OPTIMAL FAULT LOCATION

A Thesis

by

MAJA KNEZEV

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2007

Major Subject: Electrical Engineering

OPTIMAL FAULT LOCATION

A Thesis

by

MAJA KNEZEV

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2007

Major Subject: Electrical Engineering

ABSTRACT

Optimal Fault Location.

(December 2007)

Maja Knezev, Dipl. Ing., University of Novi Sad

Chair of Advisory Committee: Dr. Mladen Kezunovic

Basic goal of power system is to continuously provide electrical energy to the users. Like with any other system, failures in power system can occur. In those situations it is critical that correct remedial actions are applied as soon as possible after the accurate fault condition and location are detected. This thesis has been focusing on automated fault location procedure.

Different fault location algorithms, classified according to the spatial placement of physical measurements on single ended, multiple ended and sparse system-wide, are investigated. As outcome of this review, methods are listed as function of different parameters that influence their accuracy. This comparison is than used for generating procedure for optimal fault location algorithm selection. According to available data, and position of the fault with respect to the data, proposed procedure decides between different algorithms and selects an optimal one. A new approach is developed by utilizing different data structures such as binary tree and serialization in order to efficiently implement algorithm decision engine.

After accuracy of algorithms is strongly influenced by available input data, different data sources are recommended in proposed architecture such as the digital fault recorders, circuit breaker monitoring, SCADA, power system model and etc. Algorithm for determining faulted section is proposed based on the data from circuit breaker monitoring devices. This algorithm works in real time by recognizing to which sequence of events newly obtained recording belongs.

Software prototype of the proposed automated fault location analysis is developed using Java programming language. Fault location analysis is automatically triggered by appearance of new event files in a specific folder. The tests were carried out using the real life transmission system as an example.

DEDICATION

To My Parents and Zarko

ACKNOWLEDGMENTS

I would like to thank my advisor and committee chair, Dr. Kezunovic and my committee members, Dr. Singh, Dr. Khatri and Dr. Lively for their guidance and support during the course of this research.

Also, thanks are due to my friends and colleagues and the department faculty and staff for making my time at Texas A&M University a great learning experience. Special thanks to Yang Wu, Levi Portillo, Nitin Ved, Anisha Jonas, Goran Latisko and Zarko Djekic.

Finally, thanks to my parents for their love and encouragement during tough times and patience while I took my time to explore this opportunity.

TABLE OF CONTENTS

LIST OF FIGURES

Page

LIST OF TABLES

CHAPTER I

INTRODUCTION

1. Background

Basic goal of power system is to continuously provide electrical energy to the users. Like with any other system, failures in power system can occur. In those situations it is critical that remedial actions are applied as soon as possible. To apply correct remedial actions it is important that accurate fault condition and location are detected. The protection system is a part of the power system responsible for fault detection and execution of automatic remedial actions. When fault appears different devices that comprise protection system are triggered. Protection system consisting of protection relays and circuit breakers (CBs) will operate in order to de-energize faulted line. Different Intelligent Electronic Devices (IEDs) located in substations for the purpose of monitoring will be automatically triggered by the fault and will record corresponding current, voltage and status signals. Those records are later used by different user groups for fault investigation. Changes in the switching equipment status will automatically be seen in the control center by an operator who will mark fault event in a spreadsheet and inform other staff responsible for dealing with fault analysis and repair such as protection group or maintenance respectively. Protective relaying staff will be ready to analyze the fault in more details and maintenance will be ready to take any repair action as needed. Protective relaying staff will typically retrieve IED data from substations using manually initiated upload request, which may last from few minutes to few days to complete. Substantial time is needed by the protective relaying staff to complete the analysis before entering its findings into designated fields in the spreadsheet. Maintenance staff will be the last group to be involved. They will go to the filed in the case of a permanent fault and inspect and repair the damaged equipment as needed. Their report will result in the last part of the information being entered into the spreadsheet.

This thesis follows the style of *IEEE Transactions on Power Delivery*.

The operators will restore the line when everything seems to be in tact again and will close the spreadsheet event report, which then will be archived for any future uses. This process is shown in Figure 1.1.

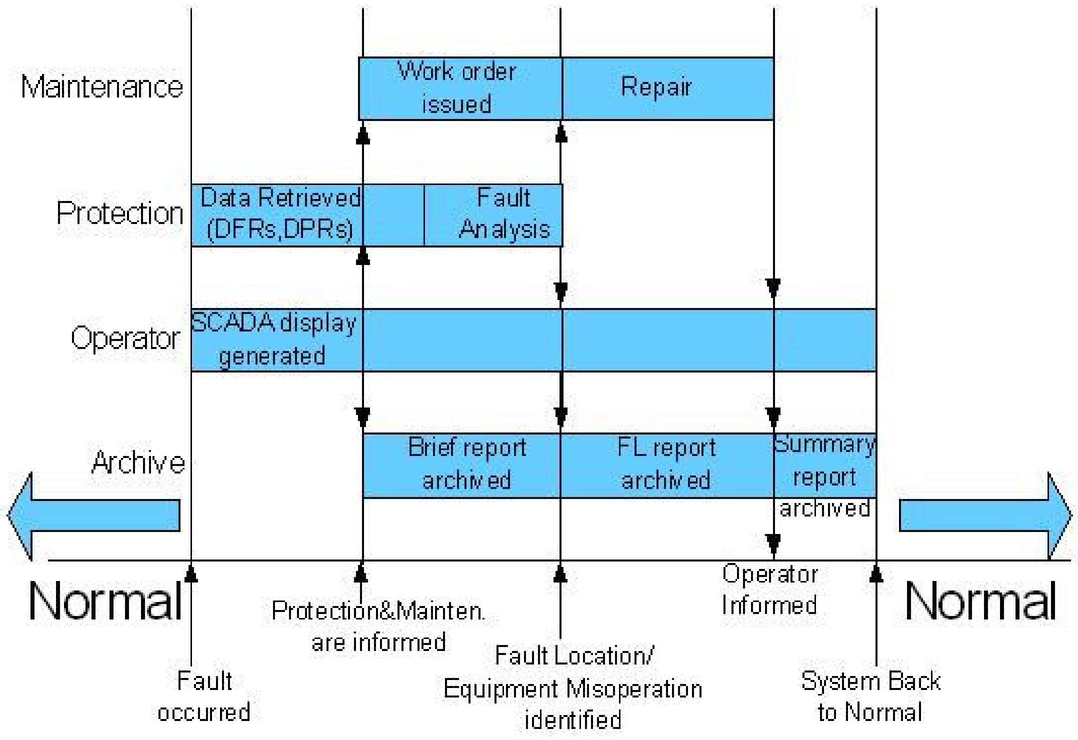


Figure 1.1: Timeline of utility personnel actions in case of permanent fault

2. Existing Approach

When a fault occurs, protection equipment initiates operation of breakers, to de-energize faulted part. This must be done before excessive currents and voltages caused by the fault inflict damage to the connected equipment. CBs have the purpose to automatically connect or disconnect different parts of the power system in order to isolate the faults and/or re-route the power flow. In order to open all circuits that supply fault current, more than one CB typically reacts. Various bus arrangements are used to minimize the number of circuits that must be opened in a case of a fault [1]. Depending on a bus arrangement and status of available breakers, different breakers will automatically react in case of different faults.

Once protective relay detects fault, it sends a trip command to corresponding breakers. CBs react and open circuits that supply fault current. In the case of a HV transmission line, after some time a breaker involved in the fault clearing assumes that fault is cleared and it will try to reclose itself automatically. Typical switching sequences occurring because of a fault present somewhere on a transmission line is shown on Figure 1.2.

Process of reclosing can be repeated a couple of times and it is initiated in order to determine whether fault, which caused opening of breaker, is still present. In the case that fault is present after a reclosing, breaker will wait for the pre-selected time out to pass and initiate reclosing again. If after selected number of attempts of reclosing, fault is still present, breaker lockout is taking place. There will be no more attempts to reclose automatically the breaker again. The length of switching sequences and time outs depend not only on the type of the breaker, but also on location of the circuit breaker and the type of protective relay connected to it, as well as the reclosing logic used in the given system.

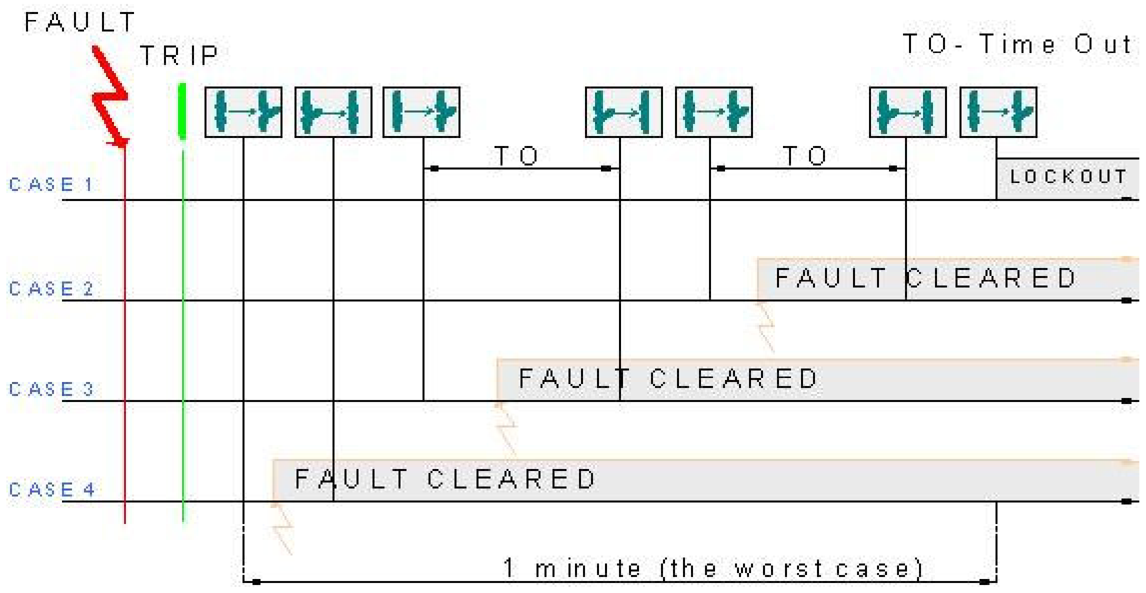


Figure 1.2: Trip and reclose sequences on a single breaker

In the case of breaker lockout the assumption taken by the operators is that fault is permanent. Special order is issued to the maintenance for the breaker to be closed back in again once an inspection of the breaker, analysis of the causes and possible repair are completed.

