. Technical losses are a function of R, whereas Voltage drop depends on X and R [22]. The reactance in AC systems causes a voltage drop and it is a function of conductor size, type and spacing between conductors i.e. the spacing between phases on the same circuit and other neighbouring circuits. The diagram below depicts the relationship between R and X on a traditional power triangle:

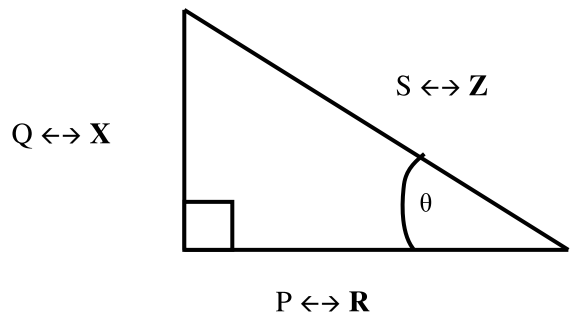


Figure 4.2.1: Depicts the relationship between X and R

From the diagram above, pf = cos (θ), tan (θ) = $\frac{X}{R}$ => θ = $\tan^{-1}(\frac{X}{R})$

Thus, pf = cos $\{\tan^{-1}(\frac{X}{R})\}$. To illustrate the relationship between the power factor and $\frac{X}{R}$, consider the following graphical illustration.

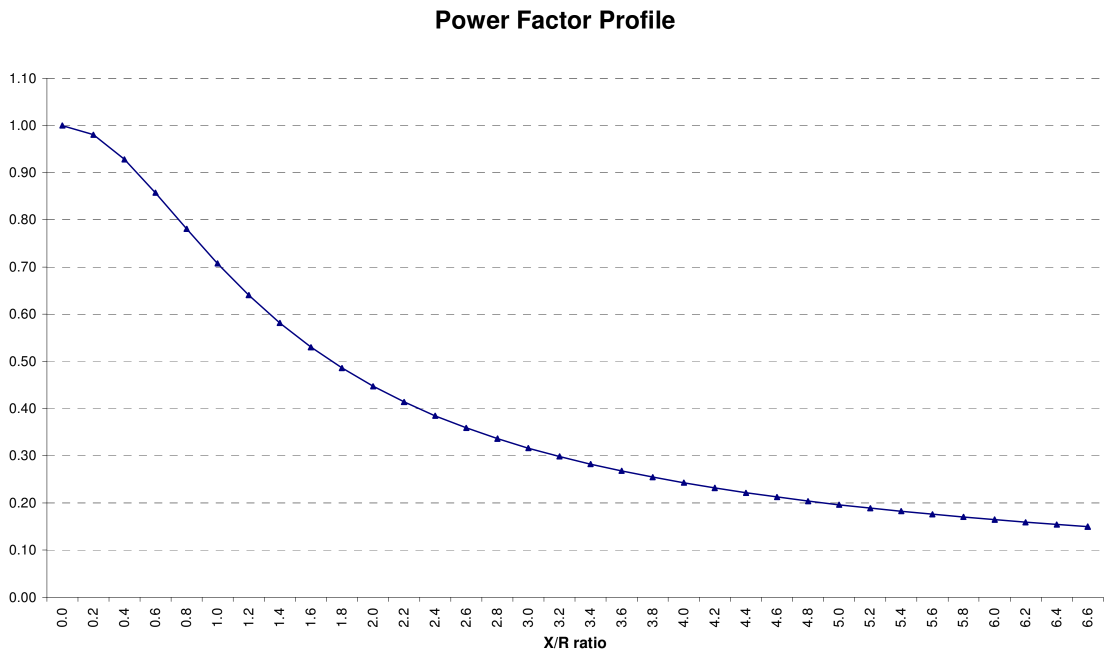


Figure 4.2.2: Depicts the relationship between pf *vs.* $\frac{X}{R}$

The reactance of a circuit is largely dependant on the spacing of the conductors and hence the larger the spacing the larger the reactance. The above figure indicates that the lower the network $\frac{X}{R}$ ratio, the higher the network inherent *pf* and if the $\frac{X}{R}$ ratio is large, the network properties are such that there will be a tendency to attenuate the overall system *pf*.

From the above illustrations, it can be deduced that networks that predominantly comprise of overhead line circuits generally have a high $\frac{X}{R}$ ratio. On the other hand, networks that predominantly comprise of underground cable circuits generally have a low $\frac{X}{R}$ ratio. A high $\frac{X}{R}$ ratio will result in a high voltage drop on the network and will also impact on the effectiveness of shunt compensation through a capacitor bank [21].

4.3 CHARACTERISTICS OF HV TRANSMISSION LINES

It is also of significance for an Operations engineer to consider the characteristics of HV lines when dealing with the optimisation problem relating to optimal placement of capacitors. Active power losses are dependant on Resistance (R) whereas Reactive power losses are dependant on Reactance (X). In HV networks X >> R and because X is quite high, it makes it difficult to transmit MVArs through transmission lines over long distances.

The inductive/ capacitive effect (characteristics) of the HV transmission lines can be classified into the following main categories [7]:

- **Low line loading**

- During low loading conditions, the capacitive effect of the line dominates and the line starts to generate MVArs. This must be borne in mind when deciding on the optimal placing of Capacitor banks as they maybe a leading *pf* produced during low loading conditions.

- **Surge impedance loading**

- A power loading condition whereby the capacitive effect of the line equals its inductive effect is known as Surge Impedance Loading (SIL)
  - SIL is the MW loading on a transmission line at which the line's natural reactive power production equals its reactive power consumption
  - Surge impedance $Z_c \sim \sqrt{\frac{L}{C}}$ & SIL (MW) $= \frac{|kV_l|^2}{\sqrt{L/C}}$, where *L* is the line's equivalent inductance and *C* is the equivalent line's Capacitance.

- **High line loading**

- During high loading conditions, the inductive effect of the line generally dominates and thus the line absorbs MVArs.

All the above loading conditions indicate that a given network, which is predominantly made of HV transmission lines, will have inherent reactive power absorption/ generation characteristics. This network reactive power absorption/ generation characteristics will depend on the loading conditions on that network, i.e. high, medium or low loading. These effects are more significant on very long lines and not so significant for short to medium length overhead networks.

4.4 QUALITY OF SUPPLY

When a capacitor is switched in or out of service, a surge in voltage and reactive power is caused. The maximum capacitor bank size to be connected on the network is limited by the maximum permissible voltage change during the switching of the capacitor banks [21]. The National Energy Regulator requirements for voltage flicker will need to be complied with as there are no specific voltage limits during switching surges.

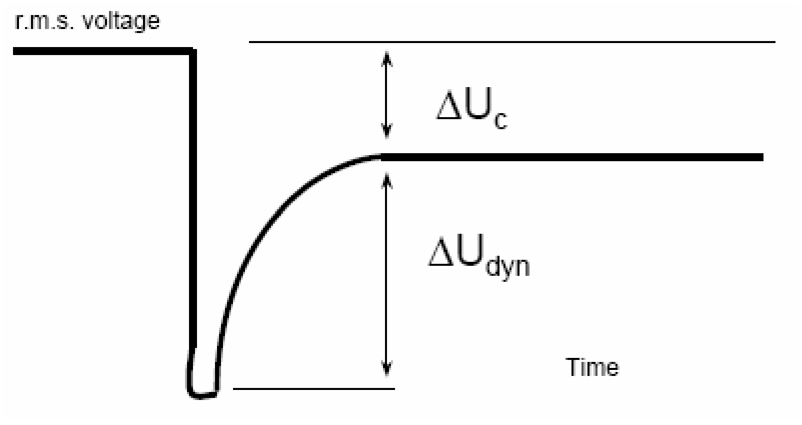


Figure 4.4: Illustrates a rapid voltage change (Figure adapted from [23])

ΔU_c is the steady state r.m.s voltage change and
ΔU_{dyn} is the dynamic r.m.s voltage change

A decrease in voltage is depicted. The voltage could also increase as a result of a rapid voltage change. It should be determined if there are any harmonics present on the network where the Capacitor is to be installed in order to avoid local harmonic resonance.
For switched capacitor operation i.e. when a capacitor is only switched on at the times when it is needed, the switching actions will result in transient voltage stresses that may compromise the quality of supply at sensitive load points, e.g. personal computers, television sets and other sensitive solid state electronic loads.

Susceptibility curves that have been determined in literature can be applied to assess the impact of these voltage stresses on sensitive loads and components [24]. Exact details on the expected amplitudes and waveforms of these transients would be required to do the necessary sensitive analysis.

4.5 SYSTEM STABILITY

Connecting shunt Capacitor(s) on the network increases the *pf* of the power system generators. This happens due to the reduced reactive power that the generators have to supply. The reduction in MVar output of the machine leads to a reduction in the excitation current of the machine which in turn reduces the output voltage of the machine [21]. Machines/ generators are often rated for a certain maximum output *pf* to ensure the security of the machine and sufficient reserve margins in case more MVArs are needed as different loading conditions dictate. Whilst some generators require *pf* <= 0.95 to ensure stability of the turbines, some of the generators can be operated between 0.95 and 1 *pf* without any stability problems [21]. The following figure shows a typical capability curve of a generator:

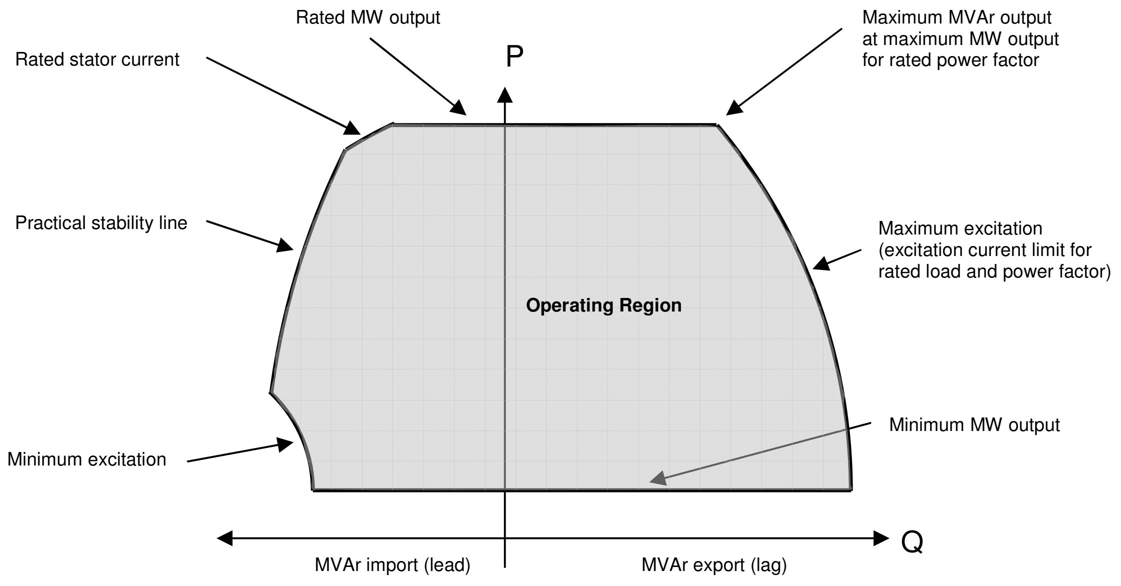


Figure 4.5: Generator capability curve (Adapted from [25])

The figure above indicates all the essential limits to be complied with on the source Generator to ensure maximum efficiency from the machine, required voltage output, required active and reactive power output and turbine stability and thereby ensuring overall system stability. It would then be advisable for the Operations Engineers to consider acquiring the reactive power capability curves for the source Generators from the manufacturer's specifications to ensure that the compensation provided by the Shunt capacitors does not result in any violations on the stipulated Generator power factor limits. This will then help optimal power factor limits to be complied with.

4.6 VOLTAGE REGULATION

While it is of critical importance to enhance the network's voltage profile, it is important to ensure that the network's voltage profile does not violate voltage limits during low loading conditions. For fixed Capacitors, it is therefore important that the size of the compensation Capacitor be such that during low load conditions, the Capacitor should not cause the *pf* to lead, which may even lead to over voltages. A network with long overhead lines is inherently capacitive and hence during low loading it generates VArs and if the Capacitor is also in service, it may cause the overall *pf* to lead. The Planning or Operations Engineer will need to make certain that the capacitor position and size are such that the voltage at all network busbars remains within limits for all loading levels. The voltage limits are as stipulated in NRS048 quality of supply policy document.

Installation of shunt capacitor banks also provide a minimised losses benefit on the sub-transmission network. The VArs are provided closer to the load centres and hence the technical losses that could have been incurred through the VArs transmitted through lines and transformers are averted. The technical losses cause heat dissipation in the conductors due to the inherent resistance of the conductors. This may lead to insulation failure and compromise the reliability of the system in the long run [26]. The improved voltage on the network will also reduce the load current, which will in turn lower technical losses.

If the control objective of a Capacitor is based on voltage i.e. the Capacitor is switched on based on the voltage level at a busbar, care must be taken to ensure that there is coordination between the transformer tap changer settings and the control of the Capacitor [21]. This is to ensure that there is no conflict between the two control mechanisms (capacitor control and transformer tap changer control) as this might lead to unnecessary switching surges and strain on the tap changer mechanism.

5. OVERVIEW OF EVOLUTIONARY ALGORITHMS

A. INTRODUCTION

Evolutionary algorithms work on the basis of Charles Darwin's natural biological evolution theories. Evolutionary Algorithms (EAs) are the probabilistic and direct search optimization algorithms which are inspired by the process of natural evolution [12]. Most of the EAs descend from three related, but independently developed approaches, i.e. Genetic Algorithms (GAs), Evolution Strategies (ESs), and Evolutionary Programming (EP). All these algorithms work on the basis of organic evolution models. The models represent a collective learning process within a population of individuals, where each individual represents a point in the search space of potential solutions. The starting population is initialized and then evolves towards a better solution in the search space by means of the processes of mutation[1], recombination[2] and selection[3].

The Evolutionary Algorithms generate an initial population of random solutions, which is evaluated by each individual's fitness. The solution is biased towards individuals that produce the best fitness values on the "survival of the fittest" principle. The principle works on the basis that the species that produce better individuals in an environment should be favoured, since the chances that a good child will be produced are high, if two good parents are mated in a population of trial solutions. This implies that only good parents must be kept in the population. This is achieved by considering a certain percentage (say T%) of the best individuals and then discarding the rest of the population {i.e. (100-T)% }. From the best percentage, parents are then allowed to mate in order to reproduce the rest of the population. The upper best percentage (T%) from the new population is considered, recombination and truncation of the population then takes place. Mutation is also introduced to avoid premature convergence of the solution, i.e. convergence to a local optimum (*cf.* 3.2.2). This carries on until the best individual that optimizes the chosen problem is found [17].

1. *"Introduces innovation into the population" [12]*
2. *"Allows for mixing of parental information while passing it to their descendants" [12]*
3. *"Favours individuals of higher quality to reproduce more often than worse individuals" [12]*

The following general evolutionary algorithm reflects the neo-Darwinian model of organic evolution; the algorithm was adapted from [13].

```
t := 0;
initialize P(t);
evaluate P(t);
while not terminate do
    P'(t) := variation [P(t)];
    Evaluate [P'(t)];
    P(t+1) := select [P'(t) U Q];
    t := t + 1;
end
```

The operation of the above general evolutionary algorithm can be described as follows: [11]

- A random initial population of solutions is generated; the population at time t is denoted as P(t) and consider the population to have M individuals.
- The generated population is then evaluated in terms of its fitness depending on whether the problem is that of minimization or maximization.
- An offspring population P'(t) is then generated by means of the variation of the recombination and mutation operators.
- Offspring values get evaluated by calculating the objective function values for each of the solutions represented by the offspring population.
- Selection based on fitness values is done to bias the solution towards better individuals. On the selection step, the offspring population is compared with the set of individuals Q that might be considered for selection.
- The process iterates until the best solution is found or until the maximum runs.

The above algorithm applies to all the three evolutionary approaches: GAs, Evolutionary Programming and Evolutionary Strategies, since they are all related with the main differences being the representation of solutions, mutation, recombination and selection operators or variation operators. This research is only constrained to one evolutionary approach i.e. genetic algorithms [17]

B. GENETIC ALGORITHMS

John Holland invented genetic algorithms with the main idea of reflecting the principles of natural evolution on a simulation computer algorithm. The reasoning behind this principle was that if nature is able to produce better performing individuals from a natural randomly created population, then it could be reflected on a computer algorithm that can be used to solve complex problems [14]. Since in the natural realm evolution works on the chromosomes, in a population the chromosomes that decode into more successful individuals tend to reproduce more often.

5.1 BENEFITS OF USING GENETIC ALGORITHMS [15]

- Genetic algorithms search from a selected set of designs and not from a single design, which increases their chances of finding a better solution.
- Genetic algorithms are not derivative based as they only use fitness values, hence they do not require additional information about the objective function, as a result they are insensitive to local optima points to which mathematical optimization techniques are sensitive.
- They are also insensitive to premature convergence because of the variation operators that they use to exploit the parameter search space, i.e. selection, recombination and mutation.
- Genetic algorithms can work with both discrete and continuous parameters, which make them applicable to almost all the engineering design problems. This makes it feasible to employ a GA-based solution for the "Optimal Capacitor Placement Problem", since the optimisation problem is discrete in nature.
- Genetic algorithms are fully parallelizable, i.e. the evaluation of chromosomes are performed independently from one another, since the generated population of solutions is a purely random process there is no dependence between generations of solutions.
- The above characteristics render genetic algorithms robust as they are able to adapt chromosomes to changes in the environment the way their biological counterparts do.

5.2 DRAWBACKS OF USING GENETIC ALGORITHMS

- One major disadvantage of genetic algorithms is the computational cost of the large number of runs of the design code needed to evaluate the fitness of a set of designs for each generation [15].
- Computation time taken by GAs is often relatively long.

5.3 GENETIC ALGORITHMS IN DESIGN OPTIMISATION

In design optimization with genetic algorithms, there are two fundamental tools that are used to locate an optimum result, viz. the search tool and the analysis tool. The search tool drives the genetic algorithm to explore every point in the search space as every point represents a different design. For every chromosome that gets decoded to a point in the search space, the GA calls on the analysis tool to evaluate the performance of that particular point [14]. Essentially the role of the analysis tool is to solve the optimization equations and return the corresponding parameters to the search tool.

There are four major steps followed in preparation to solve a problem using GAs [14]

- **Determination of the solution representation scheme**

  This is the way in which design variables are coded, whether into finite length strings binary strings or real valued strings. It is important to have a representation scheme, since it is responsible for mapping every point in the search space into a unique chromosome.
- **Determination of the fitness measure**

  This is a measure of how successful and good an individual is in some predefined environment, an individual being a decoded chromosome. In design optimization, the environment is the design objective function, so the fitness measure would be the evaluated value of this objective function. The evaluated value would then serve as the measure or indication of whether the individual is successful or not. The fitness measure would of course depend on whether it is a minimization or maximization problem.
- **Determination of the stopping rule**

  It is crucial to specify some rule that tells the algorithm when to stop, either by fixing the number of generations and use the best individual at the end of the iterations as the optimum result, fix the time elapsed and similarly choose the best individual as the optimum result or alternatively let the population converge to some fitness region with a certain pre-defined error margin.

- **Determination of the parameters of the operation of GAs**

  The parameters can be classified into primary and secondary parameters. The primary parameters are:

M	represents the population size, which is the number of chromosomes in the population.
L	represents the chromosome length, which is the number of genes used to form one chromosome.

  The secondary parameters are:

p_c	representing the crossover probability.
p_m	representing the mutation probability.

5.4 A GENETIC ALGORITHM

The pseudo code for the implementation of a genetic algorithm is presented as follows [11]:

```
Begin, t = 0;
   initialize P(t);
   evaluate structure in P(t);
   while termination condition is not satisfied, then
   begin
         t := t + 1;
         select for reproduction, C(t) from P(t-1);
         recombine structures in C(t) forming C'(t);
         evaluate structures in C'(t);
         apply mutation
         select for survival, P(t) from C'(t) and P(t-1);
   end
end
```

The above pseudo code can be described to operate as follows:

- Randomly generate an initial random population of M chromosomes as defined in section 5.3 denoted by population P(t) in the pseudo code.
- Evaluate each generated chromosome and assign it some fitness measure.
- Select a child population denoted by C(t) from the parent population for reproduction.
- The individuals from the selected child population C(t) are then mated by the crossover operator to form a new offspring population C'(t).
- Evaluate the new individuals in C'(t) for the fitness measure on the objective function of the optimization problem.

- Apply mutation to preserve diversity in the newly generated population in order to avoid premature convergence.
- If the stopping condition is reached, stop the algorithm; or go back to the second step.

5.4.1 SOLUTION REPRESENTATION

The solution or chromosome is traditionally represented as a binary vector from which decision variables are represented or encoded as real values. This is illustrated in the figure below:

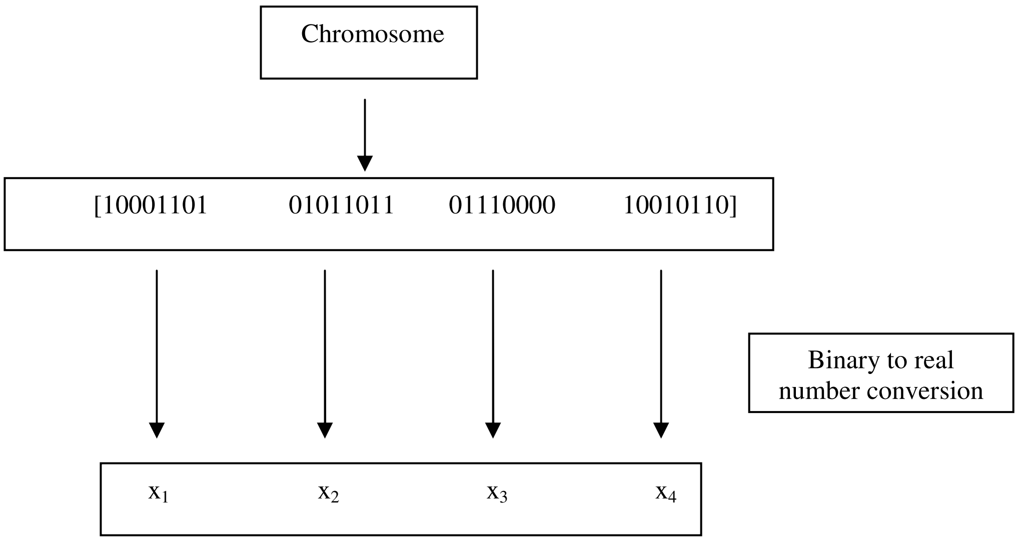


Figure 5.4.1: Shows the binary representation of a chromosome in a GA

The number of bits or genes is determined by the algorithm's precision which is specified before the algorithm is run. The precision depends on the solution accuracy that is desired. The above figure shows encoded decision variables from a bit string known as a chromosome. These encoded variables are then used to evaluate the fitness of the individuals on the objective function.