From the above, two types of faults are recognized: a) temporary and b) permanent. Each of these faults will initiate different actions among utility personnel.

a) In the case of a **temporary fault**, operator will notice CB status change on the corresponding one-line power system topology diagram. This information is tracked through the Supervisory Control and Data Acquisition (SCADA) system. Switching sequence of openings and reclosings of a group of circuit breakers will end with CBs that were involved in the switching case being restored. All the equipment actions are executed automatically and fault is cleared. In this case there is no need for operator action, but occurrence of the event is recorded and archived.

b) In the case of a **permanent fault**, operator will notice CB status change on the corresponding one-line topology diagram. Sequence of openings and reclosings of a group of breakers will end with CBs that were involved in clearing the fault staying open. Automatic fault clearing has disconnected faulted part from the rest of the power system, and further attempts to automatically restore the system to the original healthy state are not taken. This is called breaker LOCKOUT. Disconnected part must be restored manually after inspection and repair.

There are few features of the existing approach that should be further evaluated to indicate what possible improvements to the described process are: data availability, response time and decision quality, and personnel productivity.

Data availability

It should be noticed that different data and information are available to different user groups at different times. Operators have access to SCADA system data all the time. Protection group has to retrieve fault recordings from IEDs located in substations in order to do fault analysis. Protection group does not have access to SCADA system information and relies on operators to tell them what they have observed. Maintenance is informed by the operator when permanent fault is present, but they do not have clear instructions about possible actions before they get fault location estimation from the protection group. Once the maintenance inspects and repairs the damage, operators are informed and they can restore the system.

Response time and decision quality

Depending on correctness of available data and information entered by different user groups, fault may be cleared in varying time intervals. It is very important for an operator to have as fast response from other groups about fault event as possible in order to know when faulted line can be restored. In the case of protection group, if they need measurements from different substations to figure out fault location, actions of other groups have to be delayed until protection staff gets the required data and makes conclusions. If estimated fault location is not correct maintenance crew will have to patrol the line longer until they find the fault through visual inspection.

Personnel productivity

Separation of available data by different utility groups leads to a lot of waiting because the groups depend on each other when making final decisions. In the case that there are more fault events occurring as a result of a bad storm, protection group may be burdened by the analysis of multiple events. They have to retrieve measurements, sort them according to corresponding fault events, process them and draw correct conclusions about actions that should be taken. Maintenance crew is alarmed when the fault event is present but is not capable of knowing where exactly to go before protection group identifies fault location.

3. Definition of the Problem

Once the fault is present in the system there are few features that should be satisfied:

a) Protection equipment should take actions as soon as possible.
b) In the case of temporary fault operator should be able to doubtlessly conclude from available data whether the system restoration is done correctly.
c) In the case of permanent fault maintenance crew should obtain estimation of the fault location as soon as possible and repair the fault. Operator should get trusty information about status of faulted line after the repair and then restore the line.

From previous section several shortcomings of the existing approach can be noticed:

- Complete data is not made available to all the utility groups, which reduces the quality of their decisions and prolongs the time for the decisions to be made.
- Data retrieval is not automated which influences the time response since the groups make decisions after the data is retrieved, organized and analyzed manually.
- Fault location is typically calculated by the protection group, which influence time response of fault restoration since the working hours of the protection group are typically 8am-5pm while the faults may occur randomly at any time.
- Fault analysis executed by protection group may be based on fault location algorithm that is not suitable for all fault cases, and hence the result may be quite inaccurate in some instances.
- In the case of temporary faults operators do not have a way of confirming fault location and evaluating whether equipment reacted as expected.

With technological advancements, filed measurements taken from different locations can be synchronized. IEDs are capable of communicating between themselves. Data storage can easily be interfaced to different access points and intelligent techniques can be used for fast fault analysis. These benefits could be used to enhance the existing fault investigation process.

4. Objectives

The objective of this thesis is to investigate new architecture that looks at various input data and its disposition with the respect to a possible fault location, and implements an optimal procedure that decides which fault location algorithms is the most suitable in the case of a given fault event. To accomplish this, the proposed research study aims at the following tasks: a) classify existing fault location algorithms, b) outline and evaluate available data sources, c) develop procedure for Optimal Fault Location (OFL) algorithm selection capable of choosing the most suitable fault location algorithm according to available fault event data. Proposed solution will provide modular architecture that can be easily expended and altered to adjust to different power systems and fault location algorithms.

5. Conclusion

In order to allow timely inspection, repair and restoration, extraction of correct information about fault location and its nature should be available. Automatic protection equipment actions and utility personnel actions are discussed. They reveal several shortcomings in the existing fault investigation procedure. This section gives overall picture of the problem that the new approach and procedure for OFL algorithm selection aim to solve.

CHAPTER II

SUMMARY OF EXISTING FAULT LOCATION ALGORITHMS

1. Introduction

Typical power system contains several thousands of transmission lines. Installation of recording devices at each transmission line is very expensive and this approach cannot be found in practice. It is common that Digital Fault Recorders (DFRs) are placed in critical substations and record voltages, currents and breaker contacts associated with several transmission lines connected to that substation. Protective relays are spread all over the system, but some of them are still electromechanical and they do not have capability to record measurements. In some cases it can happen that there are no recordings at all available close to a fault. For a case shown on Figure 2.1 depending on fault occurrence different DFRs may be triggered but all of them are distant to the actual fault location. It is clear that depending on data availability with respect to the possible fault location the use of different fault location algorithms is possible. In the rest of this section brief review and evaluation of most known fault location algorithms will be made.

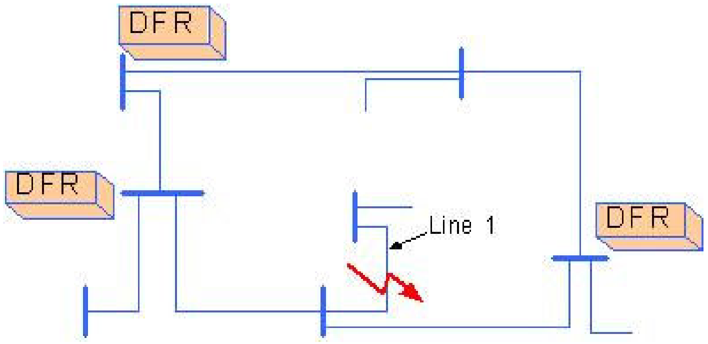


Figure 2.1: Layout of DFRs closest to a fault in the case of fault present on Line1

2. Classification of Fault Location Algorithms

There are several criteria (space, time, line model, signal component etc.) that could be used for classifying fault location (FL) algorithms.

Existing FL algorithms based on the line model and signal component used

There are two main groups of FL algorithms in this classification [2]:

a) Phasor- based algorithms that use only the fundamental component of the input signals. Phasor-based algorithms are used in the standard approaches to FL. These algorithms use steady state components of voltage and current measured at one or more points around the transmission line.

b) Partial differential equation-based algorithms that use time domain representation of the signal and distributed-parameter model of the transmission line. These algorithms are based on fact that partial differential equations of the transmission line model have two characteristics of the solution: position and time [2]. By placing measured voltage and current as boundary condition in those functions fault location can be calculated. Two approaches based on this principle can be recognized. One solves partial equations using numerical methods. The other does not require the solution of partial differential equations; it is based on traveling wave methods.

Existing FL algorithms based on time as a reference for signal sampling

In order to perform the FL calculations, samples of currents and voltages need to be taken. Samples of input signals are taken by performing analog-to-digital (A/D) conversion at the time the measurement is taken. As shown in Figure 2.2, samples are taken by sample and hold (S/H) circuit, where clock used for initiating S/H circuit can be applied either synchronously for all measured channels or sequentially as each channel is measured (scanning) [3].

Recovery of the information from data samples depends heavily on whether the signals were sampled synchronously or scanned. Although still not so commonly implemented, synchronized sampling is more desirable from stand point of FL algorithms. During fault analysis knowing phase difference between the signals may be critical, which can be easily obtained from the signals that are synchronously sampled. In order to achieve this, a reference clock from Global Positioning System (GPS) of satellites is used in practice.

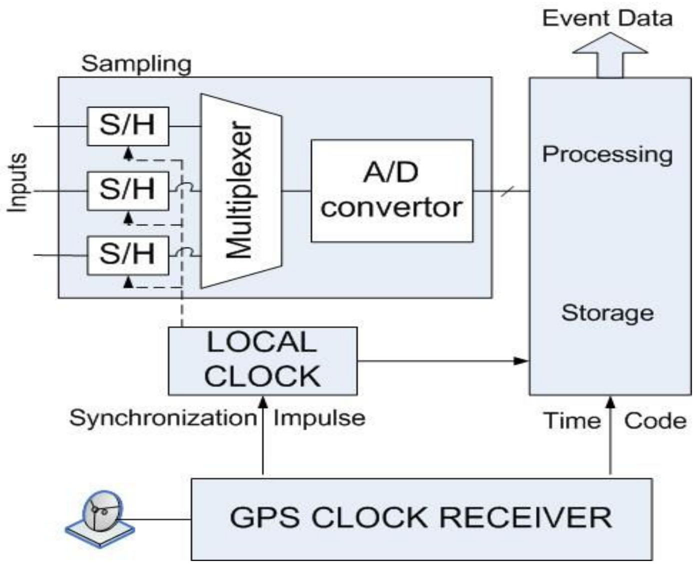


Figure 2.2: Synchronized sampling

Existing FL algorithms using the time-domain representation of the waveforms as a reference

Currents and voltages may be measured to determine time-domain representation or to reconstruct a phasor [4]. During fault event, current and voltage waveforms experience transient behavior as they change status from pre-fault to post-fault steady state. Figure 2.3 shows the different states that the current waveforms of a faulted line can experience.