5.4.2 THE CROSSOVER OPERATOR

Once reproduction is finished, crossover takes place in order for the chromosomes to exchange information [14]. The crossover operator simulates the process of natural evolution. It allows two parents to mate in order to produce an offspring. Crossover occurs at a rate that is determined by probability p_c (*cf.* 5.3)
The crossover process, which is essentially the information exchange between chromosomes, is carried out by swapping bit strings of the two parents' chromosomes at a chosen bit position or bit site. Either uniform one-point crossover or two-point crossover can be employed when performing crossover.

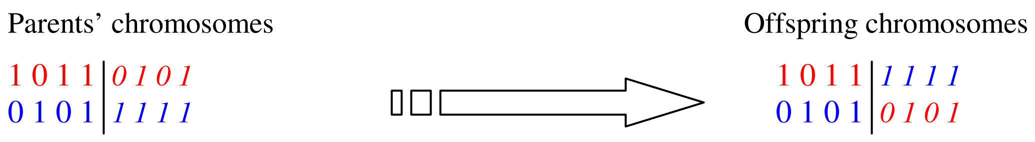


Figure 5.4.2(i): Illustration of uniform one-point crossover

The above figure shows a uniform one-point crossover with a crossover site of four and the newly reproduced offspring. To the left of Figure 5.4.2(i), the top chromosomes’ latter four bits from the parents’ chromosome are mated with the bottom chromosomes’ latter four bits from the second parents’ chromosome and the resulting offspring is depicted on the far right of Figure 5.4.2(i).

Secondly, consider a uniform two-point crossover as described below:

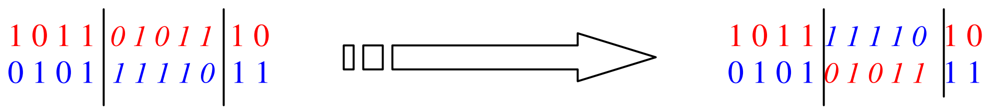


Figure 5.4.2(ii): Illustration of uniform one-point crossover

The uniform two-point crossover with a crossover site of four and nine is shown in Figure 5.4.2(ii) above. This only swaps the bit strings that are constrained or bounded by the crossover sites and the bits outside these crossover points are left unchanged. Another variation that uses multiple point crossover, where bits positioned between randomly selected sites are swapped, is also available.

5.4.3 THE MUTATION OPERATOR

The mutation operator ensures that there is diversity in the population to avoid any bias that could lead to premature convergence. The rate at which mutation is performed is determined by the mutation probability p_m (*cf.* 5.3). The mutation operator also prevents against loss of some important genetic information [14]. With the consideration of genetic algorithms in engineering optimization, it then leads to reviewing the operation of the selected set of evolutionary genetic-based methods.

5.5 THE SELECTED EVOLUTIONARY METHODS

5.5.1 THE BREEDER GENETIC ALGORITHM (BGA)

The optimization problem is generally faced with finding a set of numbers that optimizes some multivariate function. Professor Muhlenbein did sterling work on developing the Breeder Genetic Algorithm based, not on natural "Darwinian" evolution, but on artificial selection as practiced by animal breeders [16]. When two parents from a fitting environment mate together to produce an offspring that is also fitting to the environment, this is known as animal breeding.

5.5.1(a) OPERATION OF THE BGA

A population of solutions is initially generated randomly in a uniform manner. Trial solutions are real vector valued numbers. Each solution gets evaluated and truncation selection is used to truncate the population and preserve only the fit individuals. This is illustrated in the figure below [17]:

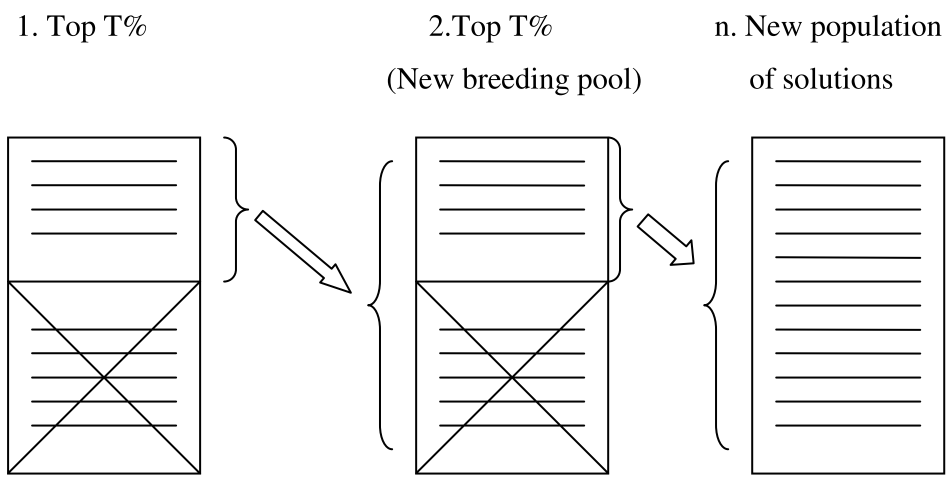


Figure 5.5.1(a): Illustration of truncation selection operator in a BGA

Depicted above is how the truncation selection is made from an initial population of trial solutions. In the first step, top T% chromosomes are selected, which are the fittest in the population and the rest of the unfit chromosomes are discarded. The top T% chromosomes form a new breeding pool of survivors. In the second step, the T% chromosome parents are allowed to mate in order to recombine and refill the whole population. After recombination, truncation takes place again to preserve the best performing individuals and discard the rest. Mutation, which will be discussed in detail later, is also performed to maintain population diversity. The above takes place until a satisfactory solution is found, i.e. the optimum solution.

There is no general stopping criterion; the algorithm is allowed to continue until the results are acceptable or until the allocated time expires [16].

5.5.1(b) THE RECOMBINATION OPERATOR IN A BGA

Once the generated population of solutions is truncated to select the best performing individuals, these better individuals need to be mated together by the recombination operator in order to reconstruct the rest of the population by breeding better offspring.
To create a new child from the survived parents, two parents are randomly selected from the breeding pool and elements from each parent are selected to breed a new offspring. In genetic algorithm terms recombination is also termed crossover. Various recombination methods have been developed in literature, three of which will be considered in this chapter i.e. **Uniform, Line** and **Volume Crossover** [16].

- **Uniform Crossover** – allows the algorithm to select each element of a child from either parents at random (determined by a tossing a coin)
- **Line Recombination –** allows for the algorithm to generate a random number ($0 < r < 1$) that determines the position of the child anywhere along the line that joins the two parents. If parent 1 is located at position x_i and parent 2 at y_i, the child's location is given by $z_i = r_i.x_i + (1- r_i).y_i$
- **Volume Crossover** – This is a multi-dimensional extension of the line crossover operator, in that a random vector r is generated and the child's location is given by the expression $z_i = r_i.x_i + (1- r_i).y_i$

In order to ensure that the search space is properly explored and that every point in the search space is accessible, the line crossover operator is modified to the **Extended Line Crossover** operator. With this operator the child is not only restricted to be between the two parents' locations. However, the line is projected about 25% beyond the end points [16], thus the tendency of the children to cluster is reduced.

5.5.1(c) THE MUTATION OPERATOR IN A BGA

Performing the steps described above has a great probability of premature convergence, which means that the solution may not be a global optimum but one that is local. To overcome this predicament, some randomness known as mutation needs to be introduced to the solutions in order to preserve population diversity. This can be achieved by adding a small normally distributed random number with standard deviation (R). Professor Greene [16] has developed a technique of manipulating the value of R. If R is too little we get premature convergence and if it's too much we disrupt the search space and fail the algorithm to converge. Thus, Professor Greene proposed that R be set to some nominal value R_{nom} (say 0.1 to begin with).

Divide the population into two equal parts A and B, then to A mutation is applied at twice the nominal rate ($2.R_{nom}$) and to B at half the nominal rate ($R_{nom}/2$). Now the mutation rate is adjusted in favour of whatever strategy seems to win, i.e. if A generates solutions that are fitter than those that are generated by B, then increase the mutation rate R, by, perhaps 10%. Conversely, if B wins, then the mutation rate R is decreased by a similar amount of 10%. In this way the mutation rate tracks the optimal one throughout the search.

5.5.2 POPULATION-BASED INCREMENTAL LEARNING

PBIL is a stochastic non-linear programming technique that has many advantages over other stochastic programming techniques [18]. It is an abstraction of a genetic algorithm and is able to maintain the statistics contained in a GA's population.
PBIL does not have the crossover operator that other conventional genetic based algorithms have. However, it is a combination of genetic algorithms and competitive learning. Another version of PBIL is parallel population-based incremental learning (PPBIL), which operates in a similar way to PBIL, except that it uses two sub-probability vectors to construct the main probability vector from which trial solutions are sampled. PPBIL constructs the main probability vector by selecting bits from the two sub-probability vectors.

5.5.2(a) SOLUTION REPRESENTATION IN A PBIL ALGORITHM

As in a genetic algorithm, the solution gets encoded into a finite binary vector of some fixed length. PBIL generates a probability vector from which samples of a trial solution are drawn to produce the next generation's population. This probability vector serves as a prototype for high evaluation vectors for the function space being explored [19]. Initially the probability vector is assigned equal probabilities equal to 0.5. A number of trial solutions are generated by sampling a random vector with the probability vector.

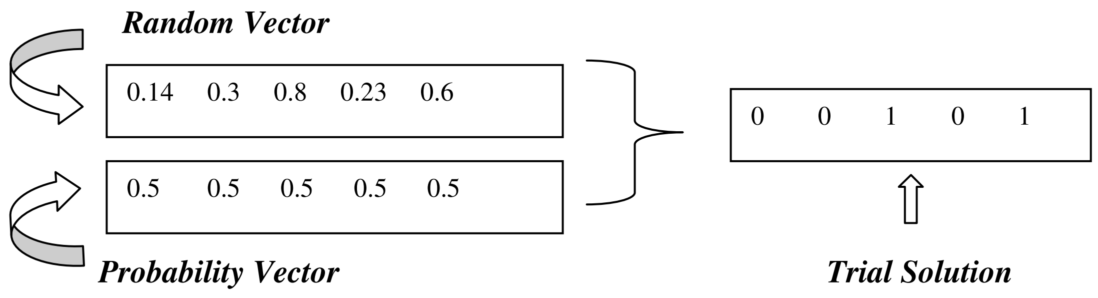


Figure 5.5.2(a): Illustration of how a PBIL trial solution is generated

A trial solution is generated by comparing element by element of the randomly-generated vector and the probability vector. If an element of the randomly-generated vector is less than the probability of 0.5, then the generated bit is 0 or else the generated bit is 1, as shown in Figure 5.5.2(a).

The probability vector is pushed towards the generated solution vector, i.e. the position where bit 1 is generated in the trial solution vector is favoured by increasing the probability of the corresponding position in the probability vector. Conversely, the probabilities where a zero bit appears in the probability vector are reduced. This is done to drive the probability vector towards favouring the trial solution. As a result of varying the probability vector in favour of the trial solution, upon convergence it might look like this: [0.01 0.01 0.99 0.01 0.99 0.99] for a trial solution of [0 0 1 0 1 1].

5.5.2(b) DISTINGUISHING FEATURES OF PBIL

The main distinguishing features of the PBIL algorithm are the **probability vector** [*cf.* 5.5.2(a)], the **learning rate** and the **forgetting factor**.

The **learning rate** is the amount by which the probability vector is being changed after each cycle [18]. The effect of the learning rate affects the probability vector as described by the following relationship: $P_i = [P_i * (1 - LR)] + (LR*s_i)$

Where P_i is the probability of generating a one in bit position i

s_i is the ith position in the solution vector towards which the probability vector is moved

LR is the learning rate.

The **forgetting factor** combats lack of diversity in the algorithm in order to avoid premature convergence to a local optimum solution. The main advantage of PBIL over the other genetic algorithmic approaches is that it is characterized by fewer parameters, so, as a result, very little specific knowledge about the problem to be optimized or solved is required.

5.6 DIGSILENT IMPLEMENTATION OF GA's

5.6.1 CONVERSION OF CODES TO DPL SCRIPTS

The algorithms described above (AMBA, PBIL and PPBIL) were coded in DigSilent using the DigSilent Programming Language (DPL). The original implementation of these GA codes was in Matlab format and since the DPL syntax format (similar to C++ syntax) is different to Matlab syntax, the conversion was therefore necessary. The conversion of the codes took the following key coding aspects into consideration:

- **Variable definition/ initialisation**
  - In Matlab, a variable type is not specified, e.g. an integer, a double or a string.
  - In DPL, variable types have to be specified accordingly in order for the code to distinguish different types of variables.

- **Arrays/ Vectors**
  - In Matlab, an array/ vector is easily defined by the expression (array name) = [ ], which has a very large / infinite size.
  - In DPL, an array/ vector is defined by a physical vector or matrix object which has to be created in the DPL folder and given finite dimension (row and column sizes).

- **Objective function representation**
  - In Matlab, the objective function is normally represented as a theoretical mathematical function with many variables to optimise (either to minimise or maximise), generally with local and global optimal points as discussed in Chapter 3, section 3.2 earlier.
  - In DPL, the objective function is comprised of real variables calculated from simulated loadflow conditions, e.g. technical losses as a function of conductor copper losses, transformer copper losses and transformer iron losses. The DPL objective function is based on the network model in DigSilent.

- **Solution evaluation**
  - In Matlab, a trial solution is simply evaluated by computing the objective function using the randomly-generated trial solutions
  - In DPL, a trial solution is evaluated by running a real loadflow with network conditions that mimic the assumed trial solution, e.g. a certain combination of capacitor solution.

- **Sorting of best fitness solutions**
  - In Matlab, best fitness solutions are very easily sorted through a sort function regardless of the array or matrix size.
  - In DPL, the sorting must be done (in nested if and for loops) as the actual solutions need to be shuffled in the arrays or matrices. This takes some time and uses up computational resources.

5.6.2 KEY FEATURES DEVELOPED ON DIGSILENT

The following key features, to make the implementation of the code conversion from Matlab to DPL scripts successful, were developed:

- **Candidate buses specification**
  - This is a feature to specify the buses to be considered for placement of the Capacitor(s). The network busbars (Candidate buses) to be considered for placing the Capacitors are represented by a string of binary numbers, i.e. 1 means a Connected Capacitor is on the busbar and in service (supplying/ injecting the specified MVArs into the network) and 0 means the Capacitor is out of service. The buses are therefore coded as a bus string as illustrated below for a network with 6 HV busbars:

    [1 2 3 4 5 6] ------- Bus index

    [1 0 1 1 0 0] ------- Bit representation for a trial solution

    The above bit representation string signifies that Capacitors at bus 1, bus 3 and bus 4 would be in service whereas, at bus 2, bus 5 and 6, the Capacitors would be out of service. The code would then calculate losses for this trial solution and store the value. The search would then be guided to ultimately locate the best optimal trial solution that would yield minimum losses on the network.

- **Capacitor size specification**
  - Once the minimum reactive power requirements of a network are determined, this feature allows one to specify a Capacitor bank size in MVArs. This is the Capacitor size that will be used at all candidate buses as mentioned above. The Cap bank size is specified once and the algorithm will then automatically update all candidate capacitors.

- **Existing solution detection**
  - Since the trial solutions are generated randomly and automatically, this feature has been added to determine whether a solution has been previously generated or not in a given pool of generations. The rationale here is that once a trial solution is generated, there is no need to re-evaluate the objective function for the same trial solution. In a theoretical scenario with a theoretical objective function, it may not make a big difference to ignore this feature; however, for a practical objective function, e.g. optimising technical losses on a real network, this involves running a loadflow over a 24-hour loading period and may take up a lot of computation time.

- **Constraints violation detection**
  - This feature determines whether a trial solution leads to violation of a predetermined constraint, e.g. HV operating voltage must meet the criteria ($95\% < V < 105\%$) as per NERSA's requirements. In this research, only the voltage constraints were considered, however, if desired, *pf* constraints, reactive power flow constraints, line loading constraints (to avoid a voltage collapse or for thermal limitations), etc. may also be defined and considered.

- **Objective function modelling**
  - The technical loss for lines (Cu losses) and transformers (Cu and Fe losses) on a given network is used as the objective function on the DPL script. This is directly evaluated from the simulated DigSilent loadflow results for different network loading conditions.

- **Energy and demand optimisation considerations**
  - For every trial solution, the DPL code evaluates the objective function over a defined period in a given season. A typical period in this case is 24 hours and typical seasons are winter and summer. As mentioned earlier the violation check is done for the 24-hour period, hence a solution must be optimal for the entire duration. This helps because often, reactive power demand may be adequately compensated over the peak period, but at off peak periods there may be over-compensation which may lead to leading pf and possibly over-voltages.
  - Since the code allows the Engineer to specify the start and end times of the period, this feature may also be used for reactive power compensation over a preferred loading period, e.g. during peak periods.
  - This feature therefore optimises a network's reactive power requirements, not only for demand during a specific loading period, but also considers the demand over the entire specified duration (which we may also call energy).

- **Best fitness**

- This feature keeps track of the best fitness as the code tries to locate the most optimal solution in the search space. As each trial solution is evaluated, a comparison between the solution and previously evaluated solutions is made to determine a “best so far” solution. At the end of the algorithm, the overall best solution is then determined.

- **Mutation Operator**

- This feature helps with adapting the search algorithm to avoid being trapped on a local optimum solution and provides the algorithm with the necessary manoeuvrability and guidance or “search biasness” towards finding the global optimum solution. Modifications to the Matlab code were done to cater for the discrete nature of the optimisation problem being investigated.

5.6.3 BENEFITS WITH DIGSILENT IMPLEMENTATION

The main benefits with implementing this solution on DigSilent are as follows:

- The Transmission/ Sub transmission networks have already been modelled on DigSilent and thus it makes it feasible and practical for Operations and Planning Engineers to implement this solution on DigSilent.
- The network data required for implementation of the proposed solution is officially kept on DigSilent; hence for this research no extra network modelling was needed. Although incomplete, some of the load type data exists on DigSilent.
- Due to the programming object-oriented nature of DPL scripts, it made it possible to implement the solution, albeit the conversion of some of the commands had its challenges, e.g. sorting of vectors, matrices etc.

6. IMPLEMENTATION OF OPTIMISATION TECHNIQUES

6.1 INTRODUCTION

In order to test for the robustness of the genetic search algorithms that have been described earlier, a criteria needs to be considered. This is to ensure that the algorithms are thoroughly evaluated based on the merit of their performance to locate a globally-optimum optimisation solution.

6.1.1 THE CONVERGENCE AND RELIABILITY TEST

The convergence and reliability test tests the reliability of an algorithm to locate a true or global optimum point of a function to be optimized. The test is implemented by running 15 independent runs of each algorithm (per season, per network) on the function to be optimized and then recording the best result of each run. The different algorithms are then comparatively evaluated by comparing the standard deviation of each algorithm's 15 best solution's distribution. In order to ensure the robustness of the test results, the variability of the best results (out of the 15 independent runs) are analysed.
This also ensures that the genetic algorithms are consistent in finding the near or true optimum point of the function to be optimized. The evolutionary algorithms are ranked in accordance with the variability or standard deviation of the best results i.e. the algorithm that yields the smallest standard deviation in its results will be ranked the best and similarly the one that has the worst or largest standard deviation will receive the least or worst ranking. Therefore the results of the best ranked algorithm will be considered as reliable and consistent [17].

6.1.2 PARAMETER SETTINGS FOR ALGORITHMS

The following genetic algorithm parameter settings have been applied to the algorithms.

BGA settings	**PBIL** settings	**PPBIL** settings
Population size: 50	Population size: 50	Parallel populations: 10
Truncation selection threshold: 15%	Learning rate: 0.005	Learning rate: 0.1
Initial Adaptive mutation rate: 0.1	Forgetting factor: 0	Forgetting factor: 0

Figure 6.1.2: GA parameter settings

6.2 NETWORKS ON WHICH THE MODELS ARE TESTED

The optimisation algorithms were applied on four different networks. The networks were chosen mainly based on the different load mix or customer base fed from the respective network. The rationale is to test the algorithms on networks feeding different customers with different reactive power requirements. For example, domestic reactive power requirements for a residential customer base will be different to industrial reactive power requirements for an industrial customer base (i.e. mining, agricultural, commercial). The following table outlines the different networks chosen together with their characteristics.

Table 6.2: Four different pilot networks considered

Eskom Distribution Network	Primary Voltage [kV]	Main Load Type Mix	Geographical Loads Location	Province
Mercury/ Jersey	88	Mining, Traction, Industrial	Orkney	North West
Watershed/ Mmabatho	88	Residential, Commercial	Mmabatho & Mafikeng	North West
Mercury/ Goat DS	88	Commercial, Rural, Mining	Klipfontein, Wolmaranstad, Leeufontein, Mimosa	North West
Midas/ Lama DS	44	Mining, Industrial	Around Carletonville	North West

The table above indicates the different networks considered for the implementation of the optimisation genetic algorithms. The primary supply voltage is indicated. This does not show the secondary voltage; however, in many cases, the network design will be such that there will be transformers at the various substations stepping down the voltage to medium voltage level for reticulation or bulk feeding to the customers. Some of the large customers do take supply at a high voltage (i.e. 88, 66, 44, etc.) An example of these customers includes big municipalities (e.g. Johannesburg’s City Power, Vereeniging’s Emfuleni), furnace loads and mining houses.