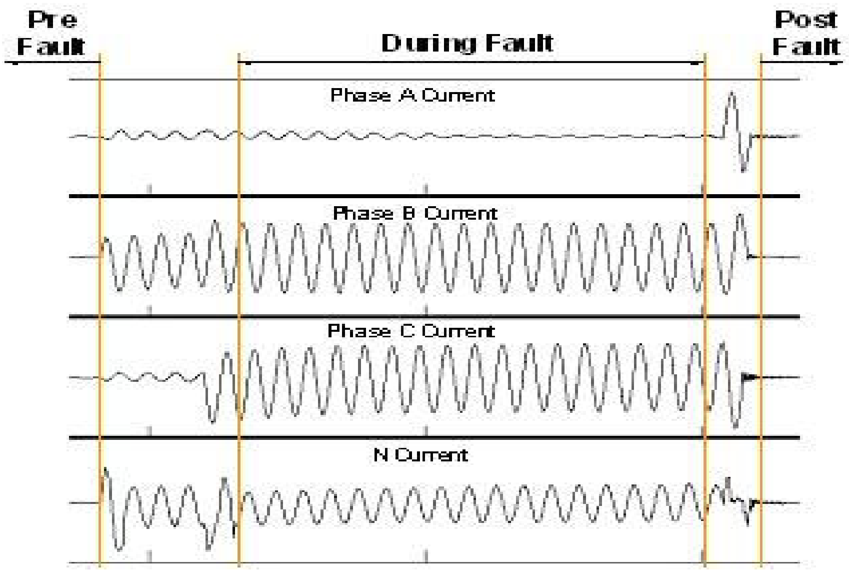


Figure 2.3: Historical measurements

FL algorithms can be classified depending on combination of pre-fault, during fault, post fault or transient states of the waveforms of the faulted line they use.

Existing FL algorithms using the spatial placements of physical measurements as a reference

In general, IEDs used for obtaining fault waveform measurements are not spread uniformly across the power system. Critical substations may be equipped with greater number of IEDs that are capable of recording and communicating data, while less accessible locations like tapped lines may be poorly instrumented with recording equipment. Once fault occurs, different IEDs are triggered. Some of them can be located close to the fault, while some of them might be far away. It is important to recognize that depending on the spatial origin of available recordings, different FL algorithms may be applicable. The following cases may be observed from the system layout given in Figure 2.4: a) DFR recording is available from only one end of the faulted section AB, b) there are no direct recordings available from the faulted section BC but only one recording far from the fault at point A is available, c) measurements from both ends of the faulted section AB are available, and d) direct recording from one end of the faulted section and a recording far from the fault at point A are available.

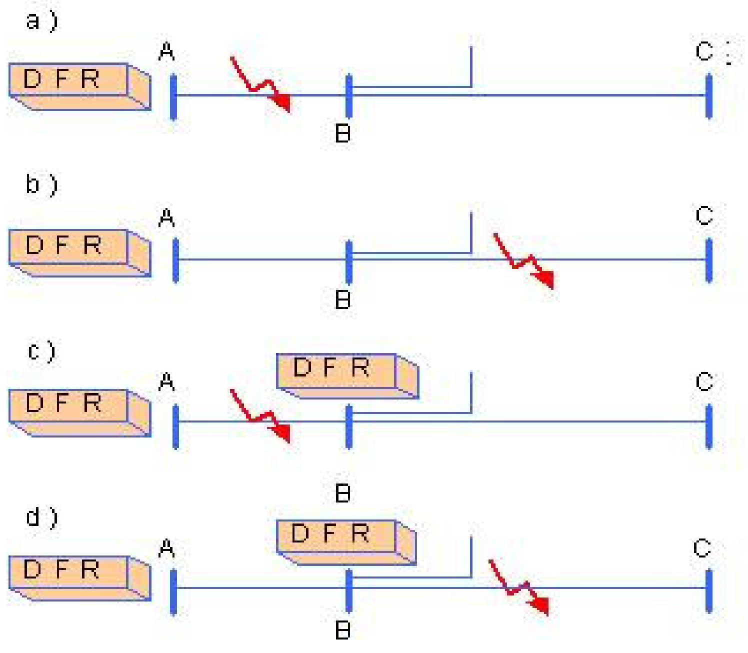


Figure 2.4: Layout of triggered DFRs

From the mentioned examples, three types of spatial placement of measurements relative to FL can be recognized:

Single ended

Multiple ended (two, three etc.)

Sparse system-wide

3. Review of Common Algorithms

In this section different representatives of FL algorithms, classified according to the placement of physical measurements will be reviewed.

Single ended FL

In many situations recorded data are available only from one end of a line, so it is possible to apply only single end FL algorithms. The most popular algorithms of this type are impedance based methods which assume that fault distance is proportional to the measured impedance. In order to understand issues in estimating FL using single end measurements simplified one-line system diagram shown in Figure 2.5 is analyzed [3].

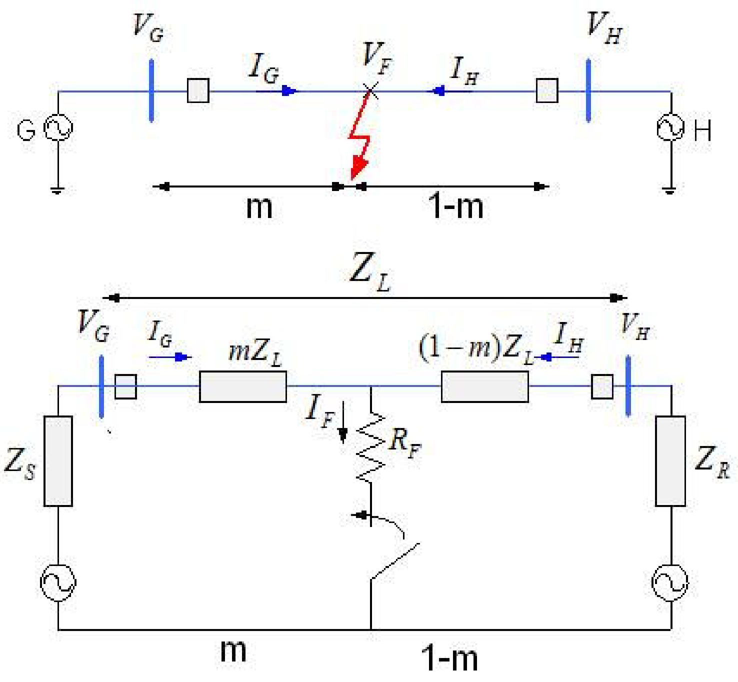


Figure 2.5: The one-line and equivalent circuit representations of a three-phase fault on a transmission line with two sources, G and H

First we will assume that faulted line shown on Figure 2.5 is homogeneous, which means that impedances are distributed uniformly through out the whole length. A fault with resistance R_F is present on the line. If we assume that we have measurements of voltages and currents at terminal G, the measurements need to be correlated with unknown fault distance m. The voltage at terminal G can be expressed as:

$$V_G = mZ_L I_G + R_F I_F \tag{1}$$

where

V_G is the voltage at terminal G

m is the distance to the fault in per unit

Z_L is the line impedance between terminals G and H

I_G is the line current from terminal G

R_F is the fault resistance

I_F is the total fault current

It can be noticed that (1) is a complex scalar equation, equivalent to two real scalar equations. However the number of unknowns is four: m, the phase and amplitude of fault current phasor I_F and fault resistance R_F. It is obvious that if it was possible to minimize effect of $R_F I_F$ term, three unknowns would be removed. By dividing (1) by the measured current, I_G impedance measured at the terminal G is obtained:

$$Z_{FG} = \frac{V_G}{I_G} = mZ_L + R_F \frac{I_F}{I_G} \tag{2}$$

where

Z_{FG} is the apparent impedance to the fault measured at terminal G

Simple reactance method ignores $R_F \frac{I_F}{I_G}$ term and expresses unknown FL as:

$$m = \frac{\operatorname{Im}(\frac{V_G}{I_G})}{X_{1L}} \tag{3}$$

If $\angle I_G = \angle I_F$ and $R_F = 0$, computation error using this method is zero. If the in feed current from remote terminal is not zero and is not in phase with local current, the ratio of fault current and measured current can be a complex number, meaning that the fault resistance will be represented as impedance with a reactive component, which can be inductive or capacitive. In order to investigate parameters that influence $R_F \frac{I_F}{I_G}$ term, this term is separated into pre-fault and during-fault systems using superposition. Measured current is presented as:

$$I_G = \Delta I_G + I_L \tag{4}$$

where

ΔI_G is the difference current

I_L is the pre fault load current

Using superposition terms we can express apparent impedance as:

$$Z_{FG} = \frac{V_G}{I_G} = mZ_L + R_F \frac{1}{d_S n_S} \tag{5}$$

where

$$d_S = \frac{\Delta I_G}{I_F} = \frac{Z_H + (1-m)Z_L}{Z_H + Z_L + Z_G} = |d_S| \angle \beta \tag{6}$$

$$n_S = \frac{I_G}{\Delta I_G} = |n_S| \angle \gamma \tag{7}$$

Two factors determine reactive component caused by the fault resistance:

d_S represents the current distribution factor determined by the system impedances. If the system is homogenous β is zero.

n_S represents the circuit loading factor determined by the load flow. If there is load flow on the system γ is not zero. If the magnitude of the fault current I_G is much greater than the magnitude of load current, the angle γ will approach zero [3].