This is where it becomes imperative to ensure that the reactive demand requirements of these networks are thoroughly optimised as the cumulative impact of unnecessary energy losses is in the order of millions of rands annually. This is due to the intensive energy usage of these large customers. The voltages at which power is supplied to these customers should also remain within the stipulated contracted limits to ensure that the efficiency of their processes is not placed in jeopardy. The geographical locations of the load distributions are also indicated.

6.3 LOADING DATA IMPLEMENTATION

When modelling the load characteristics of different load types supplied by the networks described in 6.2 above, it is very important to ensure that their load variation patterns are carefully considered for different loading conditions. This is mainly to ensure that the reactive compensation will be consistently optimal throughout different network loading conditions and network configurations. This led to developing the load profiles for the different loads mentioned above. Loading data for the recent previous years was considered for the main stations feeding the networks under consideration. This took into account different seasons (e.g. winter and summer). This was to mainly identify the characteristic load profile for the different stations per season. The seasons were defined as follows:

Table 6.3.1: Depiction of seasons

Summer							Winter					Summer		
2007 - S			2008 – S				2008 - W					2008 - S		
Oct	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec

The table above indicates how the seasons were defined, i.e. October of the preceding year to March of the current year is summer, and winter is from May to September. Consider the following table:

Table 6.3.2: Depiction of seasonal loadings of main distribution stations

Eskom Distribution Network	Main Load Type Mix	Winter Peak (MVA)	Winter Peak dd-mm-yy,hh:mm	Summer Peak (MVA)	Summer Peak dd-mm-yy,hh:mm
Mercury/ Jersey 88kV	Mining, Traction, Industrial	58	18 Jun'08,10:45	53	11 April'08, 12:25
Watershed/ Mmabatho 88kV	Residential, Commercial	118	10 Jul'08,18:45	100	19 Feb'08,19:45
Mercury/ Goat DS 88kV	Commercial, Rural, Mining	44	04 Jul'08,19:11	42	13 Nov'08,20:07
Midas/ Lama DS 44kV	Mining, Industrial	90	27 Jun'08, 12:00	96	17 Jan'08, 13:00

The main substation feeding the networks with different load types are depicted once again on Table 6.3.2 above with the seasonal loadings per main station. The idea is to base the development of the load profile on the day the main substation reached its respective seasonal peak demand. This is why the peak dates and times are also noted.

Consider the following network (Midas/ Lama 44kV Network described in Table 6.3.2)

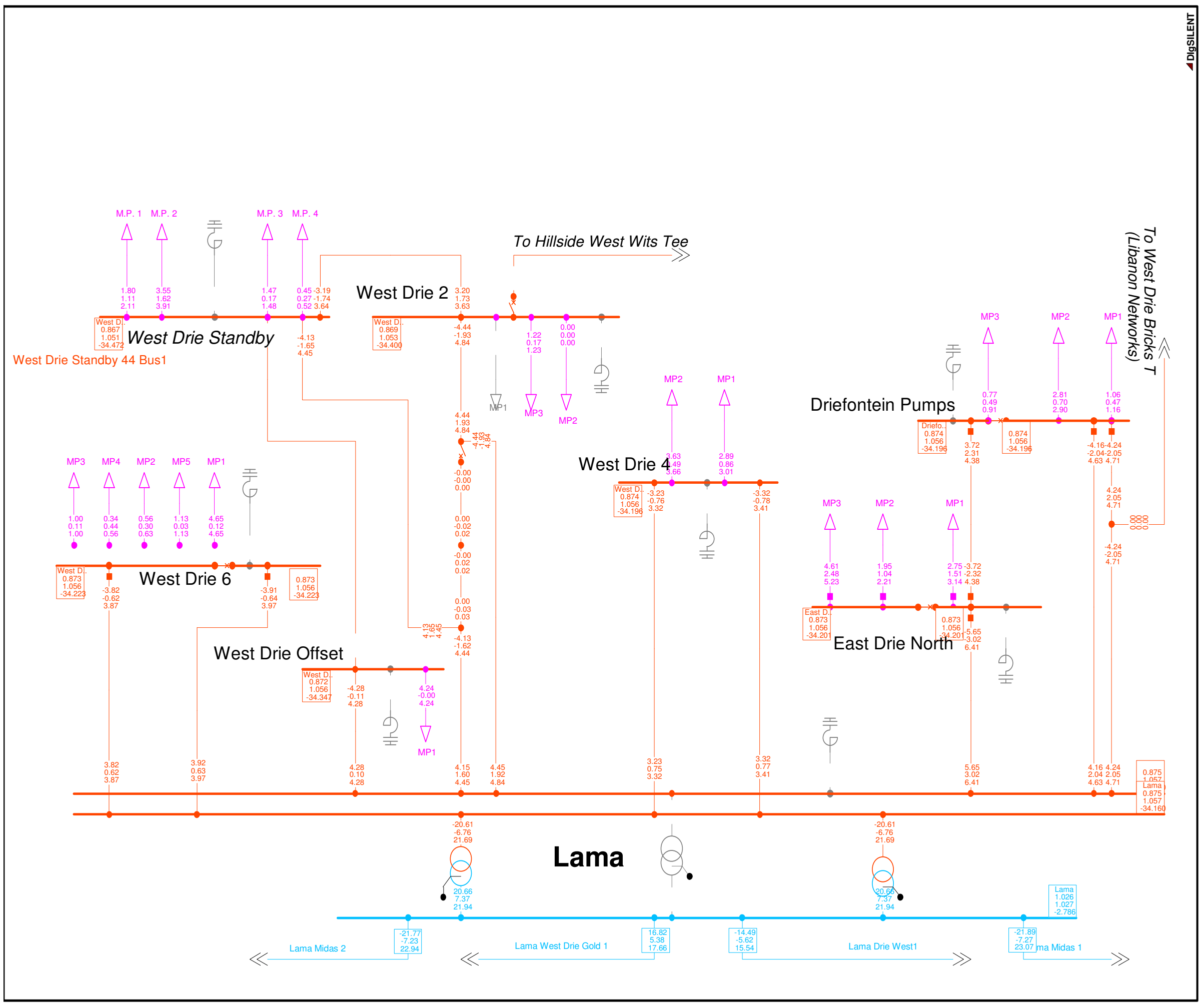


Figure 6.3: Midas/ Lama 44kV network simplified single line diagram

The diagram above depicts the Midas / Lama 44kV network simplified diagram. The blue lines indicate the 132kV power import lines into Lama Station. The 44kV lines are represented as red. The loads (supplied at 6.6kV in this case) from the various substations are represented by the purple arrow symbols on the diagram above. Please note: Where the loads are shown at the various substations, there are 44/6.6 kV transformers that are not shown on the diagram for simplification purposes. Interconnecting lines to other neighbouring networks are also shown. This is for back-feeding the Lama load during periods of emergencies i.e. when supply to Lama Station is constrained, e.g. if during peak loading conditions, one transformer at Lama is forced out of service and the other remaining transformer is unable to handle the total load. Some of the load on the Lama network will therefore need to be swung away from Lama towards the interconnecting networks.

6.3.1 DEVELOPMENT OF LOAD PROFILES

As mentioned above, seasonality played a key role in determining the characteristic load profiles for the loads considered in the simulations carried out in this research. The load profiles are developed for the active power demand and reactive power demand. This is aimed at mimicking the exact pattern the load follows as the two profiles (active and reactive power) are not always exactly the same. To illustrate this, consider the following figure:

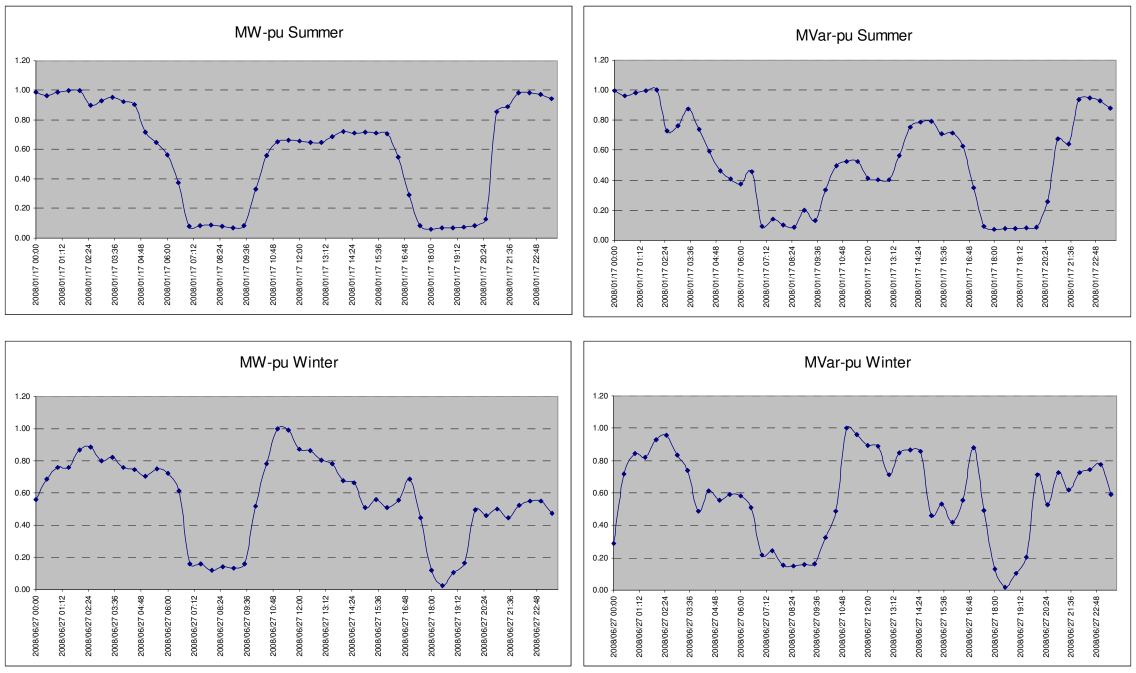
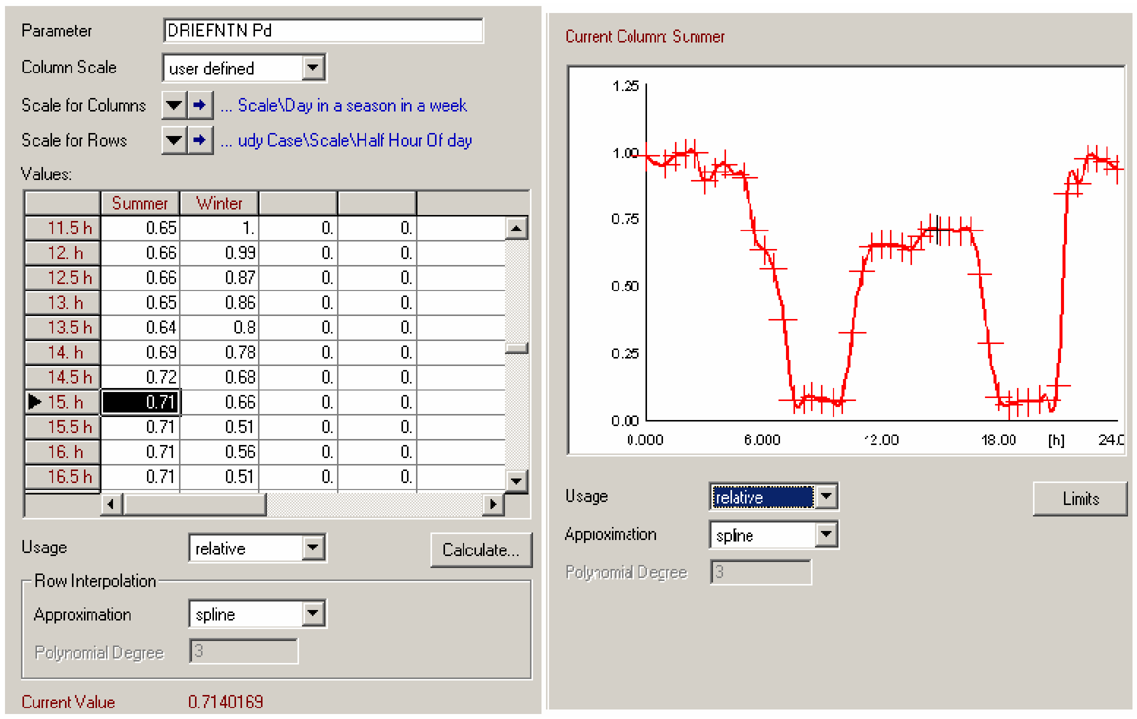


Figure 6.3.1: Driefontein Pumps load profiles

The Figure above depicts the load profile for Driefontein Pumps substation. The profiles are per unitised values. The two left profiles (top and bottom) on the figure represent daily typical Active power (MW) profiles for both summer and winter seasons. Similarly, the two profiles to the right of the figure represent daily typical Reactive power (MVAr) profiles for both summer and winter seasons. It can be seen when comparing the seasonal variations that the profiles are not exactly identical, but there is correlation. In this case, the load supplied is a mining industry and it appears that their usage patterns for winter and summer are generally similar. There is an incentive for them to use more outside the peak periods due to their tariff structure, which is punitive during traditional peak times. Due to these seasonal profile differences, even if in some cases they may be minor, it is critical to model them as such in order to closely simulate this load behaviour when solving the optimal capacitor placement optimisation problem.

6.3.2 ADAPTATION TO POWERFACTORY FOR IMPLEMENTATION

Per unitised load profiles and time scales are created and captured into Powerfactory for implementation. This is to enable a time-dependant simulation to be carried out instead of only simulating a “snapshot” peak condition. Consider the following depiction:



Parameter	DRIEFNTN Pd
Column Scale	user defined
Scale for Columns	... Scale\Day in a season in a week
Scale for Rows	... udy Case\Scale\Half Hour Of day

Values:

	Summer	Winter		
11.5 h	0.65	1.	0.	0.
12. h	0.66	0.99	0.	0.
12.5 h	0.66	0.87	0.	0.
13. h	0.65	0.86	0.	0.
13.5 h	0.64	0.8	0.	0.
14. h	0.69	0.78	0.	0.
14.5 h	0.72	0.68	0.	0.
▶ 15. h	0.71	0.66	0.	0.
15.5 h	0.71	0.51	0.	0.
16. h	0.71	0.56	0.	0.
16.5 h	0.71	0.51	0.	0.

Usage: relative

Row Interpolation
Approximation: spline
Polynomial Degree: 3

Current Value 0.7140169

Current Column: Summer

Usage: relative
Approximation: spline
Polynomial Degree: 3

Figure 6.3.2: Shows Driefontein Pumps Active power profile in Powerfactory

The picture above shows how the load profiles in Figure 6.3.1 above have been captured in Powerfactory to allow for simulations to be carried out for a varying seasonal profile. The diagram also shows the time over which the load profile is created. This time will be used by the time triggers in Powerfactory to determine which value to assign to the load when a load flow is simulated. The active and reactive power load profiles have been created (in Powerfactory) for all the loads supplied by the networks mentioned in Table 6.2.

These profiles are then linked to their respective loads in order to ensure that when a load flow is calculated, it takes into account the loading level as defined by the profile. This ensures that the load flow results are accurate and closely mimic the actual network loading at the different time periods during the day. This will also give an accurate representation of the network voltage profile and the technical losses profile of the various networks under consideration. Therefore the optimisation results that are determined will also be fairly accurate.

6.4 RESULTS PRESENTATION AND ANALYSIS

On the above said networks, the necessary parameters (loading data profiles capturing, checking that source voltages to the networks being studied are accurate by comparing the simulations to real measurements, etc.) were prepared in order to implement the Genetic Algorithm programmes. These GAs were first converted from raw Matlab codes into the DigSilent Programming Language (DPL) codes. The main challenge with the conversion is that to implement a simple Matlab command on DigSilent may be a bit more involved. This is because DigSilent programming language is a more object-oriented language.

The idea is that the GA code must be in a format that DigSilent can interpret and understand. This will make implementation of these GA codes easy to implement on existing networks as the Engineer would just need to copy the script into their DigSilent model, configure few parameters and implement. The DPL code would then be running on real modelled networks and take as its input parameters already modelled on DigSilent. The proposed methodology in section 6.5 below will outline a recommended approach that Engineers (Operations and Planning) can follow as a guideline to solving this optimisation problem.

This chapter outlines the results of each Genetic Algorithm applied on the four different pilot networks as stated in Table 6.2 above. The results are tabled for both summer and winter season's typical days. For every network, the results for the exhaustive method are first presented per season. The exhaustive method searches for the solution in the traditional way without any guidance whatsoever. The GA results are then also presented. This would indicate to the network Engineer the most optimal compensation solution per season. This would also indicate whether the optimal solution for one season would necessarily be optimal for the other season or not. The network engineer may also need to develop a "switchable" compensation solution which needs a Capacitor bank or banks to be in service during a certain period of time/ season, as the compensation may only be needed during a particular period of the year. The methodology developed in this research aims to show that this kind of optimisation technical solution can also be formulated.

6.4.1 GA CODES APPLIED ON GOAT DS NETWORK

The following tables indicate the Goat DS GA results for both typical summer and winter peak days. All the Algorithms located the most optimal solutions as follows:

Table 6.4.1: Optimal capacitor locations for Goat DS

SUMMER **OPTIMAL BUSES**	WINTER **OPTIMAL BUS**
Klipfontein Rural 88 Bus1b	Klipfontein Rural 88 Bus1b
Mimosa Rural 88 Bus1	

Table 6.4.1(a): Goat Exhaustive results

Summer				**Winter**			Average Soln Time (mins)
Tot Qg (MVar)	Eloss (kWh)	Vmax (pu)		Tot Qg (MVar)	Eloss (kWh)	Vmax (pu)	
5	19218.01	0.9528		2.5	17729.72	0.9532	12.08

Table 6.4.1(b): GA results for Goat DS in Summer

Summer Peak Day - Goat DS						
Run	Tot Qg (MVar)	Eloss (kWh)	Vmax (pu)	Solution Time (mins)		
				Amba	**PBIL**	**PPBIL**
1	5	19218.01	0.9528	3.42	1.01	3.57
2	5	19218.01	0.9528	3.34	3.12	2.76
3	5	19218.01	0.9528	2.24	1.43	1.79
4	5	19218.01	0.9528	1.66	1.41	3.60
5	5	19218.01	0.9528	3.68	1.93	0.09
6	5	19218.01	0.9528	3.52	1.85	1.22
7	5	19218.01	0.9528	2.82	3.68	2.36
8	5	19218.01	0.9528	1.87	3.43	4.39
9	5	19218.01	0.9528	1.54	4.32	0.44
10	5	19218.01	0.9528	3.56	1.71	4.11
11	5	19218.01	0.9528	1.25	4.28	2.96
12	5	19218.01	0.9528	1.44	0.26	2.81
13	5	19218.01	0.9528	4.19	1.31	1.85
14	5	19218.01	0.9528	4.41	2.88	0.73
15	5	19218.01	0.9528	4.45	0.67	0.42
			Average (mins)	**2.89**	**2.22**	**2.21**
			Std Deviation	**1.13**	**1.30**	**1.41**

Table 6.4.1(c): GA results for Goat DS in Winter

Winter Peak Day - Goat DS						
Run	Tot Qg (MVar)	Eloss (kWh)	Vmax (pu)	Solution Time (mins)		
				Amba	**PBIL**	**PPBIL**
1	2.5	17729.72	0.9532	0.94	3.49	5.84
2	2.5	17729.72	0.9532	4.05	0.18	0.15
3	2.5	17729.72	0.9532	1.47	0.24	5.69
4	2.5	17729.72	0.9532	3.49	2.69	1.81
5	2.5	17729.72	0.9532	3.86	2.21	2.03
6	2.5	17729.72	0.9532	4.05	2.85	0.55
7	2.5	17729.72	0.9532	1.47	2.14	2.82
8	2.5	17729.72	0.9532	3.49	2.02	1.22
9	2.5	17729.72	0.9532	3.86	4.79	2.36
10	2.5	17729.72	0.9532	1.20	3.24	0.42
11	2.5	17729.72	0.9532	3.71	2.81	3.79
12	2.5	17729.72	0.9532	1.37	1.95	1.79
13	2.5	17729.72	0.9532	2.37	4.93	0.93
14	2.5	17729.72	0.9532	0.70	3.39	2.07
15	2.5	17729.72	0.9532	1.88	3.88	1.92
			Average (mins)	**2.53**	**2.72**	**2.23**
			Std Deviation	**1.28**	**1.37**	**1.72**

The following tables indicate the benefit of installing the capacitor(s) on the Goat networks in terms of the different variables shown on the tables.

Table 6.4.1(d): Summer day optimised Goat DS network

Scenario	Tot Cap Qg [MVAr]	Total Import Energy [MWh]	Eloss [MWh]	Eloss [%]	Peak [MW]	System Vminpu	System Vmaxpu	Average System *pf*
Before Compensation	0	659.863	21.045	3.19%	34.75	0.888	0.948	0.959
After Compensation	5	658.213	19.218	2.92%	34.65	0.901	0.953	0.990

Table 6.4.1(e): Winter day optimised Goat DS network

Scenario	Tot Cap Qg [MVAr]	Total Import Energy [MWh]	Eloss [MWh]	Eloss [%]	Peak [MW]	System Vminpu	System Vmaxpu	Average System *pf*
Before Compensation	0	623.427	18.754	3.01%	34.52	0.892	0.951	0.972
After Compensation	2.5	622.482	17.730	2.85%	34.47	0.901	0.953	0.987

6.4.2 GA CODES APPLIED ON JERSEY DS NETWORK

The following tables indicate the Jersey DS GA results for both typical summer and winter peak days. All the Algorithms located the most optimal solutions as follows:

Table 6.4.2: Optimal capacitor locations for Jersey DS

SUMMER **OPTIMAL BUSES**	WINTER **OPTIMAL BUSES**
Orkney Munic 88 Bus1s	Orkney Munic 88 Bus1s
Western Reefs One 88 Bus1s	Western Reefs One 88 Bus1s

Table 6.4.2(a): Jersey Exhaustive results

Summer				**Winter**			Average Soln Time (mins)
Tot Qg (MVar)	Eloss (kWh)	Vmax (pu)		Tot Qg (MVar)	Eloss (kWh)	Vmax (pu)	
4	7636.33	1.0192		4	7924.45	1.0195	4.34

Table 6.4.2(b): GA results for Jersey DS in summer

Summer Peak Day - Jersey DS						
Run	Tot Qg (MVar)	Eloss (kWh)	Vmax (pu)	Solution Time (mins)		
				Amba	**PBIL**	**PPBIL**
1	4	7636.33	1.0192	2.24	2.08	0.13
2	4	7636.33	1.0192	2.69	0.44	2.09
3	4	7636.33	1.0192	0.38	3.01	0.48
4	4	7636.33	1.0192	1.32	1.98	2.54
5	4	7636.33	1.0192	2.78	0.87	2.97
6	4	7636.33	1.0192	0.60	0.39	4.33
7	4	7636.33	1.0192	2.96	2.39	2.54
8	4	7636.33	1.0192	2.54	3.01	2.32
9	4	7636.33	1.0192	0.68	1.98	4.68
10	4	7636.33	1.0192	2.24	2.20	4.15
11	4	7636.33	1.0192	1.50	2.53	0.50
12	4	7636.33	1.0192	1.51	3.20	0.29
13	4	7636.33	1.0192	1.85	1.74	0.42
14	4	7636.33	1.0192	2.19	1.29	0.43
15	4	7636.33	1.0192	1.14	0.32	0.14
			Average (mins)	**1.77**	**1.83**	**1.87**
			Std Deviation	**0.83**	**0.97**	**1.65**

Table 6.4.2(c): GA results for Jersey DS in winter

Winter Peak Day - Jersey DS						
Run	Tot Qg (MVar)	Eloss (kWh)	Vmax (pu)	Solution Time (mins)		
				Amba	**PBIL**	**PPBIL**
1	4	7924.45	1.0195	3.72	2.08	1.04
2	4	7924.45	1.0195	1.81	3.13	3.52
3	4	7924.45	1.0195	1.82	5.21	1.83
4	4	7924.45	1.0195	1.50	3.22	0.31
5	4	7924.45	1.0195	2.03	1.35	0.04
6	4	7924.45	1.0195	0.54	2.19	0.13
7	4	7924.45	1.0195	1.42	0.38	3.99
8	4	7924.45	1.0195	1.28	3.80	2.10
9	4	7924.45	1.0195	1.92	4.92	2.65
10	4	7924.45	1.0195	2.05	0.88	0.81
11	4	7924.45	1.0195	1.69	0.69	0.28
12	4	7924.45	1.0195	0.75	2.50	0.42
13	4	7924.45	1.0195	1.23	0.18	1.80
14	4	7924.45	1.0195	1.34	3.05	0.49
15	4	7924.45	1.0195	1.58	0.13	2.72
Average (mins)				**1.65**	**2.25**	**1.48**
Std Deviation				**0.72**	**1.65**	**1.29**

The following tables indicate the benefit of installing the capacitor(s) on the Jersey networks in terms of the different variables shown on the tables.

Table 6.4.2(d): Summer day optimised Jersey DS network

Scenario	Tot Cap Qg [MVAr]	Total Import Energy [MWh]	Eloss [MWh]	Eloss [%]	Peak [MW]	System Vminpu	System Vmaxpu	Average System *pf*
Before Compensation	0	509.587	7.759	1.52%	28.38	1.012	1.018	0.960
After Compensation	4	509.483	7.636	1.50%	28.37	1.014	1.019	0.996

Table 6.4.2(e): Winter day optimised Jersey DS network

Scenario	Tot Cap Qg [MVAr]	Total Import Energy [MWh]	Eloss [MWh]	Eloss [%]	Peak [MW]	System Vminpu	System Vmaxpu	Average System *pf*
Before Compensation	0	522.583	8.048	1.54%	29.09	1.012	1.018	0.963
After Compensation	4	522.479	7.924	1.52%	29.08	1.014	1.019	0.996

6.4.3 GA CODES APPLIED ON LAMA DS NETWORK

The following tables indicate the Lama DS GA results for both typical summer and winter peak days. All the Algorithms located the most optimal solutions as follows:

Table 6.4.3: Optimal capacitor locations for Lama DS

SUMMER **OPTIMAL BUSES**	WINTER **OPTIMAL BUS**
Driefontein Pumps 44 Bus1	Driefontein Pumps 44 Bus1
West Drie Standby 44 Bus1	West Drie Standby 44 Bus1

Table 6.4.3(a): Lama Exhaustive results

Summer			**Winter**			Average Soln Time (mins)
Tot Qg (MVar)	Eloss (kWh)	Vmax (pu)	Tot Qg (MVar)	Eloss (kWh)	Vmax (pu)	
8	10304.85	1.04758	8	10244.44	1.04668	10.45

Table 6.4.3(b): GA results for Lama DS in summer

Summer Peak Day - Lama DS						
Run	Tot Qg (MVar)	Eloss (kWh)	Vmax (pu)	Solution Time (mins)		
				Amba	**PBIL**	**PPBIL**
1	8	10304.85	1.04758	3.28	2.44	4.32
2	8	10304.85	1.04758	2.62	1.59	1.18
3	8	10304.85	1.04758	1.21	2.34	0.15
4	8	10304.85	1.04758	2.03	0.53	0.08
5	8	10304.85	1.04758	1.93	3.84	2.30
6	8	10304.85	1.04758	0.53	2.30	2.67
7	8	10304.85	1.04758	1.95	2.70	3.49
8	8	10304.85	1.04758	0.20	3.90	0.98
9	8	10304.85	1.04758	0.45	3.62	2.22
10	8	10304.85	1.04758	3.11	4.50	0.95
11	8	10304.85	1.04758	2.08	1.29	1.03
12	8	10304.85	1.04758	2.45	2.15	1.93
13	8	10304.85	1.04758	1.52	0.98	1.51
14	8	10304.85	1.04758	2.39	3.32	5.05
15	8	10304.85	1.04758	4.42	0.91	4.14
			Average (mins)	**2.01**	**2.43**	**2.13**
			Std Deviation	**1.14**	**1.22**	**1.53**

Table 6.4.3(c): GA results for Lama DS in winter

Winter Peak Day - Lama DS						
Run	Tot Qg (MVar)	Eloss (kWh)	Vmax (pu)	Solution Time (mins)		
				Amba	**PBIL**	**PPBIL**
1	8	10244.44	1.04668	3.21	2.27	3.98
2	8	10244.44	1.04668	2.04	1.55	1.10
3	8	10244.44	1.04668	1.22	2.25	0.15
4	8	10244.44	1.04668	2.58	0.50	2.45
5	8	10244.44	1.04668	0.16	3.59	4.89
6	8	10244.44	1.04668	3.68	2.04	2.22
7	8	10244.44	1.04668	2.75	1.09	2.73
8	8	10244.44	1.04668	2.67	4.63	3.59
9	8	10244.44	1.04668	0.16	0.96	0.16
10	8	10244.44	1.04668	0.24	3.66	2.29
11	8	10244.44	1.04668	2.31	2.79	0.32
12	8	10244.44	1.04668	2.29	5.74	5.42
13	8	10244.44	1.04668	2.66	3.51	3.56
14	8	10244.44	1.04668	3.55	4.33	2.64
15	8	10244.44	1.04668	2.63	2.04	1.13
Average (mins)				**2.14**	**2.73**	**2.44**
Std Deviation				**1.17**	**1.49**	**1.66**

The following tables indicate the benefit of installing the capacitor(s) on the Lama networks in terms of the different variables shown on the tables.

Table 6.4.3(d): Summer day optimised Lama DS network

Scenario	Tot Cap Qg [MVAr]	Total Import Energy [MWh]	Eloss [MWh]	Eloss [%]	Peak [MW]	System Vminpu	System Vmaxpu	Average System *pf*
Before Compensation	0	875.216	10.536	1.20%	42.86	1.030	1.042	0.943
After Compensation	8	875.110	10.305	1.18%	42.85	1.040	1.048	0.994

Table 6.4.3(e): Winter day optimised Lama DS network

Scenario	Tot Cap Qg [MVAr]	Total Import Energy [MWh]	Eloss [MWh]	Eloss [%]	Peak [MW]	System Vminpu	System Vmaxpu	Average System *pf*
Before Compensation	0	857.816	10.481	1.22%	41.80	1.032	1.041	0.933
After Compensation	8	857.715	10.244	1.19%	41.80	1.041	1.047	0.991

6.4.4 GA CODES APPLIED ON MMABATHO DS NETWORK

The following tables indicate the Mmabatho DS GA results for both typical summer and winter peak days. All the Algorithms located the most optimal solutions as follows:

Table 6.4.4: Optimal capacitor locations for Mmabatho DS

SUMMER **OPTIMAL BUSES**	WINTER **OPTIMAL BUS**
Mmabatho Main 88 Bus1b	Mmabatho Main 88 Bus1b
Montshiwa 88 Bus1	

Table 6.4.4a): Mmabatho Exhaustive results

Summer				**Winter**			Average Soln Time (mins)
Tot Qg (MVar)	Eloss (kWh)	Vmax (pu)		Tot Qg (MVar)	Eloss (kWh)	Vmax (pu)	
10	14280.44	1.01347		5	13320.20	1.01347	1.9

Table 6.4.4(b): GA results for Mmabatho in summer

Summer Peak Day - Mmabatho DS						
Run	Tot Qg (MVar)	Eloss (kWh)	Vmax (pu)	Solution Time (mins)		
				Amba	**PBIL**	**PPBIL**
1	10	14280.44	1.01347	0.53	0.54	0.63
2	10	14280.44	1.01347	0.79	0.06	0.54
3	10	14280.44	1.01347	0.54	0.91	0.87
4	10	14280.44	1.01347	0.11	1.34	0.06
5	10	14280.44	1.01347	1.32	0.61	0.61
6	10	14280.44	1.01347	0.38	0.68	0.66
7	10	14280.44	1.01347	0.27	0.99	2.24
8	10	14280.44	1.01347	1.08	0.99	0.29
9	10	14280.44	1.01347	2.26	1.06	0.06
10	10	14280.44	1.01347	2.16	1.75	1.49
11	10	14280.44	1.01347	1.28	0.38	0.18
12	10	14280.44	1.01347	0.56	0.76	0.30
13	10	14280.44	1.01347	0.56	1.16	0.06
14	10	14280.44	1.01347	0.95	1.96	0.35
15	10	14280.44	1.01347	0.50	0.51	0.06
			Average (mins)	**0.89**	**0.91**	**0.56**
			Std Deviation	**0.64**	**0.51**	**0.60**

Table 6.4.4(c): GA results for Mmabatho DS in winter

Winter Peak Day - Mmabatho DS						
Run	Tot Qg (MVar)	Eloss (kWh)	Vmax (pu)	Solution Time (mins)		
				Amba	**PBIL**	**PPBIL**
1	5	13320.20	1.01347	0.13	1.78	0.49
2	5	13320.20	1.01347	4.97	0.74	2.21
3	5	13320.20	1.01347	3.30	0.30	0.52
4	5	13320.20	1.01347	0.15	1.59	0.53
5	5	13320.20	1.01347	1.40	0.94	2.46
6	5	13320.20	1.01347	1.08	0.06	0.30
7	5	13320.20	1.01347	3.46	0.06	0.30
8	5	13320.20	1.01347	1.67	0.12	0.30
9	5	13320.20	1.01347	0.60	0.18	0.20
10	5	13320.20	1.01347	3.08	0.63	0.86
11	5	13320.20	1.01347	1.24	1.54	1.14
12	5	13320.20	1.01347	1.24	0.64	0.76
13	5	13320.20	1.01347	0.62	0.89	2.15
14	5	13320.20	1.01347	0.15	0.23	3.38
15	5	13320.20	1.01347	0.94	1.01	0.60
			Average (mins)	**1.60**	**0.71**	**1.08**
			Std Deviation	**1.44**	**0.58**	**0.99**

The following tables indicate the benefit of installing the capacitor(s) on the Mmabatho networks in terms of the different variables shown on the tables.

Table 6.4.4(d): Summer day optimised Mmabatho DS network

Scenario	Tot Cap Qg [MVAr]	Total Import Energy [MWh]	Eloss [MWh]	Eloss [%]	Peak [MW]	System Vminpu	System Vmaxpu	Average System *pf*
Before Compensation	0	1225.696	15.013	1.22%	68.81	0.904	0.992	0.940
After Compensation	10	1224.955	14.280	1.17%	68.75	0.932	1.013	0.987

Table 6.4.4(e): Winter day optimised Mmabatho DS network

Scenario	Tot Cap Qg [MVAr]	Total Import Energy [MWh]	Eloss [MWh]	Eloss [%]	Peak [MW]	System Vminpu	System Vmaxpu	Average System *pf*
Before Compensation	0	1146.453	13.633	1.19%	71.33	0.900	1.003	0.953
After Compensation	5	1146.135	13.320	1.16%	71.29	0.913	1.013	0.979

6.4.5 RESULTS ANALYSIS

On all the networks, the above tables clearly indicate that after the capacitor(s) compensation for both summer and winter typical days, the Energy losses and Active power have been minimised. As the simulation is done over a 24-hour period, the total energy supplied to loads connected (to all the networks) was also reduced due to reduced system losses. The minimum and maximum system voltages were significantly improved. The average system *pf* has also been substantially improved. An important point to be made is that the above mentioned results are optimal for all loading conditions throughout the different hours of the day and different seasons as well. There was consistency in the optimal buses located for summer and winter seasons, although in some cases (Goat and Mmabatho networks) the optimal solution for winter was with one bus less i.e. the summer solution has two optimal buses, but winter only has one optimal bus.

It will then be up to the Operations/ Planning Engineer to decide on whether to have a switchable Capacitor on the bus that is optimal only for one season. This will mean that during the season that is not optimal, the Capacitor will have to be switched out of service. The following tables make a comparison between the different methods used to solve the Optimal Capacitor Placement problem:

Table 6.4.5(a): Average Time per Algorithm

GA Type	Average Time to locate Solution (mins)				Overall Ave Time (mins)
	Goat DS (7)	Jersey DS (6)	Lama DS (7)	Mmabatho DS (5)	
AMBA	2.71	1.71	2.08	1.24	**1.94**
PBIL	2.47	2.04	2.58	0.81	**1.97**
PPBIL	2.22	1.67	2.29	0.82	**1.75**
Exhaustive	12.08	4.34	10.45	1.9	**7.19**

The above table indicates the average time per algorithm per network. The overall average time for each algorithm is then determined. The number of buses per network is shown in brackets next to the network names. It can be seen that the PPBIL algorithm is the fastest at an average of 1.75 minutes. The traditional Exhaustive search algorithm is the slowest at 7.19 minutes. The main reason that the Exhaustive search algorithm takes longer is because it does a search in the entire search space, it then makes a comparison for every evaluated objective function in order to ultimately decide on the best fitness that meets all the constraints defined in the algorithm. In contrast, a genetic algorithm has the ability to converge towards an optimal solution quicker.

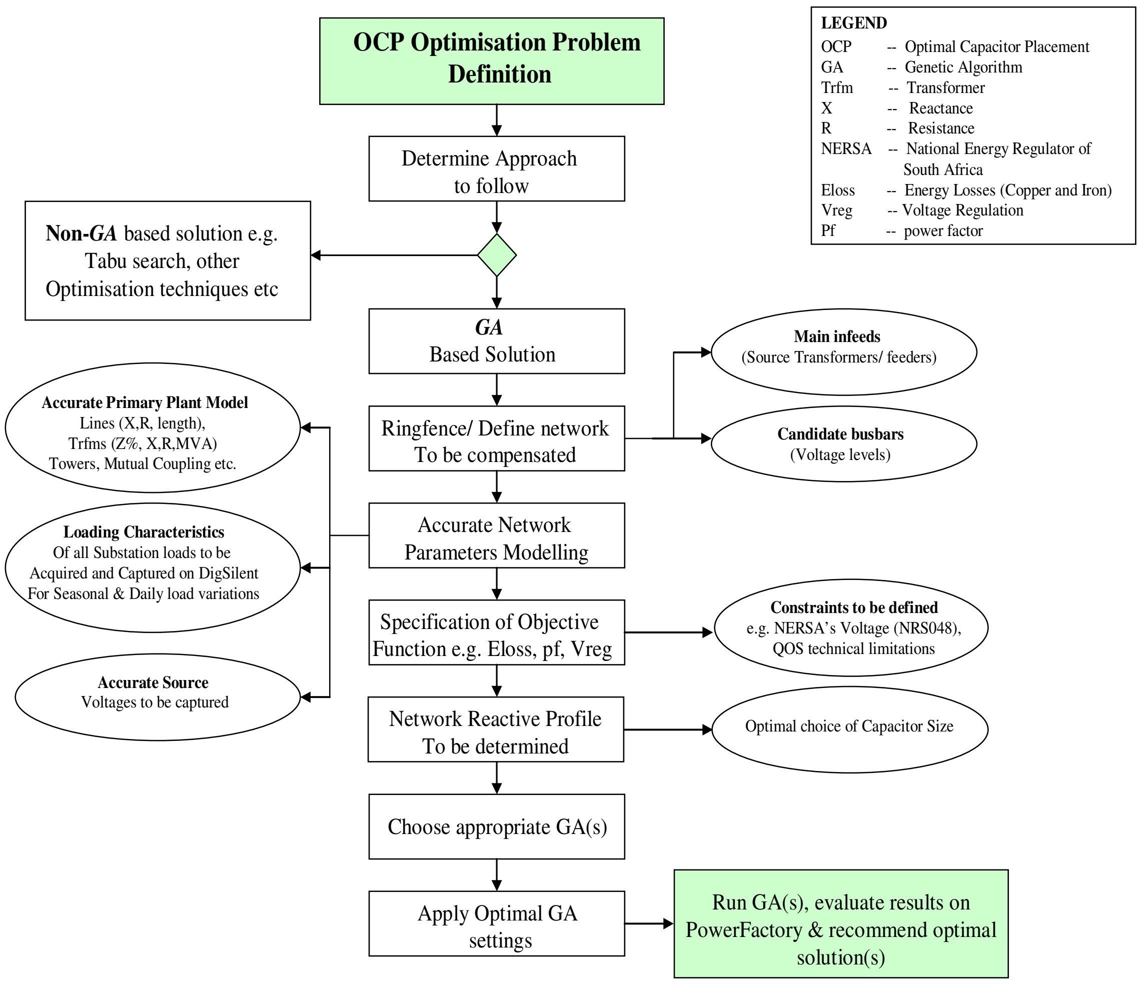
Table 6.4.5(b): Average standard deviation per network

GA Type	**Average Standard deviation per network**				Overall Std Dev (mins)
	Goat DS (7)	Jersey DS (6)	Lama DS (7)	Mmabatho DS (5)	
AMBA	1.21	0.78	1.16	1.04	**1.05**
PBIL	1.34	1.31	1.35	0.54	**1.14**
PPBIL	1.56	1.47	1.60	0.80	**1.36**

The above table shows different standard deviations per network per algorithm. The AMBA algorithm has the least standard deviation, i.e. it is the most reliable and consistent algorithm to locate the optimum solution in the search space. If the fastest speed was desired, the PPBIL algorithm may be used with some trade-off on the reliability. There is a marginal difference between AMBA's and PPBIL's average performance time.

6.5 PROPOSED METHODOLOGY TO SOLVE THE OCP PROBLEM

The following methodology is therefore proposed to solve the Optimal Capacitor placement problem:



LEGEND
OCP -- Optimal Capacitor Placement
GA -- Genetic Algorithm
Trfm -- Transformer
X -- Reactance
R -- Resistance
NERSA -- National Energy Regulator of South Africa
Eloss -- Energy Losses (Copper and Iron)
Vreg -- Voltage Regulation
Pf -- power factor
OCP Optimisation Problem Definition
Determine Approach to follow
Non-GA based solution e.g. Tabu search, other Optimisation techniques etc
GA Based Solution
Main infeeds (Source Transformers/ feeders)
Ringfence/ Define network To be compensated
Candidate busbars (Voltage levels)
Accurate Primary Plant Model Lines (X,R, length), Trfms (Z%, X,R,MVA) Towers, Mutual Coupling etc.
Loading Characteristics Of all Substation loads to be Acquired and Captured on DigSilent For Seasonal & Daily load variations
Accurate Source Voltages to be captured
Accurate Network Parameters Modelling
Specification of Objective Function e.g. Eloss, pf, Vreg
Constraints to be defined e.g. NERSA's Voltage (NRS048), QOS technical limitations
Network Reactive Profile To be determined
Optimal choice of Capacitor Size
Choose appropriate GA(s)
Apply Optimal GA settings
Run GA(s), evaluate results on PowerFactory & recommend optimal solution(s)

Figure 6.5**:** Proposed methodology

The above figure illustrates the steps that an Engineer would follow to solve an optimisation problem of interest. This starts from defining the problem, identifying constraints to be considered, relevant data (network parameters, seasonal load profiles etc) required to ultimately running the GA on the network to be compensated. Also, relevant considerations to be borne in mind are highlighted on the proposed methodology stated above. A sample transformer model with all the technical parameters considered and the implementation of the different GAs is included in the appendices section of this report.