In order to improve the simple reactance method the effect of load flow should be reduced and fault resistance should be minimized. This was the goal of various techniques used to improve this algorithm. Depending on assumptions made the performance of resulting FL algorithm varies. Few of them will be investigated in the rest of this section.

One of the well-known algorithms of this type is presented in [5]. In this algorithm it is assumed that all the impedances in (6) have approximately the same phases. As a consequence of this assumption the fault current I_F is proportional to the sending end fault current ΔI_G [2]. So by multiplying (1) with ΔI_G^* on both sides and saving imaginary part we have:

$$\mathrm{Im}(V_G * \Delta I_G^*) = m\,\mathrm{Im}(Z_L * I_G * \Delta I_G^*) + R_F\,\mathrm{Im}(I_F * \Delta I_G^*) \tag{8}$$

$$\rightarrow m = \frac{\mathrm{Im}(V_G * \Delta I_G^*)}{\mathrm{Im}(Z_L * I_G * \Delta I_G^*)} \tag{9}$$

For ideal homogeneous system the assumption is correct, but as the angle difference between I_G and I_F increases the error of the estimated fault location also increases and it is proportional to the fault resistance and $\sin(\beta)$. This method compensates for the error caused by circuit loading (n_S).

The authors of this algorithm made several claims about necessity of pre-fault current recordings or assumption of constant fault impedance. Another method, modified Takagi [6], uses the zero sequence current instead of superposition current for the ground faults. Pre fault data is not needed in this method. In order to adjust β in the case of non homogeneous systems an angle correction is proposed by this method. The corrected angle is suitable for only one fault location along the line. In general, the value of β varies with distance and it can not be calculated unless an accurate source impedances are known [3]. Fault location method using source impedance in order to compensate for the error caused by the current distribution factor (d_S) is proposed in [7]. In this approach the zero sequence current component of the fault current and the currents distribution factor are eliminated to avoid inaccuracy in the zero sequence impedances. A current distribution factor is then developed for positive and negative sequence. By rewriting equation (1) and substituting (6) we get:

$$V_G = mZ_L + R_F \frac{\Delta I_G (Z_H + Z_L + Z_G)}{Z_H + (1-m)Z_L} \tag{10}$$

Finally, quadratic expression (11) is extracted and used for estimating FL [3].

$$m^2 - mk_1 + k_2 - k_3 R_F = 0 \tag{11}$$

where k1, k2 and k3 are complex functions of measured voltage, current and source impedances.

It can be noticed that equation (12) is a complex scalar equation, equivalent to two real scalar equations with two unknowns: m and R_F. By eliminating R_F, per unit distance to the fault m can be easily solved. This method compensates for the load, fault resistance and argument (angle) of the current distribution factor. The knowledge of accurate source impedances is required, which is not always available. Pre fault data are needed.

Beside other factors impedance–based algorithms discussed above require knowledge of faulted phase, which is not always available. Mutual coupling between phases may introduce errors. By using symmetrical components and sequence circuit, other techniques for fault location using data from single end are introduced. In [8, 9] FL algorithm that uses symmetrical components is presented. This algorithm requires fault type and pre fault data information. Next, another approach where those data are not required will be presented.

There are three symmetrical component phasors: zero, positive, and negative sequence. Each phase vector is a linear combination of these three components. During non-fault condition on a transmission line, zero and negative sequences are equal to zero. Relation between phase vectors and symmetrical components is expressed as:

$$V^S = AV^P \tag{12}$$

where

V^S is symmetrical component vector $[V^0, V^1, V^2]$

V^P is phasor vector $[V^a, V^b, V^c]$

$$\text{Matrix A is } A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & e^{j\frac{4\pi}{3}} & e^{j\frac{2\pi}{3}} \\ 1 & e^{j\frac{2\pi}{3}} & e^{j\frac{4\pi}{3}} \end{bmatrix} \tag{13}$$

We can rewrite (1) in form of phasor vectors:

$$V_G^P = mZ^P I_G^P + V_F^P \tag{14}$$

In equation (14), the index p denotes phasor vector of corresponding values. By converting phasor values into symmetrical components equation (14) becomes:

$$V_G^S = mZ^S I_G^S + V_F^S \tag{15}$$

where $Z^S = A^{-1} Z^P A$ (16)

Important to notice is that while the matrix Z^P has diagonal and off-diagonal elements, only diagonal elements of matrix Z^S are non zero. The equation (15) can be broken into three independent scalar complex equations. This leads to removing the mutual inductance influence. Since the negative sequence vector equals zero prior a fault, pre fault measurements are not needed. Classification of the fault type is not necessary. According to the authors the equivalent impedances and the line impedance of the negative sequence circuit are more likely to have the same phases than in the case of the phase impedances of the line [10]. In some situations in the case of symmetric faults, negative- sequence phasors might remain zero after the fault. The negative sequence circuit is not suitable for FL estimation in those cases.

In general, these algorithms require simple calculation. Their accuracy depends on the simplified assumptions. The two-end algorithms require fewer assumptions and potentially they are more accurate, which will be discussed in the next section.

Two-ended FL

Classification of FL algorithms depends on the time used as a reference for signal sampling. They are classified into two main categories: algorithms that require and those which do not require synchronized samples of input measurements from two ends.

Synchronized sampling two-ended algorithm

A two end FL algorithm that uses synchronized sampling at two ends of a transmission line is described in [11]. It belongs to the time based methods and uses both the lumped and distributed model depending on the transmission line length. This algorithm is based on fact that the voltages and currents from one end of the faulted line can be expressed in term of the voltages and currents at the opposite end. In the case that fault occurs at some point x on transmission line as Figure 2.6 shows we have:

$$v_F = L^v \{v_S, i_S, d - x\} \quad (17)$$

$$v_F = L^v \{v_R, i_R, x\} \quad (18)$$

By combining equations (17) and (18) we get:

$$L^v \{v_S, i_S, d - x\} - L^v \{v_R, i_R, x\} = 0 \quad (19)$$

where

v_S, i_S, v_R, i_R are vectors of voltages and currents of the sending and receiving end respectively,

L^v is operator that defines mathematical model of the line

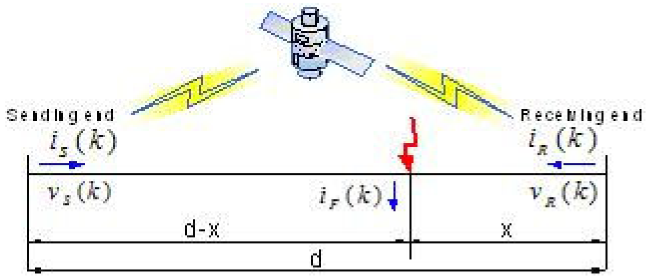


Figure 2.6: Faulted transmission line

Two types of transmission line mathematical models are used for FL algorithms: distributed and lumped parameter model. Distributed parameter models are mostly suited for long transmission lines, while lumped parameter model is used for short lines only. For the synchronized sampling based FL presented in [11] two algorithms are developed: one for short and one for long lines. Now, the short line application will be presented.

Short line is considered as less than 50 miles and it is commonly represented as the serial connection of an inductance and a resistance since the parallel line capacitance is negligible. It is assumed that the transmission line is homogeneous. We can rewrite equations (17) and (18) in terms of the short line model as:

$$v_F = L^v\{v_S, i_S, d-x\} = v_S(t) - ri_S(t)(d-x) - l\frac{di_S}{dt}(d-x) \tag{20}$$

$$v_F = L^v\{v_R, i_R, x\} = v_R(t) - ri_R(t)x - l\frac{di_R(t)}{dx}x \tag{21}$$

By combining equations (20) and (21), a generic fault location equation (19) becomes a system of three equations:

$$v_{mS}(t) - v_{mR}(t) - d\sum_{p=a,b,c}\left[r_{mp}i_{pS}(t) + l_{mp}\frac{di_{pS}(t)}{dt}\right] + x\sum_{p=a,b,c}\left[r_{mp}i_{pR}(t) + r_{mp}i_{pS}(t) + l_{mp}\frac{di_{pR}(t)}{dt} + l_{mp}\frac{di_{pS}(t)}{dt}\right] = 0 \qquad m=a,\ b,\ c \tag{22}$$

Since the phase voltage and currents are available in the sampled form expression (22) is discretized:

$$A_m(k) + B_m(k)x = 0 \tag{23}$$

$$A_m(k) = v_{mS}(k) - v_{mR}(k) - d\sum_{p=a,b,c}\left[(r_{mp} + \frac{l_{mp}}{\Delta t})i_{pS}(k) + \frac{l_{mp}}{\Delta t}i_{pS}(k-1)\right] \tag{24}$$

$$B_m(k) = \sum_{p=a,b,c}\left[(r_{mp} + \frac{l_{mp}}{\Delta t})[i_{pR}(k) + i_{pS}(k)] + l_{mp}[i_{pR}(k-1) + i_{pS}(k-1)]\right] \tag{25}$$

where,

m=a, b, c

k=1, 2, ..., N

N is the total number of considered samples

Δt is the sampling step

Finally the unknown fault location estimate is determined using least square estimate for all three phases together:

$$x = \frac{-\sum_{m=a,b,c}\sum_{k=1}^{N} A_m(k)B_m(k)}{\sum_{m=a,b,c}\sum_{k=1}^{N} B_m^2(k)} \tag{26}$$

For the transmission lines longer than 150 miles, the shunt capacitance can not be neglected. Since contribution of resistance to the serial impedance of a conductor as well as the contribution of conductance to the shunt admittance are almost negligible, the long lines are often considered lossless and they can be expressed using a pair of partial differential equations:

$$\begin{gathered}\frac{\partial v(x,t)}{\partial x} = I\frac{\partial i(x,t)}{\partial t}\\ \frac{\partial i(x,t)}{\partial x} = C\frac{\partial v(x,t)}{\partial t}\end{gathered} \tag{27}$$

Using traveling wave method and Bergeron decomposition (27), equations that relate voltages and currents of the two ends of homogeneous line become [11]:

$$v_R(t) = \frac{z}{2}\left[i_S(t-\tau) - i_X(t+\tau)\right] - \frac{1}{2}\left[v_X(t-\tau) - v_S(t+\tau)\right] \tag{28}$$

$$i_R(t) = -\frac{1}{2}\left[i_S(t-\tau) + i_S(t+\tau)\right] - \frac{1}{2z}\left[v_S(t-\tau) - v_S(t+\tau)\right] \tag{29}$$

where

z is the surge impedance of the line: $z = (l/c)^{1/2}$

τ is the surge traveling time: $\tau = d(lc)^{1/2}$

By applying equations (28) and (29) into generic FL expression (19) we get:

$$L^v\{\Delta v_R, \Delta i_R, x\} = \frac{z}{2}\left[\Delta i_R(t-vx) - \Delta i_R(t+vx)\right] + \frac{1}{2}\left[\Delta v_R(t-vx) + v_R(t+vx)\right] = 0 \tag{30}$$

where $v = (lc)^{1/2}$

After discretizing equation (30) we have:

$$L^v\{\Delta v_{Rn}, \Delta i_{Rn}, x\} = \frac{z}{2}\left[\Delta i_{R,n-m} - \Delta i_{R,n+m}\right] + \frac{1}{2}\left[\Delta v_{R,n-m} + v_{R,n+m}\right] = 0 \tag{31}$$

where

$\Delta i_{R,n} = \Delta i_R(n\Delta t)$

$\Delta v_{R,n} = \Delta v_R(n\Delta t)$

$m\Delta t = vx$

m, n = integer

The unknown fault location estimate is determined using least square estimate. Criterion that should be minimized is:

$$J(x) = \sum_{n=0}^{N}\left[L^v\{\Delta v_{Rn}, \Delta i_{Rn}, x\}\right]^2 \tag{32}$$

The search for a minimum is performed in an approximate way. Several tentative values are used to make an approximation function using parabola. The minimum of this parabola determines the final FL estimation.

The algorithm for long transmission line involves additional computational burden when compared with the short line application. Described approach based on synchronized sampling does not depend on the fault type, fault resistance or changes in the fault incidence angle. It is valid for any line operating conditions. However, this method is affected by the sampling interval and numerical method used for solving the system equations. Although data sampling synchronization capabilities between different devices are increasing in last years, still most of the data sample sets, obtained at two ends of the line, are not synchronized to each other. It is necessary to investigate other two end algorithms that don't require synchronized sampling of measurements from two ends.

Phasor based two end FL algorithms that can be used with both synchronized and unsynchronized data from two or three end transmission lines are presented in [12]. In the case that obtained voltages and currents from two terminals of a faulted line are synchronized as shown on Figure 2.6, they can be expressed as:

$$V_{S,abc} = V_{F,abc} - xZ_{abc}I_{S,abc} \tag{33}$$

$$V_{R,abc} = V_{F,abc} - (d-x)Z_{abc}I_{R,abc} \tag{34}$$

where

Z_{abc} is the three-phase series impedance of line per mile

$V_{F,abc}$ is the voltage vector at the fault

Since absolute values of $V_{F,abc}$ obtained from equations (33) and (34) are the same, quadratic equation with respect to x can be obtained and easily solved [13]. The method presented in [12] offers an increase in precision even if the noise is present in measurements and this method will be presented further. By eliminating $V_{F,abc}$ we have

$$V_{S,abc} - V_{R,abc} + dZ_{abc}I_{R,abc} = xZ_{abc}(I_{S,abc} + I_{R,abc}) \tag{35}$$

Equation (35) can be rewritten as

$$\begin{bmatrix} Y_a \\ Y_b \\ Y_c \end{bmatrix} = \begin{bmatrix} M_a \\ M_b \\ M_c \end{bmatrix} D \tag{36}$$

where

$$Y_j = V_{jS} - V_{jR} + d\sum_{i=a,b,c} Z_{ji}I_{iR} \tag{37}$$

$$M_j = -\sum_{i=a,b,c} Z_{ij}(I_{iS}+I_{iR}) \tag{38}$$

j=a, b, c

In equation (36) only one unknown is present and there are six real equations. Using the least square method FL estimate is expressed as:

$$x = -(M^T M)^{-1} M^T Y \tag{39}$$

In the case available measurements are not synchronized the above approach is modified to take into account synchronization angle δ. Equation (34) in this case becomes:

$$V_{R,abc} e^{j\delta} = V_{F,abc} - (d - x) Z_{abc} I_{R,abc} e^{j\delta} \tag{40}$$

After rearranging equations (33) and (40) we get:

$$\frac{V_{j1}}{V_{j2}} = M1_j x + M2_j e^{j\delta} + M3_j x e^{j\delta} \tag{41}$$

where

$$M1_j = \frac{1}{V_{jR}} \sum_{i=a,b,c} Z_{ji} I_{iS} \tag{42}$$

$$M1_j = 1 - \frac{1}{V_{jR}} \sum_{i=a,b,c} Z_{ji} I_{iR} \tag{43}$$

$$M1_j = \frac{1}{V_{jR}} \sum_{i=a,b,c} Z_{ji} I_{iR} \tag{44}$$

j=a, b, c

In equation (41) there are two unknowns and there are six real equations. Using iterative process and least square method for updating the values, FL estimate can be obtained. Similar approach is taken in the case when measurements from three terminals are available and it is described in [12].

One more two-end FL algorithm that does not require synchronization of data from two ends is introduced in [14]. Like in the previous approach voltage at the fault is removed by combining voltage expressions from the sending and receiving ends:

$$V_S e^{j\delta} - V_R + Z I_R = mZ(I_S e^{j\delta} + I_R) \tag{45}$$

After decoupling equation (45) to real and imaginary components we have:

$$\mathrm{Re}(V_S)\sin\delta + \mathrm{Im}(V_S)\cos\delta - \mathrm{Im}(V_R) + C_4 = m(C_1 \sin\delta + C_2 \cos\delta + C_4) \tag{46}$$

$$\mathrm{Re}(V_S)\cos\delta - \mathrm{Im}(V_S)\sin\delta - \mathrm{Re}(V_R) + C_3 = m(C_1 \cos\delta - C_2 \sin\delta + C_3) \tag{47}$$

where

$$C_1 = R * \mathrm{Re}(I_S) - X * \mathrm{Im}(I_S) \tag{48}$$

$$C_2 = R * \mathrm{Im}(I_S) + X * \mathrm{Re}(I_S) \tag{49}$$

$$C_3 = R * \mathrm{Re}(I_R) - X * \mathrm{Im}(I_R) \tag{50}$$

$$C_4 = R * \mathrm{Im}(I_R) + X * \mathrm{Re}(I_R) \tag{51}$$

After eliminating m from equations (46) and (47) a resulting equation is:

$$a * \sin\delta + b * \cos\delta + c = 0 \tag{52}$$

where

$$a = -C_3 \mathrm{Re}(V_S) - C_4 \mathrm{Im}(V_S) - C_1 \mathrm{Re}(V_R) - C_2 \mathrm{Im}(V_R) + C_1C_3 + C_2C_4) \tag{53}$$

$$b = C_4 \mathrm{Re}(V_S) - C_3 \mathrm{Im}(V_S) - C_2 \mathrm{Re}(V_R) + C_1 \mathrm{Im}(V_R) + C_2C_3 - C_1C_4) \tag{54}$$

$$c = C_2 \mathrm{Re}(V_S) - C_1 \mathrm{Im}(V_S) - C_4 \mathrm{Re}(V_R) + C_3 \mathrm{Im}(V_R) \tag{55}$$

Using the Newton-Raphson method equation (52) with one unknown δ can be solved. Once it is solved fault location estimate can be expressed from (46) as:

$$m = \frac{\mathrm{Re}(V_S)\sin\delta + \mathrm{Im}(V_S)\cos\delta - \mathrm{Im}(V_R) + C_4}{C_1 \sin\delta + C_2 \cos\delta + C_4} \tag{56}$$

In order escape the unwanted zero sequence effect equations (48-51) and (53-55) can be written in terms of positive, negative and zero sequence voltages and currents as shown in [14]. Using the positive sequence solution zero sequence effect is removed and fault type is not needed. In the case of high resistance faults the approach using negative sequence might be more suitable. Authors proposed a compensation for long lines in [14], so that shunt capacitances parallel to the fault resistance don't affect accuracy.

Previously described two-end FL algorithms that use non synchronized measurements did not need knowledge of the fault type. The developed algorithm uses symmetrical components for FL estimation and completely eliminates complex arithmetic operations [15]. This algorithm develops single equation that is valid for ten shunt faults:

$$x = \frac{k_1C_1 + k_2C_2 + k_3C_3 + k_4C_4 + k_5C_5}{k_1C_7 + k_2C_8 + k_3C_9 + k_4C_{10} + k_5C_{11}} \tag{57}$$

where

$$C_1 + jC_2 = V_{1p} - V_{1q} + (V_{rq1} + V_{rq1}^{'})$$

$$C_3 + jC_4 = V_{2p} - V_{2q} + V_{rq2}$$

$$C_5 + jC_6 = V_{0p} - V_{0q} + V_{rq0}$$

$$C_7 + jC_8 = (V_{rp1} + V'_{rp1}) + (V_{rq1} + V'_{rq1})$$

$$C_9 + jC_{10} = V_{rp2} + V_{rq2}$$

$$C_{11} + jC_{12} = V_{rp0} + V_{rq0}$$

0,1,2 mark the zero, positive and negative sequences respectively

p and q indexes mark the sending end receiving bus respectively

$$V_{rp1} = (I_{1p} - I'_{1p})Z_{1S} \text{ , } V_{rp2} = I_{2p}Z_{1S} \text{ , } V_{rp0} = I_{0p}Z_{0S} + I_0 Z_{0m} \tag{58}$$

mark (') indicates the pre fault values

Values of the coefficients k_i are given in Table 2.1.