7. CONCLUSIONS AND RECOMMENDATIONS

7.1. CONCLUSIONS

A high demand in reactive power on an electrical network leads to depressed voltages, poor power factor (*pf*) and possibly high technical (copper and transformer iron) losses. The optimal placement of capacitor banks does lead to an improvement of the network's voltage profile, more released network capacity and minimum technical losses. The dissertation achieved the following:

- Developed methodologies (Exhaustive, Linear Deterministic, Simulated Annealing, Tabu Search and Evolutionary Approach) in literature have been thoroughly reviewed and the advantages and disadvantages on these were drawn.
- The optimisation problem was defined broadly and narrowed it further to the context of this dissertation, i.e. the objective function (network technical losses) to be optimised subject to constraints (voltage constraints). Capacitor control options were also outlined.
- Capacitor application considerations were also discussed in terms of prominent network characteristics that impact on the reactive power compensation that would be expected from Compensation Capacitors.
- Genetic Algorithm codes were applied on different networks in DigSilent and have been able to successfully and reliably solve the Optimal Capacitor Optimisation Problem.
- As a solution to typical non-linear optimisation problems (e.g. Optimal Capacitor Placement Optimisation problem) with many variables to be solved, Genetic Algorithms have proved to be a suitable, feasible and realistic approach. As testimony to the stated fact, the few chosen GAs have consistently located the globally-optimum solution for different search spaces, i.e. different networks for different seasons.
- The optimal capacitor problem has been solved for different loading conditions by the GAs. This will enhance the network performance in terms of quality of supply for different load variations during the day, week, weekend, off peak and peak periods.
- A realistic methodology for solving the optimal capacitor placement problem has been developed, tried and tested on the real Eskom sub-transmission network modelled on DigSilent PowerFactory Simulation Package and has produced good engineering results.
- The AMBA algorithm has the least standard deviation, which makes it the most reliable and consistent algorithm to locate the optimum solution in the search space.

7.2. RECOMMENDATIONS

Based on the above stated conclusions, the following recommendations can be made:

- Operations and Planning Engineers need to start considering using GAs for optimal installation of capacitors for voltage support or loss minimisation.
- The Optimal Capacitor Placement methodology needs to be reviewed, rationalised and ratified by the Utility in order to start factoring the methodology into the planning and operational philosophies.
- Optimal Capacitor sizing using Genetic Algorithms needs to be considered for further research as the scope of this research only covered optimal positioning.

8. REFERENCES

[1] Capacitor placement in radial distribution networks through a linear deterministic optimization model. Roberto S. Aguiar, Pablo Cuervo.

[2] Solving the problem of general capacitor placement in radial distribution systems with laterals using simulated annealing. Branko D. Stojanović, Miloš S. Nedeljković.

[3] A tutorial on tabu search, Alain Hertz, Eric Taillard, Dominique De Werra

[4] An evolutionary approach for capacitor placement in distribution networks, Alexandre Mendes, Paulo M. França, Christiano Lyra, Cristiane Pissarra And Celso Cavelucci

[5] Eskom's network planning guideline for MV shunt capacitors, Cg Carter-Brown

[6] EPRI power system dynamics tutorial copyrighted material active and reactive power draft march, 2002

[7] Power system analysis, operation and control lecture notes University of Cape Town EEE490f, 2002, Alexander I Petroianu

[8] Reactive power and importance to bulk power system, Oak Ridge National Laboratory Engineering Science & Technology Division

[9] Dynamic performances of the hierarchical voltage regulation: the italian EHV system case, A.Berizzi, M.Merlo, P.Marannino, F.Zanellini, S. Corsi & M.Pozzi

[10] An introduction to reactive power: the national grid company pl, market development: October 2001

[11] Comparative evaluation of evolutionary design methods in engineering, S.G Mkwelo, 2001

[12] Evolutionary Algorithms In Theory And Practice, New York Oxford University Press 1996, Thomas Back

[13] Evolutionary computation: comments on the history and current state, Thomas Back, Et Al IEEE Trans. On Evolutionary Computation Vol1, No 1, April 1997 Pp 15 – 24

[14] Design optimization of electrical machines using genetic algorithms, G.F Uler, O.A Mohammed, C.S Koh IEEE Transaction On Magnetics, Vol 31, No 3, April 1990

[15] Handbook of genetic algorithms, Lawrence Davis, International Thomson Computer Press, 1996

[16] AMBA - An adaptively mutating breeder algorithm for global stochastic optimization, J.R Greene, University Of Cape Town, 2000

[17] Optimisation of the design of a small permanent magnet (PM) synchronous generator for isolated operation, M Ntusi, University Of Cape Town, 2002

[18] Permanent magnet motor technology design and applications, Jacek F Gieras, Mitchell Wing, New York: Marcel Decker, C1997

[19] PBIL - A method of integrating genetic search based function optimization and competitive learning, S. Baluja

[20] Specification of franke's mv capacitor banks and capacitors, http://www.frankeenergy.com

[21] Network planning guideline for MV shunt capacitors, CG Carter-Brown, 34-598, June 2007

[22] Power distribution planning reference book, By H. Lee Willis

[23] Electricity supply – quality of supply part2, NRS048-2:2004, NERSA

[24] Capacitor switching transients: analysis and proposed technique for identifying capacitor size and location. Mohamed M. Saied*, Senior Member, Ieee*

[25] Voltage control fundamentals, Gav Hurford, Eskom System Operations And Planning

[26] Optimal capacitor placement for loss reduction, F. Mahmoodianfard H. Askarian. Abyaneh S. Jabarooti F. Razavi

9. APPENDICES

9.1 APPENDIX A1

Matlab AMBA code:

```
% BGA   Breeder genetic Algorithm with adaptive mutation
clear

maxgen    = 100 ;
pop       = 200 ;
nvars     = 20  ;

thr =round(pop*15/100);             % 15% selection threshold
fitrec = [];
verybest = [];
for run = 1:20                              % performs multiple runs
  delta = 0.1;
  T = rand(nvars,pop) ;             % initial matrix of trial solutions
  for gen = 1:maxgen
     f = mbump(T);
     f  = f';                    % row vector of fitnesses
     flow =  mean(f(2:100));       % mean fitness with lower mutation
     fhigh = mean(f(101:200));     % mean fitness with higher mutation

     if flow > fhigh               % if lower mutation improves fitness
       delta = 0.95*delta;             % decrease mutation rate
     else
       delta = 1.05*delta;             % otherwise, increase it
     end

     [f,i] = sort(-f);             % sort by fitness in descending order
     S = T(:,i);                  % Sorted trial solutions
     best =mbump(S(:,1));           % best fitness sofar:
     fitrec = [fitrec,best];        % record it
     pool = S(:,1:thr);             % survivors (breeding pool)
     T(:,1) = S(:,1);              % elitist insertion
     for i =2:200                 % construct rest of population
       R = randperm(thr);           % pick two different parents at random
       rnd = rand;
       if rnd < 0.15              % discrete recombination
       mask = rand(nvars,1) > 0.5;
       T(:,i) = mask.*pool(R(1)) +(~mask).*pool(R(2));
       else
            rr = -0.25+ 1.5*rand(nvars,1);  % volume recomb
            % rr = rand(nvars,1);
            T(:,i) = rr.*pool(:,R(1)) + (1-rr).*pool(:,R(2));
       end
       % mutate lower/higher half of population at lower/higher rate
       r1 = 1+floor(nvars*rand);    % random integer in range (1,nvars)
       if i<101
         T(r1,i) = T(r1,i)+delta*(randn/1.1);
       else
         T(r1,i) = T(r1,i)+delta*(1.1*randn);
       end
    end
    disp(['run ', 'generation ','mutation rate ','current best'])
    disp([run,gen,delta,best])
    plot(fitrec)                    % incremental plot of fitness
    drawnow
   end
verybest = [verybest,best]                  % append to list of very=best
end
```

Matlab PBIL Code:

```
%  A REALISTIC PBIL IMPLEMENTATION: Optimising 'BUMP' in 20 dimensions.
% This is a simple but useful implementation of Population-based
% Incremental Learning suitable for general purpose function
% optimisation.  It is currently set up to find the global optimum
% of a complex 20-dimensional function called 'BUMP'(and defined in
% the m-file mbump.m.  It passes to mbump a column-vector of NVARS
% (in this case 20) real numbers in the range 0-1.
% To optimise some other function (say FUNC) simply edit the two lines
% below marked with XXXXXXXX.  Change NVARS to the number of variables
% required by FUNC,  change MBUMP to FUNC.  Write an m-file defining
% FUNC and save it as FUNC.m (alternatively, if it is a simple function
% you could simply define it in-line in place of the function call).
% Remember to scale the 0-1 input variables into the rage required by
% FUNC. For simpler functions than BUMP, MAXGEN and NTRIALS (the
% population size) can be reduced (with consequent time-saving).  PREC
% sets the precision (currently 12 bits). For many engineering problems
% 7 bits (1/2%)is adequate; reducing PREC simplifies the search somewhat.
% The program is set up to MAXIMISE the function. To minimise it, simply
% use the fact that finding argmin(F) is equivalent to finding argmax(-F)
% To optimise a function with a bitstring argument, simply omit the
% binary to decimal conversion immediately before the function all
% ( x = reshape(....) and pass the bitstring ts to the function.

fitrec   =    [];
bestever =  -inf;
nvars    =    20;                % number of variables  XXXXXXXXXXXXXX
maxgen   =   200;                    % number of generations
prec     =    12;                 % variable precision (number of bits)
ntrials  =   100;                % population size

bw =  2 .^((prec-1):-1:0)/2^prec;   % bitweights for bin 2 dec conversion
PV =  0.5*ones(1,nvars*prec);       % probability vector (initially 0.5's)

for g = 1:maxgen
    bestfit = -inf;
    for t = 1: ntrials
      ts = rand(size(PV)) <  PV;          % trial solution in binary
      x =  reshape(ts,nvars,prec) * bw' ; % NVARS numbers in range 0-1
     %-------------------------------------------------
      f  =  mbump(x);  % place function or function call here
     %-------------------------------------------------
     if f>bestfit               % if improved fitness
        bestfit = f;            % update fitness
        bestsol = ts;            % store best (binary) trial solution
     end
    end

    disp('Maxgen,      Gen,      Best,     Bestever');
    disp([maxgen,g,bestfit,bestever]);

    PV = 0.9*PV + 0.1*bestsol;      % update probability vector
    PV = PV - 0.005*(PV-0.5);       % relax to maintain diversity

    fitrec = [fitrec, bestfit];     % append fitness to fitness record
    if bestfit > bestever            % if improvement on best to date
        bestever = bestfit;         % update best-to-date
        besteversol = bestsol;      % store best ever solution (binary)
    end
end

plot(fitrec)                       % plot fitness over time
bestever                           % display best fitness attained

result = reshape(besteversol,nvars,prec)*bw' %convert best result to decimal
bestfitness = mbump(result)          % and evaluate function
```

Matlab PPBIL Code:

```
%  Parallel Population-based Learning  pPBIL
%  This version runs multiple PBIL populations.  Each population
%  has two probability vectors associated with it. PVa and PVb.
%  Each trial solution in a given population is generated by
%  randomly sampling an interim PV assembled from PVa and PVb
%  by uniform crossover (i.e. randomly selecting bits from one
%  or the other)  The best in a set (or 'generation') of trial
%  solutions from a given population is used to update both PVa
%  and PVb for that population.   Periodically (when the populations
%  have begun to converge) the PVb vectors are cyclically
%  interchanged between the populations (say every 50-100 generations).

clear all

lr     =    0.1;
ff     =      0;    % (seems unnecessary in parallel version)

fitrec =   [];
nvars   =  20;            % num of decision variables
prec    =  12;            % bit-precision of each variable
ntrials =  10;            % num trial solutions per population
maxgen  = 40%0;            % num of generations before migration
npops   =  10;            % num of parallel populations
maxrun =   10;            % num of migration cycles

strlen  = nvars*prec;     % length of bitstring

% Initialise 2*<npops> probability vectors. Each population
% has two PVs, PVA and PVB.   Rows 1:10 are the PVAs for
% populations 1:10, and rows 11:20 the PVBs for populations 1:10
                                % Initialisation
best   = -inf;
evaluations =   0;
PM = 0.5 * ones(2*npops,strlen);

fmax = 0;

for run = 1: maxrun

for gen = 1:maxgen

  disp(['       ','run  ','generation  ', 'evalsx10^3  ' , 'fmax  ' ,'best  '])
  disp([run, gen ,evaluations/1000, fmax , best])

  for pop = 1:npops

      % using each pair pop,pop+10 of rows from PM, create a
      % matrix P of ntrials PVs by random uniform crossover

      pva = ones(ntrials,1)*PM(pop,:);
      pvb = ones(ntrials,1)*PM(pop+10,:);

            randmask =(rand(ntrials,strlen)>0.5); % rand binary mask
      P = randmask.*pva+(~randmask).*pvb;   % ntrials PVs


      % Sample the resulting probability matrix with <ntrials>
      % random row vectors to create <ntrials> trial solutions

            trialsolutions = P > rand(size(P));


      % evaluate the trialsolutions and update both the A and B
      % probability vectors of the current population toward it.
      % Maintain diversity by relaxing both PVs toward 0.5.
```

```
            fmin =  inf;
        fmax = -inf;

        for t = 1:ntrials

          ts = trialsolutions(t,:);
          f =  bump20(ts);
          evaluations = evaluations+1;
          if f > fmax
            fmax    =  f;
            bestsol = ts;
          end
                if f < fmin;
            fmin = f;
            worst = ts;
          end
        end

        if fmax > best
          best = fmax;
          fitrec = [fitrec,best];
        end

        PM(pop,:)   =(1-lr)*PM(pop,:)    + lr*bestsol;
        PM(pop+10,:)=(1-lr)*PM(pop+10,:) + lr*bestsol;

            PM(pop,:) = PM(pop,:) - ff*(PM(pop,:)-0.5);
        PM(pop+10,:) = PM(pop+10,:) - ff*(PM(pop+10,:)-0.5);

    end

end

  disp('migrating...')       % Interchange of 'B'
  N(1,:) = PM(20,:);         % probability vectors
  N(2:10,:) = PM(1:9,:);
  PM(11:20,:) = N;

end

save 'fitrec'
plot(fitrec)
```