Table 2.1: List of coefficients for equation (57)

Fault type	k_1	k_2	k_3	k_4	k_5
a-b-c	1	0	0	0	0
b-c	1	0	-1	0	0
a-b	1	0	0.5	0.866	0
a-c	0.5	0.866	1	0	0
b-c-g	1	0	-1	0	0
a-b-g	0.5	0.866	0	0	1
c-a-g	0.5	0.866	1	0	0
a-g	1	0	1	0	1
b-g	0.5	-0.866	0.5	0.866	-1
c-g	0.5	0.866	0.5	-0.866	-1

Finally one more algorithm that belongs to the group of impedance based multi terminal FL algorithms is presented. This algorithm is applicable even when zero-sequence infeed from the transmission line taps is present [16]. It uses negative sequence quantities and it could be used for FL for unbalanced faults. Similar starting assumption like in the previous cases can be made that fault voltage is the same when viewed from all ends of the protected line Figure 2.7.

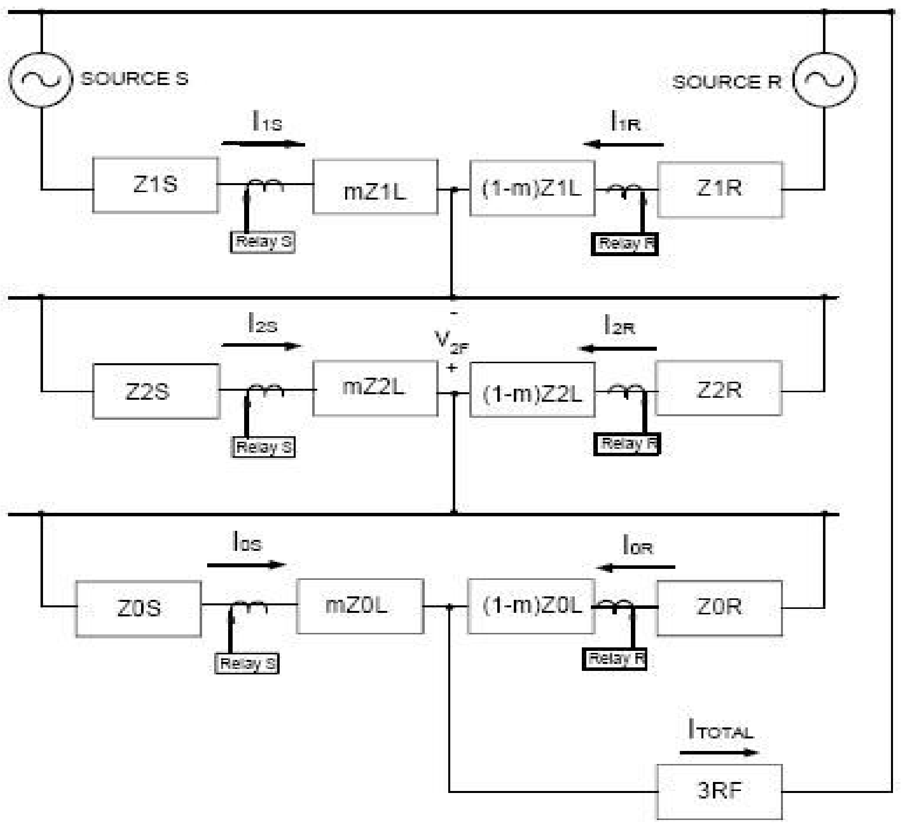


Figure 2.7: Connection of sequence networks for a line-to-ground fault at m [16]

We express negative sequence fault voltage seen by relays at end S and end R as:

$$V_{2F} = -I_{2S}(Z_{2S} + mZ_{2L}) \tag{59}$$

$$V_{2F} = -I_{2R}(Z_{2R} + (1-m)Z_{2L}) \tag{60}$$

By eliminating V_{2F} from (59) and (60) we get:

$$I_{2R} = I_{2S}\frac{(Z_{2S} + mZ_{2L})}{(Z_{2R} + (1-m)Z_{2L})} \tag{61}$$

In order to avoid the requirement for synchronization of measurements at relays locations S and R, the magnitude of the value shown in equation (61) is used:

$$|I_{2R}| = \left| \frac{I_{2S}Z_{2S} + mI_{2S}Z_{2L}}{(Z_{2R} + Z_{2L}) - mZ_{2L})} \right| \tag{62}$$

Equation (62) can be further simplified as:

$$|I_{2R}| = \left| \frac{(a + jb) + m(c + jd)}{(e + jf) - m(g + jh))} \right| \tag{63}$$

where

$I_{2S}Z_{2S} = a + jb$

$I_{2S}Z_{2L} = c + jd$

$Z_{2R} + Z_{2L} = e + jf$

$Z_{2L} = g + jh$

After calculating magnitudes from equation (63), a quadratic equation is obtained, which is used for solving m.

$$Am^2 + Bm + C = 0 \tag{64}$$

where

$$\begin{aligned} A &= \left|I_{2R}\right|^2 (g^2 + h^2) - (c^2 + d^2) \\ B &= -2\left|I_{2R}\right|^2 (eg + fh) - 2(ac + bd) \\ C &= \left|I_{2R}\right|^2 (e^2 + f^2) - (a^2 + b^2) \end{aligned} \tag{65}$$

What should be noticed is that most phasor based two-end fault location algorithms presented till this point use lumped model of the lines. A FL algorithm that uses distributed parameters line model and equates the fault point voltage from both line ends is introduced in [17]. First, the algorithm derivation is demonstrated here for single phase line. With reference to Figure 2.6, faulted voltage can be expressed as:

$$V_F = \cosh(\gamma x)V_S - Z_O \sinh(\gamma x)I_S \tag{66}$$

$$V_F = \cosh(\gamma(d - x))V_R - Z_O \sinh(\gamma(d - x))I_R \tag{67}$$

where

Z_O is the characteristic impedance defined as $Z_O = (Z/Y)^{1/2}$

γ is the line propagation constant defined as $\gamma = (ZY)^{1/2}$

Z is the series impedance per unit length unit

Y is the shunt admittance per unit length unit

Similar to the previous approaches, an expression for FL is obtained from equations (66) and (67):

$$x = \frac{\tanh^{-1}(-B/A)}{\gamma} \tag{68}$$

where

$$A = Z_O \cosh(\gamma d) I_R - \sinh(\gamma d) V_R + Z_O I_S$$

$$B = \cosh(\gamma d) V_R - Z_O \sinh(\gamma d) I_R - V_S$$

Developed estimation is obviously independent of both fault and source impedances. For three phase application, the algorithm needs to be adjusted so that mutual impedance and capacitance between phases is taken into account. For that reason it is necessary to decouple three phase representation of equations (66) and (67) so that they are independent. This is done using theory of natural nodes and matrix function theory [17]. Method involves finding the matrix of eigenvectors of product [Q]=[Z][Y] and [S]=[Z]/[Y]. Corresponding modal quantities obtained are:

$$[V_{Sn}] = \begin{bmatrix} V_{S1} & V_{S2} & V_{S3} \end{bmatrix}^T = [Q]^{-1} \begin{bmatrix} V_{Sa} & V_{Sb} & V_{Sc} \end{bmatrix}^T \tag{69}$$

$$[I_{Sn}] = \begin{bmatrix} I_{S1} & I_{S2} & I_{S3} \end{bmatrix}^T = [S]^{-1} \begin{bmatrix} I_{Sa} & I_{Sb} & I_{Sc} \end{bmatrix}^T \tag{70}$$

As a consequence, three pairs of equation that correspond to equations (66) and (67) are available. In this case FL estimation is expressed as (mode two is used):

$$x = \frac{\tanh^{-1}(-B_2/A_2)}{\gamma_2} \tag{71}$$

where

$$A_2 = Z_{O2} \cosh(\gamma_2 d) I_{R2} - \sinh(\gamma_2 d) V_{R2} + Z_{O2} I_{S2}$$

$$B_2 = \cosh(\gamma_2 d) V_{R2} - Z_{O2} \sinh(\gamma_2 d) I_{R2} - V_{S2}$$

$$[Z_{On}] = [\gamma]^{-1} [Q]^{-1} [Z][S]$$

System-wide sparse measurement algorithm

If the power grid switching status, as well as FL and fault resistance are known and represented in the short circuit program, simulated waveform will completely match with recorded waveform for corresponding fault case. Waveform matching approach is based on the idea to compare recordings of waveforms corresponding to faults against simulated values obtained using short circuit program across the same power grid. By posing fault at different locations in the program, different simulations are carried out and corresponding waveforms (phasors) are obtained. The case where the simulated waveforms match the recorded waveforms extracted from the fault recordings the best reveals the FL. Value of equation (72) represents the matching degree of the comparison [18].

$$f_c(x, R_f) = \sum_{k=1}^{Nv} r_{kv} \left| V_{ks} - V_{kr} \right| + \sum_{k=1}^{Ni} r_{ki} \left| I_{ks} - I_{kr} \right| \tag{72}$$

where,

$f_c(x, R_f)$ -the cost function using phasors for matching

x, R_f -the fault location and fault resistance

r_{kv}, r_{ki} -weights for the errors of the voltages and currents respectively

V_{ks}, V_{kr} -simulated and recorded during-fault voltages respectively

I_{ks}, I_{kr} -simulated and recorded during-fault currents respectively

N_s, N_r -the numbers of selected voltage and current phasors respectively

k -the index of voltage or current phasors

Accuracy of the waveform matching method can be drastically influenced by accuracy of the performed simulation, as well as the selection algorithm used for posing faults for the next iteration of matching. In the approach proposed in [18] power flow and short circuit study are performed by using PSS/E Short Circuit program [19]. In order to obtain the system model that reflects status of the power grid before simulation, only the local power system topology is used. To obtain more accurate simulation results, generation, load and system wide topology status should also be used for tuning the system model prior to the use for the analysis. The authors propose use of EMS SCADA PI Historian for this purpose in [20]. This approach could be used for obtaining the latest load, branch and generator data in order to update power system model before simulation.