9.2 APPENDIX A2

DPL AMBA code:

```
int ts,gen,maxgen,maxrun,i,j,popsize,nvars,hr,size,k,x,y,val,u,ind,run,cnt1;
double w1,w2,b,bb,v,v1,thr,fl,fh,delta,Absminl,flt,fht,popind;
double L1,L2,tim,tim1,tim2,a1,a2,a3,a4,tims,sam,qtmp1,Vmxtmp1;
double qtmp2,Eltmp2,Vmxtmp2,rdom,rdom2,ofsp,par1,par2,grace,Eloss;
double r1,rnd1,rnd2,rnd3,rnd4,rnd5,par1i,par2i,prn,caps,capi,Lsi,hri,chld,chld2,chld3,rr,rr1;
double Optbusi,minL,Lbusi,hri2,bw,pv,rndn,rndn1,rndn2,rndn3,yek,c1,c2,cr1,cr2,cc1,cc2,tmm;
int str,q,callloss,strc,sc,busi,mara,wel,tsv,prec,sm,Chksum,sumtot,vcheck,kk,ki,sumi,si,ct;
string nm,str1,str2,capt,termii;
set Bf,B,Bcheck,allcaps;
int zz,ii,ntu,zek,gencheck,nam,boy,ctp,juy;
int bb1,bb2,bb3,bb4,bb5,bb6,bb7,bb8,bb9,bb10,bb11,bb12,bb13,bb14,ib;
int bb15,bb16,bb17,bb18,bb19,bb20,bb21,bb22,bb23,bb24,nzp,viol;
int bb1i,bb2i,bb3i,bb4i,bb5i,bb6i,bb7i,bb8i,bb9i,bb10i,bb11i,bb12i,bb13i,bb14i;
int bb15i,bb16i,bb17i,bb18i,bb19i,bb20i,bb21i,bb22i,bb23i,bb24i,zkk,tnj,ik,msk,pop2,bru;
object t,t1,bus,Optbus,userset,zm,cpp,termi;
double Qgi,Qgi1,Qgi2,bit1,bit2,bit3,bv1,bv2,bv3,sumq,minl,Vmaxa,Qmin,pv1,pv2,pv3,Eli;
double Qgt, Elosst, Vmaxt, vioL1, vioL2,vv1,vv2,qq1,qq2,qq3,qq4,qq5;

for(bru=1;bru<=20;bru=bru+1)
{

str = 0;

tim1 = GetTime(3);
userset = GetCaseObject('SetUser');
popsize = 50;
maxgen = 2;
maxrun = 2;
thr = round(popsize*15/100);            ! 15% selection threshold
size = popsize+1;
popind = popsize/2;

fl = 0;
fh = 0;
ts = 0;
ct = 0;
juy = 0;

Vmaxa = 1.1;
Qmin = 100;
ctp = 1;
cnt = 1;
minl = 100000;
Absminl = 100001;
Eloss = 25000;
pop2 = popsize;

 Bf = Terms.Get();
 allcaps = capss.Get();

 t = Bf.First();

 while(t)
 {

   if(t:e:iUsage = 'Busbar')      ! Only consider busbars not terminals
    {
     if(t:e:uknom = 88)            ! Only optimise at specified kV, e.g.44kV
      {
       B.Add(t);
       ts = ts + 1;               ! Determine how many busbars the grid has
       !printf('Found %s',t:loc_name);
     }
    }
    t = Bf.Next();
 }

 !printf('There are %.0f buses',ts);
 !input(str,'What on earth???');
 pv = 0.5;
 cnt1 = 1;
 prec = 7;
 ClearOutput();                        ! Clear output window!
 Chksum = 0;

 bv1 = V.Get(1);
 !v2 = V.Get(2);
 !bv3 = V.Get(3);
 zkk = ts*1;
 zkk = pow(2,zkk);
 !zkk = 10;

 if(ts > 1)
 {

         ctp = 1;
         !while(ctp <= maxrun)
         while(minl > 17730)           ! winter 17730
         !while(Vmaxa > 1.05)
         {
          fl = 0;
          fh = 0;
          delta = 0.1;
          Bcheck.Clear();
          zek = 1;

          for(ii=1; ii<=popsize; ii=ii+1)
          {
            ClearOutput();
            sumq = 0;
            gencheck = 0;              ! Reset gen soln check
            ib = 2;
            for(i=1; i<=ts ;i=i+1)      ! T = random(nvars,pop)
            {                           ! Initialise the population size
              T.Set(i,1,i);             ! Store bus position
              Tchk.Set(zek,ib-1,i);     ! Store bus position

              rndn1 = Random();

              T.Set(i,2,0);             ! Reset previous values
              T.Set(i,3,0);

              if(rndn1 < 0.5)
              {
                T.Set(i,2,1);
                Tchk.Set(zek,ib,1);
               }
              else
              {
                T.Set(i,2,0);
                Tchk.Set(zek,ib,0);
               }

             bit1 = T.Get(i,2);

             Qgi = bit1*bv1;
             T.Set(i,3,Qgi);            ! Store the equivalent total cap generation

             sumq = sumq + Qgi;
 ib = ib + 2;
 }

 Tchk.Set(zek,25,sumq);

 !input(str,'What on earth???');

     bb1i = Tchk.Get(zek,2);      ! Get current generation
     bb2i = Tchk.Get(zek,4);
     bb3i = Tchk.Get(zek,6);
     bb4i = Tchk.Get(zek,8);
     bb5i = Tchk.Get(zek,10);
     bb6i = Tchk.Get(zek,12);
     bb7i = Tchk.Get(zek,14);
     bb8i = Tchk.Get(zek,16);
     bb9i = Tchk.Get(zek,18);
     bb10i = Tchk.Get(zek,20);
     bb11i = Tchk.Get(zek,22);
     bb12i = Tchk.Get(zek,24);

 if(zek > 1)                    ! Checks if generation already exists
 {
    for(zz=1;zz<zek;zz=zz+1)
    {
     bb1 = Tchk.Get(zz,2);
     bb2 = Tchk.Get(zz,4);
     bb3 = Tchk.Get(zz,6);
     bb4 = Tchk.Get(zz,8);
     bb5 = Tchk.Get(zz,10);
     bb6 = Tchk.Get(zz,12);
     bb7 = Tchk.Get(zz,14);
     bb8 = Tchk.Get(zz,16);
     bb9 = Tchk.Get(zz,18);
     bb10 = Tchk.Get(zz,20);
     bb11 = Tchk.Get(zz,22);
     bb12 = Tchk.Get(zz,24);

     if({bb1i=bb1}.and.{bb2i=bb2}.and.{bb3i=bb3}.and.{bb4i=bb4}
        .and.{bb5i=bb5}.and.{bb6i=bb6}.and.{bb7i=bb7}.and.{bb8i=bb8}
        .and.{bb9i=bb9}.and.{bb10i=bb10}.and.{bb11i=bb11}.and.{bb12i=bb12})
      {
      !input(str,'What on earth???');
      Qgt    = Tchk.Get(zz,25);        ! Invoke the calculated values already
      Elosst = Tchk.Get(zz,26);
      Vmaxt  = Tchk.Get(zz,27);
      vv1    = Tchk.Get(zz,28);
      Tchk.Set(zek,25,Qgt);            ! Write invoked values into generation
      Tchk.Set(zek,26,Elosst);
      Tchk.Set(zek,27,Vmaxt);
      Tchk.Set(zek,28,vv1);
      gencheck = 1;
      !popsize = popsize + 1;   ! Because of the match, compensate with 1 more generation
      !input(str,'What on earth???');
      }
    }
 }


    if(gencheck = 0)       ! Check if soln hasn't been located before
    {
      cpp = allcaps.First();
      termi = B.First();
      !capt = cpp:r:bus1:r:cBusBar:loc_name;
      tnj = 2;

      for(zz=1; zz<=ts; zz=zz+1) ! Implement the random generation
      {
```

```
termii = termi:loc_name;                ! Name of busi
while(cpp)
{
  capt = cpp:r:bus1:r:cBusBar:loc_name; !Invoke bus connected to cap
  ntu = strcmp(termii,capt);
  if(ntu = 0)                   ! If capi is connected to busi
   {
    cpp:outserv = 0;         ! Ensure cap is in service
    !Qgi = T.Get(zz,4);
    !if(zz=1)
    !{
     Qgi1 = Tchk.Get(zek,tnj);
     !Qgi2 = Tchk.Get(zek,3);
     Qgi = bv1*Qgi1;
     cpp:qtotn = Qgi;         ! Tap the capacitor to Qgi
    !}
    !else
    !{
     !Qgi1 = Tchk.Get(zek,zz+3);
     !Qgi2 = Tchk.Get(zek,zz+4);
     !Qgi = (2*Qgi1) + Qgi2;
     !cpp:qtotn = Qgi;
    !}
    !input(str,'What on earth???');
    if(Qgi = 0)
    {
     cpp:outserv = 1;        ! If Qgi = 0, put cap out of service to ensure load flow solves
    }
   break;
    }
   !printf('Cap %.0f tapped to %.0f MVar',zz,Qgi);
   cpp = allcaps.Next();         ! Invoke cap-i
 }
 termi = B.Next();
 tnj = tnj + 2;
}
 !input(str,'What on earth???');
 Qmin = sumq;
 Losses.Execute(B,Eloss,hr,Vmaxa,viol);   ! Execute losses script with bus index & object bus
 Tchk.Set(zek,26,Eloss);            ! Keep track of Eloss per gen
 Tchk.Set(zek,27,Vmaxa);
 Tchk.Set(zek,28,viol);
 !input(str,'What on earth???');
 !R.WriteDraw();
if(viol = 0)           ! Only consider if volt limits are not
{                      ! violated
 if(Eloss < minl)
 {
  tims = GetTime(3);
  tim = (tims-tim1)/60;   ! compute time taken to locate soln
  !ctp = ctp + 1;
  minl = Eloss;

  T2.Set(cnt1,1,0);          ! clear old values
  T2.Set(cnt1,2,0);
  T2.Set(cnt1,3,0);          ! Reset Eloss
  T2.Set(cnt1,4,0);          ! Reset Vmaxpu
  T2.Set(cnt1,5,0);

  T2.Set(cnt1,1,cnt1);          ! Store run i
  T2.Set(cnt1,2,sumq);        ! Store total Qgen [MVAr]
  T2.Set(cnt1,3,Eloss);       ! Store Eloss i
  T2.Set(cnt1,4,Vmaxa);
  T2.Set(cnt1,5,tim);

  !Eltmp1 = Eloss;          ! Eltmp1 is used to plot fitness
  !R.WriteDraw();
  cnt1 = cnt1 + 1;

  ik = 2;                    ! Keep track of best generation
  for(i=1; i<=ts ;i=i+1)
   {
     T3.Set(i,3,0);              !clear old values
     T3.Set(i,2,0);
     T3.Set(i,1,0);

     bit1 = Tchk.Get(zek,ik);    !get stored bits for best gen
     Qgi = bv1*bit1;

     T3.Set(i,3,Qgi);
     T3.Set(i,2,bit1);
     T3.Set(i,1,i);
     ik = ik + 2;
    }
  }
  !zek = zek + 1;
 }     ! end if(viol....

 !else                ! If viol = 1, adjust pop size
 !{
 ! popsize = popsize + 1;
 !}

 }       !end if(gencheck....

 zek = zek + 1;
 !if(ii <= (size-1)/2)  ! Calculate fitnesses for the 1st half of the population
 !{fl = fl + Eloss;}

 !if(ii > (size-1)/2)   ! Calculate fitnesses for the 2nd half of the population
 !{fh = fh + Eloss;}

} ! end for(ii=1....

!input(str,'What on earth???');

pop2 = size-1;               ! Reset pop size

for(gen=1;gen<=maxgen;gen=gen+1)
{
 zek = 2;                    ! Initialise new count
 fl = 0;
 fh = 0;
 !minl = 100000;

 for(i=1;i<size;i=i+1)       ! Calculate mean fitnesses
 {
  if(i <= popind)           ! for lower & higher populations
  {
    flt = Tchk.Get(i,26);
    fl = fl + flt;
  }
  if(i > popind)
  {
    fht = Tchk.Get(i,26);
    fh = fh + fht;
  }
 }

 fl = fl/popind;
 fh = fh/popind;
 !fl = fl/(0.5*(zek-1));     ! mean fitness with lower mutation
 !fh = fh/(0.5*(zek-1));     ! mean fitness with higher mutation
```

```
!input(str,'What on earth???');
if(fl > fh)              ! If lower mutation improves fitness, decrease mutation rate
{ delta = 0.95*delta; }
else
{ delta = 1.05*delta; }    ! Otherwise increase it

for(y=1;y<size-1;y=y+1)           ! This code sorts fitnesses from best to worst
{
 for(x=1;x<size-1;x=x+1)
 {
  L1 = Tchk.Get(x,26);
  L2 = Tchk.Get(x+1,26);
  vioL1 = Tchk.Get(x,28);
  vioL2 = Tchk.Get(x+1,28);
  !printf('x = %.0f, y = %.0f',x,y);
  if({L1 > L2})     ! L1 > L2, because best loss is less
  {
    bb1i = Tchk.Get(x,2);      ! Get current generation
    bb2i = Tchk.Get(x,4);
    bb3i = Tchk.Get(x,6);
    bb4i = Tchk.Get(x,8);
    bb5i = Tchk.Get(x,10);
    bb6i = Tchk.Get(x,12);
    bb7i = Tchk.Get(x,14);
    bb8i = Tchk.Get(x,16);
    bb9i = Tchk.Get(x,18);
    bb10i = Tchk.Get(x,20);
    bb11i = Tchk.Get(x,22);
    bb12i = Tchk.Get(x,24);
    qtmp1  = Tchk.Get(x,25);
    Eltmp1 = Tchk.Get(x,26);
    Vmxtmp1 = Tchk.Get(x,27);
    vv1 = Tchk.Get(x,28);

    bb1 = Tchk.Get(x+1,2);
    bb2 = Tchk.Get(x+1,4);
    bb3 = Tchk.Get(x+1,6);
    bb4 = Tchk.Get(x+1,8);
    bb5 = Tchk.Get(x+1,10);
    bb6 = Tchk.Get(x+1,12);
    bb7 = Tchk.Get(x+1,14);
    bb8 = Tchk.Get(x+1,16);
    bb9 = Tchk.Get(x+1,18);
    bb10 = Tchk.Get(x+1,20);
    bb11 = Tchk.Get(x+1,22);
    bb12 = Tchk.Get(x+1,24);
    qtmp2  = Tchk.Get(x+1,25);
    Eltmp2 = Tchk.Get(x+1,26);
    Vmxtmp2 = Tchk.Get(x+1,27);
    vv2 = Tchk.Get(x+1,28);

    Tchk.Set(x,2,bb1);      ! Get current generation
    Tchk.Set(x,4,bb2);
    Tchk.Set(x,6,bb3);
    Tchk.Set(x,8,bb4);
    Tchk.Set(x,10,bb5);
    Tchk.Set(x,12,bb6);
    Tchk.Set(x,14,bb7);
    Tchk.Set(x,16,bb8);
    Tchk.Set(x,18,bb9);
    Tchk.Set(x,20,bb10);
    Tchk.Set(x,22,bb11);
    Tchk.Set(x,24,bb12);
    Tchk.Set(x,25,qtmp2);
    Tchk.Set(x,26,Eltmp2);
    Tchk.Set(x,27,Vmxtmp2);
    Tchk.Set(x,28,vv2);
```

```
   Tchk.Set(x+1,2,bb1i);      ! Get current generation
   Tchk.Set(x+1,4,bb2i);
   Tchk.Set(x+1,6,bb3i);
   Tchk.Set(x+1,8,bb4i);
   Tchk.Set(x+1,10,bb5i);
   Tchk.Set(x+1,12,bb6i);
   Tchk.Set(x+1,14,bb7i);
   Tchk.Set(x+1,16,bb8i);
   Tchk.Set(x+1,18,bb9i);
   Tchk.Set(x+1,20,bb10i);
   Tchk.Set(x+1,22,bb11i);
   Tchk.Set(x+1,24,bb12i);
   Tchk.Set(x+1,25,qtmp1);
   Tchk.Set(x+1,26,Eltmp1);
   Tchk.Set(x+1,27,Vmxtmp1);
   Tchk.Set(x+1,28,vv1);
  }
 }
}

for(y=1;y<size-1;y=y+1)    ! This code sorts fitnesses from non violations to violations
{
 for(x=1;x<size-1;x=x+1)
 {
  L1 = Tchk.Get(x,26);
  L2 = Tchk.Get(x+1,26);
  vioL1 = Tchk.Get(x,28);
  vioL2 = Tchk.Get(x+1,28);
  !printf('x = %.0f, y = %.0f',x,y);
  if({vioL1 > vioL2})     ! L1 > L2, because best loss is less
  {
   bb1i = Tchk.Get(x,2);      ! Get current generation
   bb2i = Tchk.Get(x,4);
   bb3i = Tchk.Get(x,6);
   bb4i = Tchk.Get(x,8);
   bb5i = Tchk.Get(x,10);
   bb6i = Tchk.Get(x,12);
   bb7i = Tchk.Get(x,14);
   bb8i = Tchk.Get(x,16);
   bb9i = Tchk.Get(x,18);
   bb10i = Tchk.Get(x,20);
   bb11i = Tchk.Get(x,22);
   bb12i = Tchk.Get(x,24);
   qtmp1  = Tchk.Get(x,25);
   Eltmp1 = Tchk.Get(x,26);
   Vmxtmp1 = Tchk.Get(x,27);
   vv1 = Tchk.Get(x,28);

   bb1 = Tchk.Get(x+1,2);
   bb2 = Tchk.Get(x+1,4);
   bb3 = Tchk.Get(x+1,6);
   bb4 = Tchk.Get(x+1,8);
   bb5 = Tchk.Get(x+1,10);
   bb6 = Tchk.Get(x+1,12);
   bb7 = Tchk.Get(x+1,14);
   bb8 = Tchk.Get(x+1,16);
   bb9 = Tchk.Get(x+1,18);
   bb10 = Tchk.Get(x+1,20);
   bb11 = Tchk.Get(x+1,22);
   bb12 = Tchk.Get(x+1,24);
   qtmp2  = Tchk.Get(x+1,25);
   Eltmp2 = Tchk.Get(x+1,26);
   Vmxtmp2 = Tchk.Get(x+1,27);
   vv2 = Tchk.Get(x+1,28);

   Tchk.Set(x,2,bb1);      ! Get current generation
```

```
    Tchk.Set(x,4,bb2);
    Tchk.Set(x,6,bb3);
    Tchk.Set(x,8,bb4);
    Tchk.Set(x,10,bb5);
    Tchk.Set(x,12,bb6);
    Tchk.Set(x,14,bb7);
    Tchk.Set(x,16,bb8);
    Tchk.Set(x,18,bb9);
    Tchk.Set(x,20,bb10);
    Tchk.Set(x,22,bb11);
    Tchk.Set(x,24,bb12);
    Tchk.Set(x,25,qtmp2);
    Tchk.Set(x,26,Eltmp2);
    Tchk.Set(x,27,Vmxtmp2);
    Tchk.Set(x,28,vv2);

    Tchk.Set(x+1,2,bb1i);      ! Get current generation
    Tchk.Set(x+1,4,bb2i);
    Tchk.Set(x+1,6,bb3i);
    Tchk.Set(x+1,8,bb4i);
    Tchk.Set(x+1,10,bb5i);
    Tchk.Set(x+1,12,bb6i);
    Tchk.Set(x+1,14,bb7i);
    Tchk.Set(x+1,16,bb8i);
    Tchk.Set(x+1,18,bb9i);
    Tchk.Set(x+1,20,bb10i);
    Tchk.Set(x+1,22,bb11i);
    Tchk.Set(x+1,24,bb12i);
    Tchk.Set(x+1,25,qtmp1);
    Tchk.Set(x+1,26,Eltmp1);
    Tchk.Set(x+1,27,Vmxtmp1);
    Tchk.Set(x+1,28,vv1);

   }
  }
 }

!input(str,'What on earth???');
Eltmp1 = Tchk.Get(1,26);
R.WriteDraw();
cnt = cnt + 1;                    ! increment the fitness x value

for(zz=1;zz<size;zz=zz+1)   ! Store the fitnesses from best to worse
{
 ik = 2;
 qtmp2  = Tchk.Get(zz,25);
 Eltmp2 = Tchk.Get(zz,26);
 Vmxtmp2 = Tchk.Get(zz,27);
 v1      = Tchk.Get(zz,28);

 Tbest.Set(zz,25,qtmp2);    ! Store best tot MVar generation
 Tbest.Set(zz,26,Eltmp2);   ! Store kWh losses
 Tbest.Set(zz,27,Vmxtmp2);  ! Store Vmax per generation
 Tbest.Set(zz,28,v1);       ! Store voltage violation check

 for(i=1;i<=ts;i=i+1)
 {
   nzp = Tchk.Get(zz,ik);     ! Invoke best bit
   Tbest.Set(zz,ik-1,i);      ! Store bus position
   Tbest.Set(zz,ik,nzp);      ! Store best bit

   ik = ik + 2;
 }
}

bb1i = Tchk.Get(1,2);      ! Get best generation
bb2i = Tchk.Get(1,4);
```

```
 bb3i = Tchk.Get(1,6);
 bb4i = Tchk.Get(1,8);
 bb5i = Tchk.Get(1,10);
 bb6i = Tchk.Get(1,12);
 bb7i = Tchk.Get(1,14);
 bb8i = Tchk.Get(1,16);
 bb9i = Tchk.Get(1,18);
 bb10i = Tchk.Get(1,20);
 bb11i = Tchk.Get(1,22);
 bb12i = Tchk.Get(1,24);
 qtmp1  = Tchk.Get(1,25);
 Eltmp1 = Tchk.Get(1,26);
 Vmxtmp1 = Tchk.Get(1,27);

!if(Eltmp1 < Absminl)
!{
! Absminl = Eltmp1;

! ik = 2;
! for(i=1; i<=ts ;i=i+1)
! {
!   T3.Set(i,3,0);             !clear old values
!   T3.Set(i,2,0);
!   T3.Set(i,1,0);

!   bit1 = Tchk.Get(1,ik);   !get stored bits for best gen
!   Qgi = bv1*bit1;

!   T3.Set(i,3,Qgi);
!   T3.Set(i,2,bit1);
!   T3.Set(i,1,i);
!   ik = ik + 2;
!  }
! }

 !printf('pop2 = %.2f',pop2);
 !input(str,'What on earth???');
 for(nam=2;nam<=pop2;nam=nam+1)   ! construct rest of popl
 {
   ClearOutput();
   gencheck = 0;
   par1i = Random();                ! Pick two different parents at random
   par1i = ceil(thr*par1i);
   par2i = Random();
   par2i = ceil(thr*par2i);
   rdom = Random();
   !printf('rdom = %.2f',rdom);
   if(rdom < 0.15)
   {
    ik = 2;
    for(i=1; i<=ts ;i=i+1)     ! reconstruct an offspring
    {
     rdom2 = Random();
     if(rdom2 > 0.5)
     {
      mask.Set(i,1);
      msk = 1;
     }
     else
     {
      mask.Set(i,0);
      msk = 0;
     }
     par1 = Tbest.Get(par1i,ik);
     par2 = Tbest.Get(par2i,ik);
     ofsp = (msk*par1) + ((1-msk)*par2);
     Tchk.Set(zek,ik,ofsp);
```

```
  Tchk.Set(zek,ik-1,i);
  !if(zek=11)
  !{
  ! printf('Something fishy here....');
  ! input(str,'What on earth???');
  !}
  ik = ik + 2;
 }    ! end for(i=1....

 !printf('par1i = %.2f, par2i = %.2f ',par1i,par2i);
 !input(str,'What on earth???');

 }    ! end if(rdom...
else
{
 ik = 2;
 for(i=1; i<=ts ;i=i+1)       ! reconstruct an offspring
 {
  rr1 = Random();
  rr = -0.25 + (1.5*rr1);
  par1 = Tbest.Get(par1i,ik);
  par2 = Tbest.Get(par2i,ik);
  ofsp = (rr*par1) + ((1-rr)*par2);
  ofsp = round(ofsp);
  Tchk.Set(zek,ik,ofsp);
  !printf('rr1 = %.2f, par1 = %.2f, par2 = %.2f, ',rr1,par1,par2);
  !printf('rr = %.2f & child = %.2f',rr,ofsp);
  Tchk.Set(zek,ik-1,i);
  ik = ik + 2;
 }
 !input(str,'What on earth???');
}

  bb1i = Tchk.Get(zek,2);      ! Get current generation
  bb2i = Tchk.Get(zek,4);
  bb3i = Tchk.Get(zek,6);
  bb4i = Tchk.Get(zek,8);
  bb5i = Tchk.Get(zek,10);
  bb6i = Tchk.Get(zek,12);
  bb7i = Tchk.Get(zek,14);
  bb8i = Tchk.Get(zek,16);
  bb9i = Tchk.Get(zek,18);
  bb10i = Tchk.Get(zek,20);
  bb11i = Tchk.Get(zek,22);
  bb12i = Tchk.Get(zek,24);

   ! Checks against previous pop if gen exists
  for(zz=2;zz<size;zz=zz+1)
   {
    bb1 = Tbest.Get(zz,2);
    bb2 = Tbest.Get(zz,4);
    bb3 = Tbest.Get(zz,6);
    bb4 = Tbest.Get(zz,8);
    bb5 = Tbest.Get(zz,10);
    bb6 = Tbest.Get(zz,12);
    bb7 = Tbest.Get(zz,14);
    bb8 = Tbest.Get(zz,16);
    bb9 = Tbest.Get(zz,18);
    bb10 = Tbest.Get(zz,20);
    bb11 = Tbest.Get(zz,22);
    bb12 = Tbest.Get(zz,24);

    if({bb1i=bb1}.and.{bb2i=bb2}.and.{bb3i=bb3}.and.{bb4i=bb4}
       .and.{bb5i=bb5}.and.{bb6i=bb6}.and.{bb7i=bb7}.and.{bb8i=bb8}
       .and.{bb9i=bb9}.and.{bb10i=bb10}.and.{bb11i=bb11}.and.{bb12i=bb12})
     {
       gencheck = 1;
```

```
      Qgt    = Tbest.Get(zz,25);      ! Invoke the calculated values already
      Elosst = Tbest.Get(zz,26);
      Vmaxt  = Tbest.Get(zz,27);
      vv1    = Tbest.Get(zz,28);
      Tchk.Set(zek,25,Qgt);         ! Write invoked values into generation
      Tchk.Set(zek,26,Elosst);
      Tchk.Set(zek,27,Vmaxt);
      Tchk.Set(zek,28,vv1);
     }
    }
        ! Checks against reconstructed pop if gen exists

 if(gencheck = 0)
 {
  for(zz=1;zz<zek;zz=zz+1)
  {
   bb1 = Tchk.Get(zz,2);
   bb2 = Tchk.Get(zz,4);
   bb3 = Tchk.Get(zz,6);
   bb4 = Tchk.Get(zz,8);
   bb5 = Tchk.Get(zz,10);
   bb6 = Tchk.Get(zz,12);
   bb7 = Tchk.Get(zz,14);
   bb8 = Tchk.Get(zz,16);
   bb9 = Tchk.Get(zz,18);
   bb10 = Tchk.Get(zz,20);
   bb11 = Tchk.Get(zz,22);
   bb12 = Tchk.Get(zz,24);

   if({bb1i=bb1}.and.{bb2i=bb2}.and.{bb3i=bb3}.and.{bb4i=bb4}
      .and.{bb5i=bb5}.and.{bb6i=bb6}.and.{bb7i=bb7}.and.{bb8i=bb8}
      .and.{bb9i=bb9}.and.{bb10i=bb10}.and.{bb11i=bb11}.and.{bb12i=bb12})
     {
      !input(str,'What on earth???');
      gencheck = 1;
      Qgt    = Tchk.Get(zz,25);      ! Invoke the calculated values already
      Elosst = Tchk.Get(zz,26);
      Vmaxt  = Tchk.Get(zz,27);
      vv1    = Tchk.Get(zz,28);
      Tchk.Set(zek,25,Qgt);         ! Write invoked values into generation
      Tchk.Set(zek,26,Elosst);
      Tchk.Set(zek,27,Vmaxt);
      Tchk.Set(zek,28,vv1);
     }
  }
 }

if(gencheck = 0)       ! Check if soln hasn't been located before
{
  cpp = allcaps.First();
  termi = B.First();
  !capt = cpp:r:bus1:r:cBusBar:loc_name;
  tnj = 2;
  sumq = 0;
  for(zz=1; zz<=ts; zz=zz+1) ! Implement the random generation
  {
   termii = termi:loc_name;               ! Name of busi
   while(cpp)
   {
     capt = cpp:r:bus1:r:cBusBar:loc_name; !Invoke bus connected to cap
     ntu = strcmp(termii,capt);
     if(ntu = 0)                   ! If capi is connected to busi
      {
       cpp:outserv = 0;         ! Ensure cap is in service
       !Qgi = T.Get(zz,4);
       !if(zz=1)
       !{
```

```
       Qgi1 = Tchk.Get(zek,tnj);
       !Qgi2 = Tchk.Get(zek,3);
       Qgi = bv1*Qgi1;
       cpp:qtotn = Qgi;         ! Tap the capacitor to Qgi
      !}
      !else
      !{
       !Qgi1 = Tchk.Get(zek,zz+3);
       !Qgi2 = Tchk.Get(zek,zz+4);
       !Qgi = (2*Qgi1) + Qgi2;
       !cpp:qtotn = Qgi;
      !}
      !input(str,'What on earth???');
      sumq = sumq + Qgi;

      if(Qgi = 0)
      {
       cpp:outserv = 1;        ! If Qgi = 0, put cap out of service to ensure load flow solves
      }
     break;
     }
     !printf('Cap %.0f tapped to %.0f MVar',zz,Qgi);
     cpp = allcaps.Next();         ! Invoke cap-i

   }
   termi = B.Next();
   tnj = tnj + 2;
  } ! for(zz=1.....
  !input(str,'What on earth???');
  !Qmin = sumq;
  Losses.Execute(B,Eloss,hr,Vmaxa,viol);   ! Execute losses script with bus index & object bus
  Tchk.Set(zek,26,Eloss);             ! Keep track of Eloss per gen
  Tchk.Set(zek,27,Vmaxa);
  Tchk.Set(zek,28,viol);
  !input(str,'What on earth???');
  !R.WriteDraw();
 if(viol = 0)          ! Only consider if volt limits are not
 {                     ! violated
  if(Eloss < minl)
  {
    tims = GetTime(3);
    tim = (tims-tim1)/60;   ! compute time taken to locate soln
    !ctp = ctp + 1;
    minl = Eloss;

    T2.Set(cnt1,1,0);          ! clear old values
    T2.Set(cnt1,2,0);
    T2.Set(cnt1,3,0);          ! Reset Eloss
    T2.Set(cnt1,4,0);          ! Reset Vmaxpu
    T2.Set(cnt1,5,0);

    T2.Set(cnt1,1,cnt1);          ! Store run i
    T2.Set(cnt1,2,sumq);        ! Store total Qgen [MVAr]
    T2.Set(cnt1,3,Eloss);       ! Store Eloss i
    T2.Set(cnt1,4,Vmaxa);
    T2.Set(cnt1,5,tim);

    !Eltmp1 = Eloss;          ! Eltmp1 is used to plot fitness
    !R.WriteDraw();
    cnt1 = cnt1 + 1;

    ik = 2;                   ! Keep track of best generation
    for(i=1; i<=ts ;i=i+1)
     {
       T3.Set(i,3,0);             !clear old values
       T3.Set(i,2,0);
       T3.Set(i,1,0);
```

```
            bit1 = Tchk.Get(zek,ik);   !get stored bits for best gen
            Qgi = bv1*bit1;

            T3.Set(i,3,Qgi);
            T3.Set(i,2,bit1);
            T3.Set(i,1,i);
            ik = ik + 2;
          }
         }
         !zek = zek + 1;
        }

       !else               ! If viol = 1, adjust pop size
       !{
       ! pop2 = pop2 + 1;
       ! juy = juy + 1;
       ! Chk2.Set(juy,2,pop2);
       ! Chk2.Set(juy,1,nam);
       !}

       }      !end if(gencheck....

      r1 = Random();
      r1 = 1 + floor(ts*r1);

      if(r1 > ts){ r1 = ts; }

      if(nam < (popsize+1)/2)            ! Mutation
      {
        rnd4 = Random();
        a1 = Tchk.Get(zek,(r1*2));
        a2 = a1 + (delta*(rnd4/1.1));
        if(a2>1){ a2 = 1; }             ! This bit is my modification as this
        else    { a2 = 0; }             ! Opts problem is discrete in nature
        Tchk.Set(zek,(r1*2),a2);
      }

      else
      {
        rnd4 = Random();
        a1 = Tchk.Get(zek,(r1*2));
        a2 = a1 + (delta*1.1*rnd4);
        if(a2>1){ a2 = 1; }             ! This bit is my modification as this
        else    { a2 = 0; }             ! Opts problem is discrete in nature
        Tchk.Set(zek,(r1*2),a2);
      }
     !input(str,'What on earth???');
     !boy = boy + 1;
     zek = zek + 1;
    }   !end for(nam....

  }   ! end of for(gen=1;gen<=maxgen.....

  ctp = ctp + 1;
 !input(str,'What on earth???');
  tmm = GetTime(3);
  printf('The time taken = %.2f mins',(tmm-tim1)/60);
  printf('The minimum loss = %.2f kWh',Absminl);
  !input(str,'What on earth???');
  ClearOutput();
 !}
 !printf('Still computing.....Ctp = %.0f',ctp);
}
  tmm = GetTime(3);

  !printf('The time taken = %.2f secs',tmm-tim1);
```

```
            printf('The time taken = %.2f mins',(tmm-tim1)/60);
            !printf('The minimum loss = %.2f kWh',minl);
            !input(str,'What on earth???');

          for(i=1; i<=ts ;i=i+1)
          {
           sam = T3.Get(i,3);
           if(sam > 0)
           {
             Optbus = B.Obj(i-1);
             printf('Optimal bus%.0f is %s',i,Optbus:loc_name);
           }
          }

   }
   else
   {
    printf('Specify at least two buses to connect cap!');
   }

    qq1 = T2.Get(cnt1-1,1);
    qq2 = T2.Get(cnt1-1,2);
    qq3 = T2.Get(cnt1-1,3);
    qq4 = T2.Get(cnt1-1,4);
    qq5 = T2.Get(cnt1-1,5);

    T2tr.Set(bru,1,qq1);
    T2tr.Set(bru,2,qq2);
    T2tr.Set(bru,3,qq3);
    T2tr.Set(bru,4,qq4);
    T2tr.Set(bru,5,qq5);

}
```