From equation (66) we notice that the cost function will be zero if the phasors obtained from the simulated waveforms completely match the phasors obtained from the recordings. It is obvious that the best FL is found as a global minimum of equation (66).

Hence, the FL estimation problem can be translated into an optimization problem. An optimal way for posing faults based on Genetic Algorithm (GA) is used. Block diagram of this method is shown in Figure 2.8. In order to utilize GA, the minimization problem is converted into the maximization problem as shown in (73).

$$f(x,R_f) = C_{\max} - f_c(x,R_f) \qquad (73)$$

where

$f(x,R_f)$ -the fitness function,

Cmax –maximal fitness value in the current population

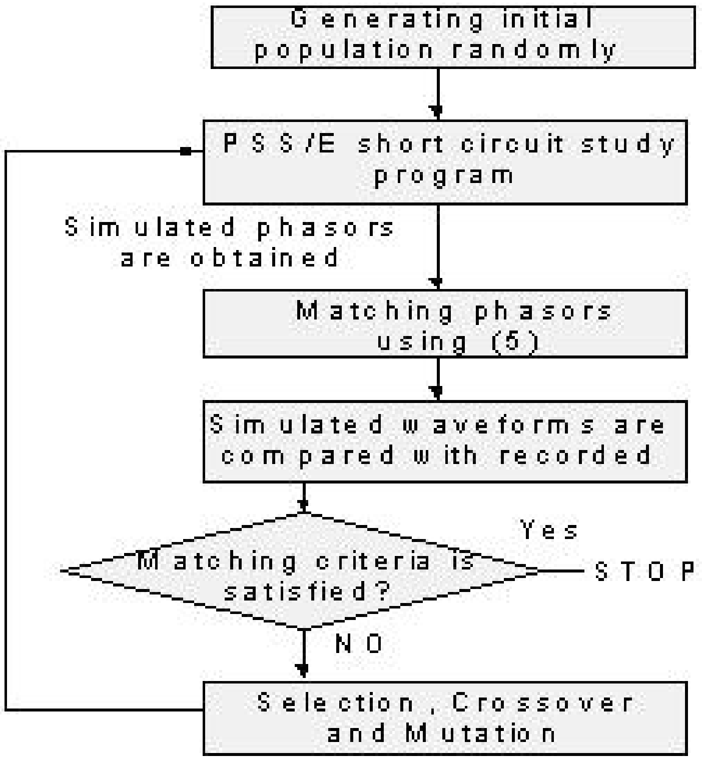

Generating initial population randomly
PSS/E short circuit study program
Simulated phasors are obtained
Matching phasors using (5)
Simulated waveforms are compared with recorded
Matching criteria is satisfied?
Yes
STOP
NO
Selection, Crossover and Mutation

Figure 2.8: Waveform matching block diagram

From equation (72) we see that x and R_f are selected as two variables, represented as binary strings in GA. Successive 'generations' of the population are created by several simple 'genetic' operators, as illustrated in Figure 2.9.

By using three GA operators optimal fault posing for next iteration of the matching is implemented [18]:

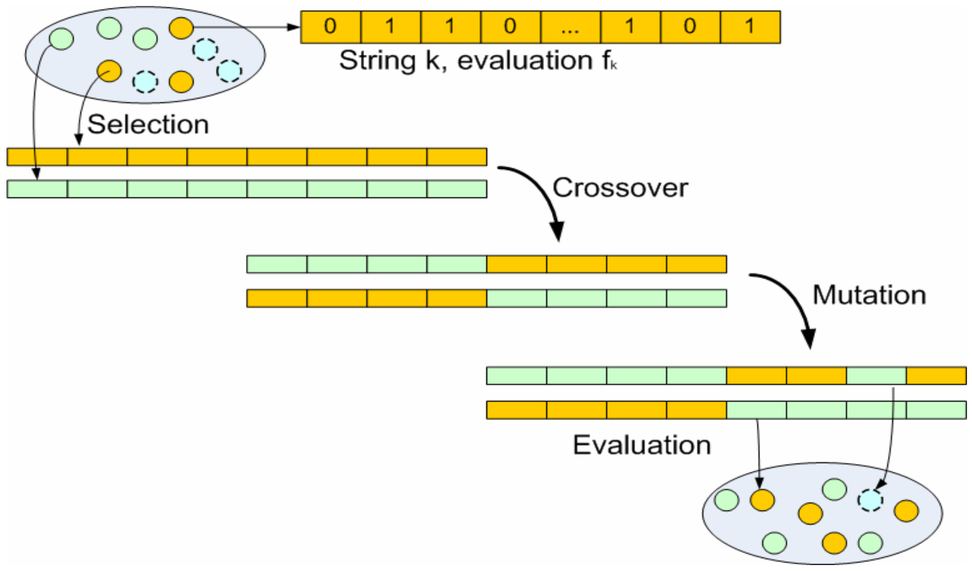


Figure 2.9: A generation of a simple genetic algorithm

a) Selection operator mimics the process of natural selection where the fittest members reproduce most often. Individual strings are copied according to their fitness function values. This means that strings with a higher value have a higher probability of contributing one or more offspring in the next generation. The reproduction operator may be implemented using a biased roulette wheel. Each string in the current generation will be allocated a slot sized in proportion to its fitness as shown in Figure 2.10.

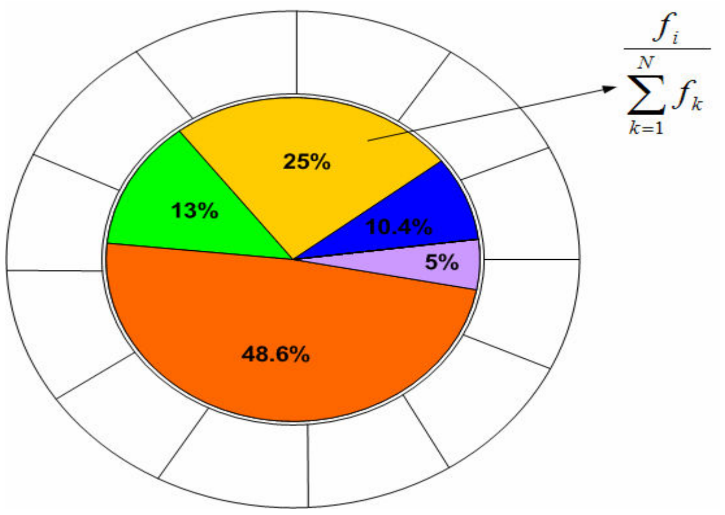


Figure 2.10: Simple reproduction allocates offspring strings using a roulette wheel with slots sized according to fitness

For each string *i* in the population, its fitness is evaluated as f_i. and the appropriate share of the roulette wheel is assigned. Thus the strings with high fitness values are allocated a large share of the wheel, while those strings with low fitness values are given a relatively small portion of the roulette wheel. To reproduce, we just spin the weighted roulette wheel N times (N is the number of the current population) to yield the reproduction candidate. After the selection relatively 'fit' solutions survive; 'unfit' solutions tend to be discarded.

b) Crossover operator, applied with probability, acts on a pair of selected members providing the exchange of binary strings. The crossover proceeds in two steps. First, pairs of the members of the reproduction candidate strings (candidate number should be even) are chosen. Second, each pair of strings undergoes crossing over as follows: an integer position k along the string is selected uniformly at random between 1 and the string length less one [1, l-1]. Two new strings are created by exchanging the partial string of the two strings between positions k+1 and l inclusively. The probability that the crossover operator is applied is denoted by P_c. This process is repeated until all pairs have been covered. Of the three genetic operators, the crossover operator is the most crucial in obtaining the final optimal result because crossover is responsible for mixing the useful information contained in each of the strings of the population in the search leading to better and better performance.

c) Mutation operator, applied with probability, affects the single bit in a member. It is applied to the candidate strings after crossover. Mutation is the occasional (with small probability) random alternation of the value of a string position. For the binary coding, the mutation operator is a stochastic bit-wise complement applied with uniform probability P_m . The mutation is needed since even though reproduction and crossover effectively search and recombine high-performance notions, occasionally they may loose some potentially useful genetic material. The mutation operator can be helpful in diversifying the search and introducing new strings into the population in order to fully explore the search space. The mutation enables the search to overcome local minima. Applying mutation too frequently may destroy the highly fit strings in the population, which may slow and impede the convergence to a solution. Usually the mutation probability is small; the empirical value is $0.01 \le P_m \le 0.1$. The detail operation is as follows: We need to choose a uniform random number $r \in [0,1]$ for each bit in a specific string. If $r \le P_m$, then the bit is flipped (from 0 to 1 or from 1 to 0); otherwise, the bit remains the same.