DPL PBIL code:

```
int ts,gen,maxgen,maxrun,i,j,popsize,nvars,hr,size,k,x,y,val,u,ind,run,cnt1;
double w1,w2,b,bb,v,v1,thr,fl,fh,delta,tmpb1,tmpb2,tmpL1,tmpL2,tmphr1,tmphr2,tmpc1,tmpc2;
double L1,L2,tim,tim1,tim2,mask,a1,a2,a3,a4,tims,sam;
double r1,rnd1,rnd2,rnd3,rnd4,rnd5,par1i,par2i,prn,caps,capi,Lsi,hri,chld,chld2,chld3,rr,rr1;
double Optbusi,minL,Lbusi,hri2,bw,pv,rndn,rndn1,rndn2,rndn3,yek,c1,c2,cr1,cr2,cc1,cc2,tmm;
int str,q,callloss,strc,sc,busi,mara,wel,tsv,prec,sm,Chksum,sumtot,vcheck,kk,ki,sumi,si,ct;
string nm,str1,str2,capt,termii;
set Bf,B,Bcheck,allcaps;
int zz,cnt,ii,ntu,zek,gencheck,viol;
int bb1,bb2,bb3,bb4,bb5,bb6,bb7,bb8,bb9,bb10,bb11,bb12,bb13,bb14,ib;
int bb15,bb16,bb17,bb18,bb19,bb20,bb21,bb22,bb23,bb24;
int bb1i,bb2i,bb3i,bb4i,bb5i,bb6i,bb7i,bb8i,bb9i,bb10i,bb11i,bb12i,bb13i,bb14i;
int bb15i,bb16i,bb17i,bb18i,bb19i,bb20i,bb21i,bb22i,bb23i,bb24i,zkk,tnj,bru;
object t,t1,bus,Optbus,userset,zm,cpp,termi;
double Qgi,Qgi1,Qgi2,bit1,bit2,bit3,bv1,bv2,bv3,sumq,minl,Vmaxa,Qmin,pv1,pv2,pv3,Eli;
double Qgt, Elosst, Vmaxt, vioL1, vioL2,vv1,vv2,qq1,qq2,qq3,qq4,qq5,yob;

yob = 17730;

for(bru=1;bru<=24;bru=bru+1)
{

str = 0;

tim1 = GetTime(3);
userset = GetCaseObject('SetUser');
popsize = 50;
maxgen = 1;
maxrun = 2;
thr = round(popsize*15/100);          ! 15% selection threshold
size = popsize+1;

fl = 0;
fh = 0;
ts = 0;
ct = 0;

minl = 100000;
Vmaxa = 1.1;
Qmin = 100;
ctp = 0;
Eloss = 25000;

 Bf = Terms.Get();
 allcaps = capss.Get();

 t = Bf.First();

 while(t)
  {

   if(t:e:iUsage = 'Busbar')      ! Only consider busbars not terminals
    {
     if(t:e:uknom = 88)           ! Only optimise at specified kV, e.g.44kV
      {
       B.Add(t);
       ts = ts + 1;               ! Determine how many busbars the grid has
       !printf('Found %s',t:loc_name);
      }
    }
   t = Bf.Next();
  }

 !printf('There are %.0f buses',ts);
```

```
!input(str,'What on earth???');
pv = 0.5;
cnt1 = 1;
prec = 7;
ClearOutput();                          ! Clear output window!
Chksum = 0;

bv1 = V.Get(1);
!v2 = V.Get(2);
!bv3 = V.Get(3);
zkk = ts*1;
zkk = pow(2,zkk);
!zkk = 10;

if(ts > 1)
 {
   !for(run=1; run<=maxrun; run=run+1)    ! Performs multiple runs
   !{
        printf('Preparation of population in progress....');
        !input(str,'What on earth???');
         for(i=1; i<=ts ;i=i+1)           ! Initialise the probability vectors
         {
          PV.Set(i,1,0.5);
         }

        cnt = 1;
        !while(cnt <= 2)
        while(minl > yob)
        {
         zek = 1;
         Bcheck.Clear();

         !while(zek <= zkk)
         !{
         for(ii=1; ii<=100; ii=ii+1)
         {
           ClearOutput();
           sumq = 0;
           gencheck = 0;                      ! Reset gen soln check
           ib = 2;
           for(i=1; i<=ts ;i=i+1)              ! T = random(nvars,pop)
           {                                 ! Initialise the population size
            T.Set(i,1,i);              ! Store bus position

            Tchk.Set(zek,ib-1,i);      ! Store bus position
            !if(i = 1)
            !{ Tchk.Set(zek,1,1); }
            !else
            !{ Tchk.Set(zek,i+2,i); }
         !   v = 0;
         !   for(wel=1; wel<=prec;wel=wel+1)        ! Initialise bit weigths
         !     {
         !       bw = pow(2,(prec-wel))/pow(2,prec);
         !       rndn = Random();
         !       if (rndn < pv)                     ! if random num < prob vector = 0.5
         !       { tsv = 1; }
         !       else
         !       { tsv = 0; }
         !       v = v + (bw*tsv);
         !
         !       Rands.Set(wel,1,bw);                ! Store bw/ bit width
         !       Rands.Set(wel,2,pv);               ! Store probability vector = 0.5
         !       Rands.Set(wel,3,rndn);              ! Store random number
         !       Rands.Set(wel,4,tsv);              ! Store bit
         !     }

              rndn1 = Random();
```

```
    pv1 = PV.Get(i,1);

    T.Set(i,2,0);      ! Reset previous values
    T.Set(i,3,0);

    if(rndn1 < pv1)
    {
      T.Set(i,2,1);
      Tchk.Set(zek,ib,1);
     }
    else
    {
      T.Set(i,2,0);
      Tchk.Set(zek,ib,0);
     }

   bit1 = T.Get(i,2);

   Qgi = bit1*bv1;
   T.Set(i,3,Qgi);              ! Store the equivalent total cap generation

   sumq = sumq + Qgi;
   ib = ib + 2;
  }

  Tchk.Set(zek,25,sumq);

!input(str,'What on earth???');

      bb1i = Tchk.Get(zek,2);      ! Get current generation
      bb2i = Tchk.Get(zek,4);
      bb3i = Tchk.Get(zek,6);
      bb4i = Tchk.Get(zek,8);
      bb5i = Tchk.Get(zek,10);
      bb6i = Tchk.Get(zek,12);
      bb7i = Tchk.Get(zek,14);
      bb8i = Tchk.Get(zek,16);
      bb9i = Tchk.Get(zek,18);
      bb10i = Tchk.Get(zek,20);
      bb11i = Tchk.Get(zek,22);
      bb12i = Tchk.Get(zek,24);

   if(zek > 1)
   {
     for(zz=1;zz<zek;zz=zz+1)
     {
      bb1 = Tchk.Get(zz,2);
      bb2 = Tchk.Get(zz,4);
      bb3 = Tchk.Get(zz,6);
      bb4 = Tchk.Get(zz,8);
      bb5 = Tchk.Get(zz,10);
      bb6 = Tchk.Get(zz,12);
      bb7 = Tchk.Get(zz,14);
      bb8 = Tchk.Get(zz,16);
      bb9 = Tchk.Get(zz,18);
      bb10 = Tchk.Get(zz,20);
      bb11 = Tchk.Get(zz,22);
      bb12 = Tchk.Get(zz,24);

      if({bb1i=bb1}.and.{bb2i=bb2}.and.{bb3i=bb3}.and.{bb4i=bb4}
          .and.{bb5i=bb5}.and.{bb6i=bb6}.and.{bb7i=bb7}.and.{bb8i=bb8}
          .and.{bb9i=bb9}.and.{bb10i=bb10}.and.{bb11i=bb11}.and.{bb12i=bb12})
        {
        !input(str,'What on earth???');
        Qgt    = Tchk.Get(zz,25);        ! Invoke the calculated values already
        Elosst = Tchk.Get(zz,26);
        Vmaxt  = Tchk.Get(zz,27);
    vv1    = Tchk.Get(zz,28);
    Tchk.Set(zek,25,Qgt);            ! Write invoked values into generation
    Tchk.Set(zek,26,Elosst);
    Tchk.Set(zek,27,Vmaxt);
    Tchk.Set(zek,28,vv1);
    gencheck = 1;
   }
  }
 }


 if(gencheck = 0)       ! Check if soln hasn't been located before
 {
   cpp = allcaps.First();
   termi = B.First();
   !capt = cpp:r:bus1:r:cBusBar:loc_name;
   tnj = 2;

   for(zz=1; zz<=ts; zz=zz+1) ! Implement the random generation
   {
    termii = termi:loc_name;               ! Name of busi
    while(cpp)
    {
      capt = cpp:r:bus1:r:cBusBar:loc_name; !Invoke bus connected to cap
      ntu = strcmp(termii,capt);
      if(ntu = 0)                    ! If capi is connected to busi
       {
        cpp:outserv = 0;         ! Ensure cap is in service
        !Qgi = T.Get(zz,4);
        !if(zz=1)
        !{
         Qgi1 = Tchk.Get(zek,tnj);
         !Qgi2 = Tchk.Get(zek,3);
         Qgi = bv1*Qgi1;
         cpp:qtotn = Qgi;         ! Tap the capacitor to Qgi
        !}
        !else
        !{
         !Qgi1 = Tchk.Get(zek,zz+3);
         !Qgi2 = Tchk.Get(zek,zz+4);
         !Qgi = (2*Qgi1) + Qgi2;
         !cpp:qtotn = Qgi;
        !}
        !input(str,'What on earth???');
        if(Qgi = 0)
        {
         cpp:outserv = 1;        ! If Qgi = 0, put cap out of service to ensure load flow solves
        }
       break;
       }
       !printf('Cap %.0f tapped to %.0f MVar',zz,Qgi);
       cpp = allcaps.Next();         ! Invoke cap-i
     }
     termi = B.Next();
     tnj = tnj + 2;
    }
    !input(str,'What on earth???');
    Qmin = sumq;
    Losses.Execute(B,Eloss,hr,Vmaxa,viol);   ! Execute losses script with bus index & object bus
    Tchk.Set(zek,26,Eloss);            ! Keep track of Eloss per gen
    Tchk.Set(zek,27,Vmaxa);
    Tchk.Set(zek,28,viol);

     if(viol = 0)           ! Only consider if volt limits are not
     {                      ! violated
       if(Eloss < minl)
       {
        R.WriteDraw();
        tims = GetTime(3);
        tim = (tims-tim1)/60;   ! compute time taken to locate soln
        ctp = ctp + 1;
        T2.Set(ctp,1,0);
        T2.Set(ctp,2,0);
        T2.Set(ctp,3,0);          ! Reset Eloss
        T2.Set(ctp,4,0);          ! Reset Vmaxpu
        T2.Set(ctp,5,0);

        T2.Set(ctp,1,ctp);          ! Store run i
        T2.Set(ctp,2,sumq);          ! Store total Qgen [MVAr]
        T2.Set(ctp,3,Eloss);         ! Store Eloss i
        T2.Set(ctp,4,Vmaxa);
        T2.Set(ctp,5,tim);

        minl = Eloss;         ! Keep track on minloss

        if(minl < yob)         ! Once solution is located, exit the for loop
        {
         ii = 100;
        }

        for(i=1; i<=ts ;i=i+1)
        {
         Qgi = T.Get(i,3);
         bit1 = T.Get(i,2);
         !bit2 = T.Get(i,2);
         !bit3 = T.Get(i,2);

         !T3.Set(i,5,0);   ! Reset contents of T3
         !T3.Set(i,4,0);
         T3.Set(i,3,0);
         T3.Set(i,2,0);
         T3.Set(i,1,0);

         T3.Set(i,3,Qgi);
         T3.Set(i,2,bit1);
         !T3.Set(i,2,bit2);
         !T3.Set(i,2,bit3);
         T3.Set(i,1,i);
        }
        !input(str,'What on earth???');
       }
     }
      !zek = zek + 1;
   }       !end if(gencheck....

   zek = zek + 1;
   !printf('The tot losses = %.2f',Eloss);
   !input(str,'What on earth???');

      !if(callloss = 1)
      !{

        ! Bcheck.Add(bus);
        ! Losses.Execute(b,bus,B,Eloss,hr);   ! Execute losses script with bus index & object bus

        ! R.WriteDraw();                  ! Plot convergence rate

        ! printf('%.0f',cnt);

        ! T.Set(cnt,2,Eloss);             ! Store fitness in the matrix
        ! T.Set(cnt,3,hr);              ! Store hr in the matrix
        ! cnt1 = cnt1 + 1;
!zek = zek + 1;
```

```
           !}
          } !end for(ii...
           cnt = cnt + 1;
           !input(str,'What on earth???');
           ClearOutput();
           !}
           !printf('Still computing.....Ctp = %.0f',ctp);
           for(i=1; i<=ts ;i=i+1)           ! Initialise the probability vectors
            {
             !pv3 = PV.Get(i,1);             ! Invoke prob vector
             !pv2 = PV.Get(i,1);
             pv1 = PV.Get(i,1);

             bit1 = T3.Get(i,2);            ! Invoke best vector bits
             !bit2 = T3.Get(i,2);
             !bit3 = T3.Get(i,2);

             pv1 = (0.9*pv1) + (0.1*bit1);  ! Update probability vector
             !pv2 = (0.9*pv2) + (0.1*bit2);
             !pv3 = (0.9*pv3) + (0.1*bit3);

             pv1 = pv1 - (0.005*(pv1-0.5));  ! Relax to maintain diversity
             !pv2 = pv2 - (0.005*(pv2-0.5));
             !pv3 = pv3 - (0.005*(pv3-0.5));

             !PV.Set(i,1,pv3);
             !PV.Set(i,1,pv2);
             PV.Set(i,1,pv1);
            }

        }
            tmm = GetTime(3);

            !printf('The time taken = %.2f secs',tmm-tim1);
            printf('The time taken = %.2f mins',(tmm-tim1)/60);
            printf('The minimum loss = %.2f kWh',minl);
            !input(str,'What on earth???');

        for(i=1; i<=ts ;i=i+1)
        {
         sam = T3.Get(i,3);
         if(sam > 0)
         {
           Optbus = B.Obj(i-1);
           printf('Optimal bus%.0f is %s',i,Optbus:loc_name);
         }
        }

        !minL = T.Get(1,2);
        !printf('The Optimal bus is %s & Min Eloss = %.2f kWh',Optbus:loc_name,minL);

       ! printf('cnt1 = %.0f',cnt1);
        !printf('Min Losses = %.2f',minL);
       ! tim2 = GetTime(3);
       ! tim = (tim2-tim1)/60;
       ! printf('Total elapsed time = %.3f mins',tim);
  !}

}
else
{
 printf('Specify at least two buses to connect cap!');
}

 qq1 = T2.Get(ctp,1);
 qq2 = T2.Get(ctp,2);
 qq3 = T2.Get(ctp,3);
```

```
  qq4 = T2.Get(ctp,4);
  qq5 = T2.Get(ctp,5);

  T2tr.Set(bru,1,qq1);
  T2tr.Set(bru,2,qq2);
  T2tr.Set(bru,3,qq3);
  T2tr.Set(bru,4,qq4);
  T2tr.Set(bru,5,qq5);

}
```

DPL PPBIL code:

```
int ts,gen,maxgen,maxrun,i,j,popsize,nvars,hr,size,k,x,y,val,u,ind,run,cnt1;
double w1,w2,b,bb,v,v1,thr,fl,fh,delta,tmpb1,tmpb2,tmpL1,tmpL2,tmphr1,tmphr2,tmpc1,tmpc2;
double L1,L2,tim,tim1,tim2,mask,a1,a2,a3,a4,tims,sam;
double r1,rnd1,rnd2,rnd3,rnd4,rnd5,par1i,par2i,prn,caps,capi,Lsi,hri,chld,chld2,chld3,rr,rr1;
double Optbusi,minL,Lbusi,hri2,bw,pv,rndn,rndn1,rndn2,rndn3,yek,c1,c2,cr1,cr2,cc1,cc2,tmm;
int str,q,callloss,strc,sc,busi,mara,wel,tsv,prec,sm,Chksum,sumtot,vcheck,kk,ki,sumi,si,ct;
string nm,str1,str2,capt,termii;
set Bf,B,Bcheck,allcaps;
int zz,cnt,ii,ntu,zek,gencheck,viol;
int bb1,bb2,bb3,bb4,bb5,bb6,bb7,bb8,bb9,bb10,bb11,bb12,bb13,bb14,ib;
int bb15,bb16,bb17,bb18,bb19,bb20,bb21,bb22,bb23,bb24;
int bb1i,bb2i,bb3i,bb4i,bb5i,bb6i,bb7i,bb8i,bb9i,bb10i,bb11i,bb12i,bb13i,bb14i;
int bb15i,bb16i,bb17i,bb18i,bb19i,bb20i,bb21i,bb22i,bb23i,bb24i,zkk,tnj;
object t,t1,bus,Optbus,userset,zm,cpp,termi;
double Qgi,Qgi1,Qgi2,bit1,bit2,bit3,bv1,bv2,bv3,sumq,minl,Vmaxa,Qmin,pv1,pv2,pv3,Eli,lr;
double Qgt, Elosst, Vmaxt, vioL1, vioL2,vv1,vv2,pvr,pva,pvb,randmask,prov,qq1,qq2,qq3,qq4,qq5;
double bestsol,yob;
int bestsolcheck,psh,jj,bru;

yob = 19219;
for(bru=1;bru<=23;bru=bru+1)
{

str = 0;
lr = 0.1;

tim1 = GetTime(3);
userset = GetCaseObject('SetUser');
popsize = 10;
maxrun = 1;
maxgen = 10;

ts = 0;
ct = 0;
minl = 100000;
Vmaxa = 1.1;
Qmin = 100;
ctp = 0;
Eloss = 25000;
pv = 0.5;
cnt1 = 1;
prec = 7;
Chksum = 0;
bv1 = V.Get(1);
Bf = Terms.Get();
allcaps = capss.Get();
t = Bf.First();

ClearOutput();                    ! Clear output window!
 while(t)
  {

   if(t:e:iUsage = 'Busbar')      ! Only consider busbars not terminals
    {
     if(t:e:uknom = 88)           ! Only optimise at specified kV, e.g.44kV
      {
       B.Add(t);
       ts = ts + 1;               ! Determine how many busbars the grid has
       !printf('Found %s',t:loc_name);
      }
     }
   t = Bf.Next();
  }
```

```
if(ts > 1)
 {
    for(ii=1; ii<=(2*popsize); ii=ii+1)
     {
      for(i=1; i<=ts; i=i+1)
      {
       PM.Set(ii,i,0.5);                    ! Initialise Prob Matrix to 0.5
      }
     }

    for(run=1; run<=maxrun; run=run+1)     ! Performs multiple runs
    {
      cnt = 1;
      while(cnt <= maxgen)
      !while(minl > 18969)
      {
       zek = 1;
       Bcheck.Clear();
       !while(zek <= zkk)
       !{
       !bestsolcheck = 0;
       for(ii=1; ii<=popsize; ii=ii+1)
       {
         ClearOutput();
         sumq = 0;
         gencheck = 0;                     ! Reset gen soln check
         for(i=1; i<=ts ;i=i+1)            ! Create trial solutions
         {
           pva = PM.Get(ii,i);
           pvb = PM.Get(ii+popsize,i);

           pvr = Random();
             if(pvr > 0.5)
             { randmask = 1; }
             else
             { randmask = 0; }

           prov = (randmask*pva) + ((1-randmask)*pvb);
           P.Set(i,1,prov);                          ! Store probability vector bit
           rndn1 = Random();
           T.Set(i,1,i);       ! Store bus position
           T.Set(i,2,0);       ! Reset previous values
           T.Set(i,3,0);
             if(prov > rndn1)
             {
               T.Set(i,2,1);
               Tchk.Set(zek,i,1);
              }
             else
             {
               T.Set(i,2,0);
               Tchk.Set(zek,i,0);
              }

           bit1 = T.Get(i,2);
           Qgi = bit1*bv1;
           T.Set(i,3,Qgi);               ! Store the equivalent total cap generation
           sumq = sumq + Qgi;
          }

            Tchk.Set(zek,11,sumq);

            bb1i = Tchk.Get(zek,1);       ! Get current generation
            bb2i = Tchk.Get(zek,2);
            bb3i = Tchk.Get(zek,3);
            bb4i = Tchk.Get(zek,4);
            bb5i = Tchk.Get(zek,5);
 bb6i = Tchk.Get(zek,6);
 bb7i = Tchk.Get(zek,7);
 bb8i = Tchk.Get(zek,8);
 bb9i = Tchk.Get(zek,9);
 bb10i = Tchk.Get(zek,10);

if(zek > 1)
{
 for(zz=1;zz<zek;zz=zz+1)
 {
  bb1 = Tchk.Get(zz,1);
  bb2 = Tchk.Get(zz,2);
  bb3 = Tchk.Get(zz,3);
  bb4 = Tchk.Get(zz,4);
  bb5 = Tchk.Get(zz,5);
  bb6 = Tchk.Get(zz,6);
  bb7 = Tchk.Get(zz,7);
  bb8 = Tchk.Get(zz,8);
  bb9 = Tchk.Get(zz,9);
  bb10 = Tchk.Get(zz,10);

  if({bb1i=bb1}.and.{bb2i=bb2}.and.{bb3i=bb3}.and.{bb4i=bb4}
      .and.{bb5i=bb5}.and.{bb6i=bb6}.and.{bb7i=bb7}.and.{bb8i=bb8}
      .and.{bb9i=bb9}.and.{bb10i=bb10})
   {
    !input(str,'What on earth???');
    Qgt    = Tchk.Get(zz,11);        ! Invoke the calculated values already
    Elosst = Tchk.Get(zz,12);
    Vmaxt  = Tchk.Get(zz,13);
    vv1    = Tchk.Get(zz,14);
    Tchk.Set(zek,11,Qgt);           ! Write invoked values into generation
    Tchk.Set(zek,12,Elosst);
    Tchk.Set(zek,13,Vmaxt);
    Tchk.Set(zek,14,vv1);
    gencheck = 1;
    !input(str,'What on earth???');
   }
 }
}


 if(gencheck = 0)       ! Check if soln hasn't been located before
 {
   cpp = allcaps.First();
   termi = B.First();
   !capt = cpp:r:bus1:r:cBusBar:loc_name;
   tnj = 2;

   for(zz=1; zz<=ts; zz=zz+1) ! Implement the random generation
   {
    termii = termi:loc_name;                ! Name of busi
    while(cpp)
    {
      capt = cpp:r:bus1:r:cBusBar:loc_name; !Invoke bus connected to cap
      ntu = strcmp(termii,capt);
      if(ntu = 0)                    ! If capi is connected to busi
       {
        cpp:outserv = 0;         ! Ensure cap is in service
        !Qgi = T.Get(zz,4);
        !if(zz=1)
        !{
         Qgi1 = Tchk.Get(zek,zz);
         !Qgi2 = Tchk.Get(zek,3);
         Qgi = bv1*Qgi1;
         cpp:qtotn = Qgi;         ! Tap the capacitor to Qgi
        !}
        !else
```

```
     !{
      !Qgi1 = Tchk.Get(zek,zz+3);
      !Qgi2 = Tchk.Get(zek,zz+4);
      !Qgi = (2*Qgi1) + Qgi2;
      !cpp:qtotn = Qgi;
     !}
     !input(str,'What on earth???');
     if(Qgi = 0)
     {
      cpp:outserv = 1;        ! If Qgi = 0, put cap out of service to ensure load flow solves
     }
    break;
     }
    !printf('Cap %.0f tapped to %.0f MVar',zz,Qgi);
    cpp = allcaps.Next();         ! Invoke cap-i
  }
  termi = B.Next();
  tnj = tnj + 2;
 }
 !input(str,'What on earth???');
 Qmin = sumq;
 !printf('Qmin = %.2f',Qmin);
 Losses.Execute(B,Eloss,hr,Vmaxa,viol);   ! Execute losses script with bus index & object bus
 Tchk.Set(zek,12,Eloss);             ! Keep track of Eloss per gen
 Tchk.Set(zek,13,Vmaxa);
 Tchk.Set(zek,14,viol);

  if(viol = 0)           ! Only consider if volt limits are not
  {                     ! violated
    if(Eloss < minl)
    {
     R.WriteDraw();
     tims = GetTime(3);
     tim = (tims-tim1)/60;   ! compute time taken to locate soln
     ctp = ctp + 1;
     T2.Set(ctp,1,0);
     T2.Set(ctp,2,0);
     T2.Set(ctp,3,0);          ! Reset Eloss
     T2.Set(ctp,4,0);          ! Reset Vmaxpu
     T2.Set(ctp,5,0);

     T2.Set(ctp,1,ctp);          ! Store run i
     T2.Set(ctp,2,sumq);         ! Store total Qgen [MVAr]
     T2.Set(ctp,3,Eloss);        ! Store Eloss i
     T2.Set(ctp,4,Vmaxa);
     T2.Set(ctp,5,tim);

     minl = Eloss;         ! Keep track on minloss

     if(minl < yob)         ! Once solution is located, exit the for loop
     {
      cnt = maxgen;
      ii = popsize;
     }

     for(i=1; i<=ts ;i=i+1)
     {
      bestsolcheck = 1;
      Qgi = T.Get(i,3);
      bit1 = T.Get(i,2);

      T3.Set(i,3,0);
      T3.Set(i,2,0);
      T3.Set(i,1,0);

      T3.Set(i,3,Qgi);
      T3.Set(i,2,bit1);
```

```
            T3.Set(i,1,i);
           }
           !input(str,'What on earth???');
          }
        }
         !zek = zek + 1;
       }       !end if(gencheck....
       !input(str,'What on earth???');
       zek = zek + 1;

       for(i=1; i<=ts; i=i+1)
        {
         if(bestsolcheck = 0)                  ! If there's no valid solution
         {}                                    ! Don't modify the algorithm
         else
         {
          bestsol = T3.Get(i,2);               ! Invoke best vector bits
          pva = PM.Get(ii,i);                  ! Invoke popl A
          pva = ((1-lr)*pva) + (lr*bestsol);  ! Modify pop vector a
          PM.Set(ii,i,pva);                    ! Write the values back to matrix

          pvb = PM.Get(ii+popsize,i);          ! Invoke popl B
          pvb = ((1-lr)*pvb) + (lr*bestsol);  ! Modify pop vector b
          PM.Set(ii+popsize,i,pvb);            ! Write the values back to matrix
         }
        } !end for(i=1...

      } !end for(ii...

      ClearOutput();

      !for(jj=1; jj<=(2*popsize); jj=jj+1)      ! Swap prob matrices
      !{
      !  for(i=1; i<=ts; i=i+1)
      !  {

      !  }
      !}

      cnt = cnt + 1;
     }  !end while(cnt...

   } !end for(run=1...
      tmm = GetTime(3);

      !printf('The time taken = %.2f secs',tmm-tim1);
      printf('The time taken = %.2f mins',(tmm-tim1)/60);
      printf('The minimum loss = %.2f kWh',minl);
      !input(str,'What on earth???');

  for(i=1; i<=ts ;i=i+1)
  {
   sam = T3.Get(i,3);
   if(sam > 0)
   {
    Optbus = B.Obj(i-1);
    printf('Optimal bus%.0f is %s',i,Optbus:loc_name);
   }
  }

  !minL = T.Get(1,2);
  !printf('The Optimal bus is %s & Min Eloss = %.2f kWh',Optbus:loc_name,minL);

 ! printf('cnt1 = %.0f',cnt1);
  !printf('Min Losses = %.2f',minL);
 ! tim2 = GetTime(3);
 ! tim = (tim2-tim1)/60;
```

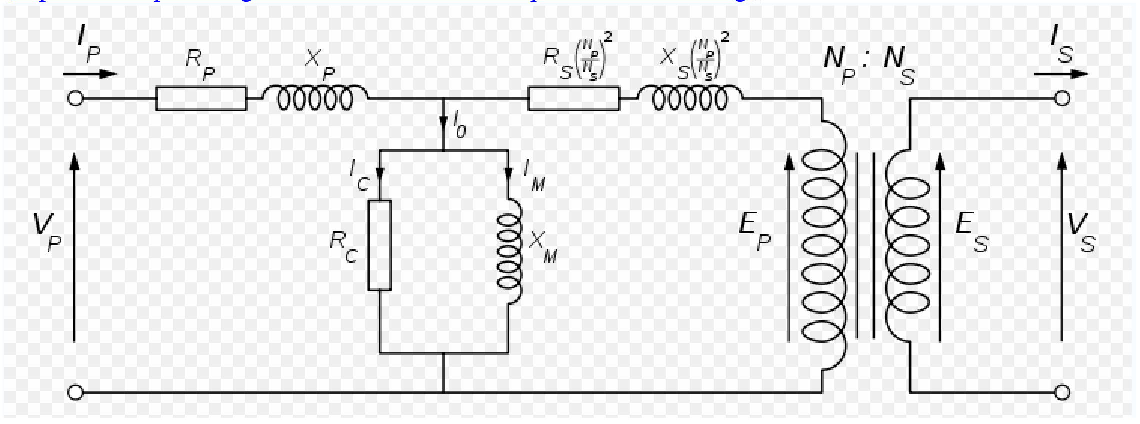
```
        ! printf('Total elapsed time = %.3f mins',tim);
    !}

  }
  else
  {
   printf('Specify at least two buses to connect cap!');
  }

   qq1 = T2.Get(ctp,1);
   qq2 = T2.Get(ctp,2);
   qq3 = T2.Get(ctp,3);
   qq4 = T2.Get(ctp,4);
   qq5 = T2.Get(ctp,5);

   T2tr.Set(bru,1,qq1);
   T2tr.Set(bru,2,qq2);
   T2tr.Set(bru,3,qq3);
   T2tr.Set(bru,4,qq4);
   T2tr.Set(bru,5,qq5);

}
 !for(mara=1; mara<=ts; mara=mara+1)
 !{
 ! zm = B.Obj(mara-1);
 ! printf('Bus %.0d = %s',mara,zm:loc_name);
 !}
```

DPL Losses code:

```
set lns,caps,tms,optterms,Tr2set,Tr2nset,Tr3set;
object ln,cap,tm,cub1,cub2,Time,optterm,trfm2,trfm2n,trfm3,bs;
double Loss1,Loss2,loadloss2,noloadloss2,totalloss2,loadloss2n,noloadloss2n,totalloss2n;
double totP,totQ,loadloss3,noloadloss3,totalloss3,dt,minLoss;
double Vmaxb,Vminb,Vmina,bsv,Vmax;
int i,hr,str;

Vmax = 1.05;
viol = 0;                 ! Checks if Max V > permissible voltage limit
i = 1;
Loss1 = 0;
Loss2 = 0;
dt = 1;                        ! This is change in time i.e. load interval period
Eloss = 0;
minLoss = 10000;
!caps = shf1.Get();              ! Get all OCP caps      (double) busind & object (bus)
lns = Lines.Get();               ! Get all lines
!cap = caps.First();
!cap = CMVar;
Tr2set  = Trfms2.Get();
Tr2nset = Trfms2n.Get();
Tr3set  = Trfms3.Get();

Time = GetCaseObject('SetTime');

Vmaxa = 0.95;          ! Make max ridiculously low to start with
Vmina = 1.1;          ! Make min quite high to start with

for(hr = 0; hr < 24 ; hr = hr + 1)
 {
  totalloss2 = 0;
  totalloss2n = 0;
  totalloss3 = 0;

  Time:hour = hr;                  ! Set time trigger to ith hour
  Ldf.Execute();
  !ResetCalculation();

   Loss2 = 0;                                    ! Reset losses
 !--------------------------------------------------------------------
   bs = buses.First();
   !Vmaxa = bs:m:u;
   !Vmina = bs:m:u;
  for(bs = buses.First(); bs; bs = buses.Next()) ! determine max & min voltages
  {
   bsv = bs:m:u;
   if(bsv > Vmaxa)
   { Vmaxa = bsv; }

   if(bsv < Vmina)
   { Vmina = bsv; }

    if(bsv > Vmax)       ! Check if any bus voltage > Vmax pu
    {
     viol = 1;
    }
  }

  LF.Set(hr+1,10,Vmaxa);
  LF.Set(hr+1,12,Vmina);

  !if(Vmaxa > Vmax)
  !{
  ! printf('Oopsie violation detected');
  ! break;
  !}

  !if(viol = 1)           ! If bus voltage > limit, make the loss worse
  !{
  ! Loss2 = 20;           !1.01*Loss1;
  ! Eloss = 1.01*Loss2;
  ! viol = 0;             ! Reset the voltage violation check
  !}
  !else
  !{
  !--------------------------------------------------------------------

   for(ln = lns.First(); ln; ln = lns.Next())
    {
     Loss2 = Loss2 + ln:c:Losses;
    }
    Loss2 = Loss2/1000;       !Convert from kW to MW

!_____________________________________________________________________________________________
________
 !2 winding after cap
    for(trfm2 = Tr2set.First(); trfm2; trfm2 = Tr2set.Next())
    {
     !loadloss2   = loadloss2 + trfm2:m:Plossld:bushv;        ! 2 winding copper losses MW
     !noloadloss2 = noloadloss2 + trfm2:m:Plossnld:bushv;     ! 2 winding no load losses MW
     !totalloss2 = totalloss2 + loadloss2 + noloadloss2;
     totalloss2 = totalloss2 + trfm2:m:Plossld:bushv + trfm2:m:Plossnld:bushv;
    }


!_____________________________________________________________________________________________
________
 !2n winding after cap
    for(trfm2n = Tr2nset.First(); trfm2n; trfm2n = Tr2nset.Next())
    {
     !loadloss2n   = loadloss2n + trfm2n:m:Plossld:bushv;      ! 2n winding copper losses MW
     !noloadloss2n = noloadloss2n + trfm2n:m:Plossnld:bushv;   ! 2n winding no load losses MW
     !totalloss2n  = totalloss2n + loadloss2n + noloadloss2n;

     if(trfm2n:e:outserv = 0)
     {
      totalloss2n = totalloss2n + trfm2n:m:Plossld:bushv + trfm2n:m:Plossnld:bushv;
     }
    }


!_____________________________________________________________________________________________
________
 !3 winding after cap
 !   for(trfm3 = Tr3set.First(); trfm3; trfm3 = Tr3set.Next())
 !   {
 !    loadloss3   = loadloss3 + trfm3:m:Plossld:bushv;        ! 3 winding copper losses MW
 !    noloadloss3 = noloadloss3 + trfm3:m:Plossnld:bushv;     ! 3 winding no load losses MW
 !    totalloss3  = totalloss3 + loadloss3 + noloadloss3;
 !   }

   !printf('With the %s cap at %s, Total system losses = %.2f kW',cap:loc_name,tm:loc_name,Loss);
   !LS.Set(i,1,tm:loc_name);
   !LS.Set(i,3,Loss);            !Store total system losses

   Loss2 = Loss2 + totalloss2 + totalloss2n + totalloss3;
   Eloss = Eloss + (Loss2*dt);

  !}
  !cub = optterm.CreateObject('StaCubic','Cubicle1');        ! Recreate cubicle to simulate
  !1MVar:bus1 = cub;                                          ! losses when cap is connected at
                                                              ! optimal terminal
  !Ldf.Execute();
  !totP = LibanonTrf1:m:P:buslv + LibanonTrf2:m:P:buslv;
  !totQ = LibanonTrf1:m:Q:buslv + LibanonTrf2:m:Q:buslv;
  totP = -1*(GoatTrf1:m:P:buslv + GoatTrf2:m:P:buslv);
  totQ = -1*(GoatTrf1:m:Q:buslv + GoatTrf2:m:Q:buslv);

  if(totQ < 0)
  { viol = 1;}

  LF.Set(hr+1,5,totP);  ! Store Total MW in matrix
  LF.Set(hr+1,6,totQ);  ! Store Total MW in matrix
  LF.Set(hr+1,7,Loss2);
  !printf('Loss2 = %.2f',Loss2);
  !printf('Optimal term is %s',optterm:loc_name);
 }
 Eloss = 1000*Eloss;                ! Convert Eloss to kWh
 Vmaxa = Vmaxa;

 !printf('Eloss = %.2f kWh', Eloss);
 !printf('Vmax = %.4f p.u.', Vmaxa);
 !printf('Vmin = %.4f p.u.', Vmina);
```

APPENDIX B: Equivalent transformer model

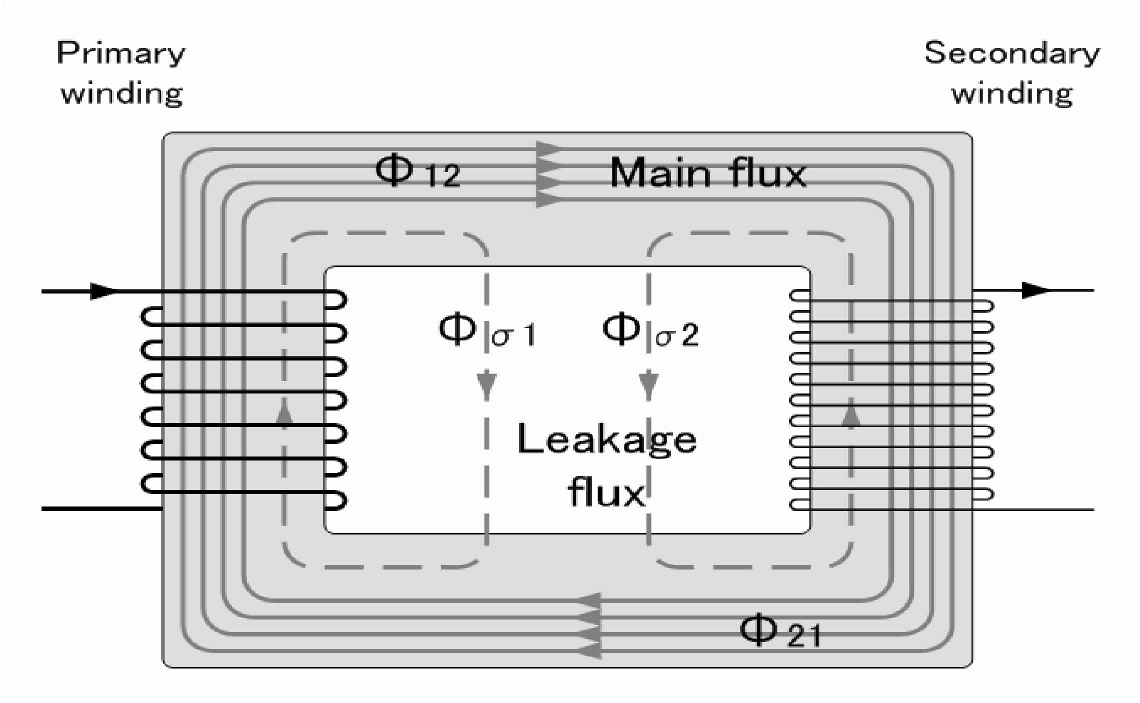
[http://en.wikipedia.org/wiki/File:Transformer_equivalent_circuit.svg]



Equivalent circuit for a single phase transformer with N_P turns in the primary winding and N_S in the secondary, showing:

- Primary winding resistance R_P
- Primary leakage reactance X_P
- Secondary winding resistance R_S, referred to the primary circuit by the turns ratio squared
- Secondary leakage reactance X_S, referred to the primary circuit by the turns ratio squared
- Equivalent core loss resistance R_C and core loss current I_C
- Magnetising reactance X_M and magnetising current I_M
- No-load current I_0
- Primary and secondary EMF E_P and E_S, developed over an ideal transformer
- Primary and secondary terminal voltages and currents V_P, I_P, V_S and I_S

http://en.wikipedia.org/wiki/File:Transformer_flux.gif



Primary winding
Secondary winding
Φ_{12}
Main flux
$\Phi_{\sigma 1}$
$\Phi_{\sigma 2}$
Leakage flux
Φ_{21}