Finally, the question is what is the matching criteria that should be satisfied. Several convergence schemas can be utilized:

1. Fitness Sum - if the sum of the fitness values over the population falls below the convergence criteria;
2. Average Fitness – when the population average falls bellow the criteria;
3. Best Gene – when the chromosome of the best fit falls bellow the criteria;
4. Worst – when the worst chromosome falls below the criteria.

In practical implementation of this algorithm the third schema is used. It is important to notice that result will depend very much on population size. Ideally, the population size should be as large as possible to enhance the exploration of the search space. That approach can be expensive in terms of computation time, so a small population is more desirable. If the population is too small then the loss of genetic diversity may compromise the search, especially when the solution space is not topographically smooth. A small population would be more likely to quickly converge to what may be a local optimum. With the sparse spread of initial points, the global optimum may never be reached before the search converges on a poor local optimum.

In general the crossover process randomly selects two parents to exchange genes with a crossover rate Pc. A higher crossover rate allows the exploration of the solution space around the parent solution. If population is not big enough it is possible that poor local minimum is achieved as a final result. In this case the survival of the fittest solution necessary for the improvement becomes random. Because of this random property, during testing of the waveform matching, the method based on GA will be repeated several times and average value of accumulated results will be calculated.

From the above summary of algorithms, it is very important to notice that beside captured fault recordings most FL algorithms need some additional data. Some of them require knowledge of fault type, other the knowledge about the faulted transmission line. Some of the algorithms use pre-fault, some use post-fault phasors. In either case it necessary that extracted phasor is valid and input data accurate before applying the corresponding algorithm. This clearly shows that it is not possible to pick up only one FL algorithm that is applicable with high accuracy in all situations and for all possible data measurement locations. Instead, a careful comparison and evaluation of reviewed algorithms is needed, which is done in the next section.

4. Advantages and Disadvantages

Reviewed algorithms (Table 2.2) use different assumptions and different methods for coming up with a final FL estimate. Table 2.3 summarizes features of reviewed algorithms. It is important to notice that most shortcomings of the proposed methods are coming from incorrect assumptions that do not meet their requirements. This summary is used later for creating the procedure for optimal fault location algorithm selection.

Table 2.2 Reviewed Algorithms

Algorithm	Ref.	Terminal type
Simple reactance	[3]	Single end
Takagi at. al.	[5]	
Modified Takagi	[6]	
Novosel at. al.	[10]	
Ericsson at. al.	[7]	
Synchronized sampling 2-end	[11]	Multi end
Using syn. and unsyn. phasors for 2- and 3- end lines	[12]	
Using unsynchronized 2- end phasors	[14]	
2- end using symmetrical components	[15]	
Using negative sequence for 2- and 3- end lines	[16]	
Johns at. al.	[17]	
Waveform matching method using genetic algorithm	[18]	Sparse measurement

Table 2.3 Reviewed algorithms summary

Alg	Fault type	Pre fault data	R_f	Mutual coupling	d_S	n_S	Non homoge neous line	Synchroni zation	Neglected capacitance	Transposed lines	Source impedance	Sampling rate	Compu tation	# of ends
Single end														
Impedance based methods														
	Without source impedance													
[3]	X	*	X	X	X	X	X	Na	X	X	*	*		1
Reactive component of impedance is ignored														
[5]	X	X	X	X	X	*	X	Na	X	X	*	*		1
Assumption: fault current I_F is proportional to the sending end fault current ΔI_G. For ideal homogeneous system the assumption is correct, but as angle differences between I_G and I_F increases the error of estimated fault location also increases and it is proportional to R_f and $\sin(\beta)$. This method compensates for error caused by circuit loading (n_S).														
[6]	X	*	X	X	**	*	X	Na	X	X	*	*		1
In order to adjust β in case of non homogeneous systems angle correction is proposed by this method. However, corrected angle is suitable only for one fault location along the line.														
[10]	*	*	X	*	**	*	X	Na	X	X	*	*		1
Assumption: Neg. sequence d_S is a real number. According to the authors [10], the equivalent and line impedance of negative sequence circuit are more likely to have the same phases than in the case of phase impedances of the line. This method should be used FOR UNBALANCED FAULTS.														
	With source impedance													
[7]	X	X	*	*	*	*	X	Na	X	X	X	*		1
Method compensates for n_S, R_f and argument of d_S if settings are adequate. Source impedances are needed. Authors proposed extension of the method to take into account mutual coupling. Lumped capacitance model is assumed when source impedance values are determined.														

*Note: X means depends on specific property, * means independent and ** means independent under assumption*

$d_S = |d_S| \angle \beta$ is the current distribution factor, n_S is the circuit loading factor and R_f is the fault resistance

Problems of the single end algorithms:
- When fault impedance is high, the fault current is small. Since the fault current for the sending end is in the denominator in these algorithms the system introduce large errors in all cases.
-phasor extraction errors
-errors in assumptions

Table 2.3 (Continued)

Alg	Fault type	Pre fault data	R_f	Mutual coupling	Infeed from remote end	Infeed from tapped loads	Non homoge neous line	Synchroni zation	Neglected capacitance	Transposed lines	Source impedance	Sampling rate	Compu tation	# of ends
Multi ended														
Time based														
[11]	*	*	*	*	*	*	*	X	*	*	*	X	MLS	2
Phasor based														
	Without using symmetrical components													
[12]	*	*	*	X	*	X	X	*	**	*	*	*	MLS	2-3
Neglected capacitance effect is assumed to be ineffective due MLS method used, which minimizes errors.														
	With using symmetrical components													
[14]	*	*	*	*	*	X	X	*	*	X	*	*	Iter.	2
The positive sequence is used and due to high fault resistance accuracy might decrease, so it would be better to use the negative sequence at least for unbalanced faults. In that case it would be necessary to recognize fault type (balanced or not). Although an iteration method is used, result converges very fast.														
[15]	X	X	*	*	*	X	X	*	**	X	*	*	EASY	2
It uses only simple real arithmetic computations. Pre fault data can be used for estimating synchronization angle. Lumped model of the neglected capacitance can be taken into account.														
[16]	*	*	*	*	*	*	X	*	X	X	*	X	Avg	2-3
Selected quantities needed from the remote end, which can be easily transferred, are: magnitude of the negative sequence current and calculated negative-sequence source impedance. This method should be used FOR UNBALANCED FAULTS ONLY. In this case the longer an event lasts the better estimate is because using of averaging is recommended by authors.														
[17]	*	X	*	*	*	*	*	*	*	X	*	*	Matrix	2
If phasors are not synchronized, computational burden is increased. Pre fault data are used only for the synchronization. In the case measurements from two ends are already synchronized, pre fault data are not needed.														
Sparse measurements														
[18]	X	*	X	*	*	*	X	*	*	*	*	*	GA	*
This algorithm can be affected by the accuracy of the system model parameters, but once the parameters are taken into account it is applicable even in the case when taken measurements are not directly from faulted line, which is the most frequent case.														
MLS- Minimum Least Squares, GA-genetic algorithm, Iter.-iterative method, Avg- averaging The problems of phasor – based algorithms (all except [11]) is the fact that filters must be applied in order to remove noise, in the case of short lasting faults and presence of the transients it is harder to estimate phasors.														

5. Conclusion

This section presents different solutions used for FL estimation. First, it describes several criterions (space, time, line model, signal component etc.) that could be used for classifying FL methods. Then detail review of different FL algorithms is made. Finally, those algorithms are summarized and their features are compared. Summary from this section will be used for designing an approach for selecting an optimal FL algorithm.

CHAPTER III

DATA SOURCES

1. Introduction

It is important to notice that beside errors introduced by FL algorithms (incorrect or incomplete algorithm, incorrect application) there are other types of errors that could appear due to bad input data. Those errors can be classified as:

- Transducer errors: current transformer and voltage transformer errors
- Measurement errors: Amplifier gain, non-linearities, dc offset, A/D converter bit resolution, synchronization accuracy, sampling rate, etc.
- Model errors: Physical transmission line parameters, transposed vs. untransposed, charging capacitance, zero sequence impedance, mutual coupling, load flow unbalance, tapped lines, etc.

Errors introduced in the input data can influence accuracy of FL estimation much. As an example of the importance of accurate input data, sensitivity studies of the proposed FL algorithm are carried out in [21]. It is shown that: an error of –N percent in line parameters will result in an error of +N percent in computed fault distance; an error of +N percent, in recorded voltage magnitude, results in an error of +N percent in computed fault distance etc.

In order to calculate fault location two types of data may be needed: event data (fault measurements) and power grid topology and status data (system and line model). Different sources can be used to obtain these data. They will be evaluated in this chapter.

2. Obtaining Fault Recordings

Once fault occurs, different IEDs will automatically notice the fault as abnormality and record corresponding current, voltage and status signals, which are used for fault investigation. Figure 3.1 summarizes disturbance or event recorders of interest to protection engineer. They are categorized by the event duration as [22]:

- Transient - These are very short in duration and typically include faults that are cleared immediately by the circuit breaker operation. These events are generally no longer than 8 cycles for high speed clearing and 16 cycles for sequential line clearing. These events are usually analyzed to determine correct protection operation, FL, or verification of system model parameters.

- Short Term - These generally include all other time-delayed fault clearing and reclosing events where the system operation (stability) is not affected. These events are typically 20 to 60 cycles in length but may be longer if multiple protection operations are required to clear the fault. These events are analyzed to determine protection operation, FL or verify system model parameters.
- Long Term - These include those events that affect system stability such as power swings, frequency variations and abnormal voltage problems. These events are usually analyzed to determine causes of incorrect system operations. Data management techniques are employed to process a number of samples and record the value for the parameter of interest.
- Steady State - There are steady state disturbances where system operation is not threatened, but power quality is affected. This may include harmonics or sub-harmonics produced by the load and/or the interaction between power system's components. Depending upon the type of phenomena being analyzed, higher sampling rates may be required to capture these events.

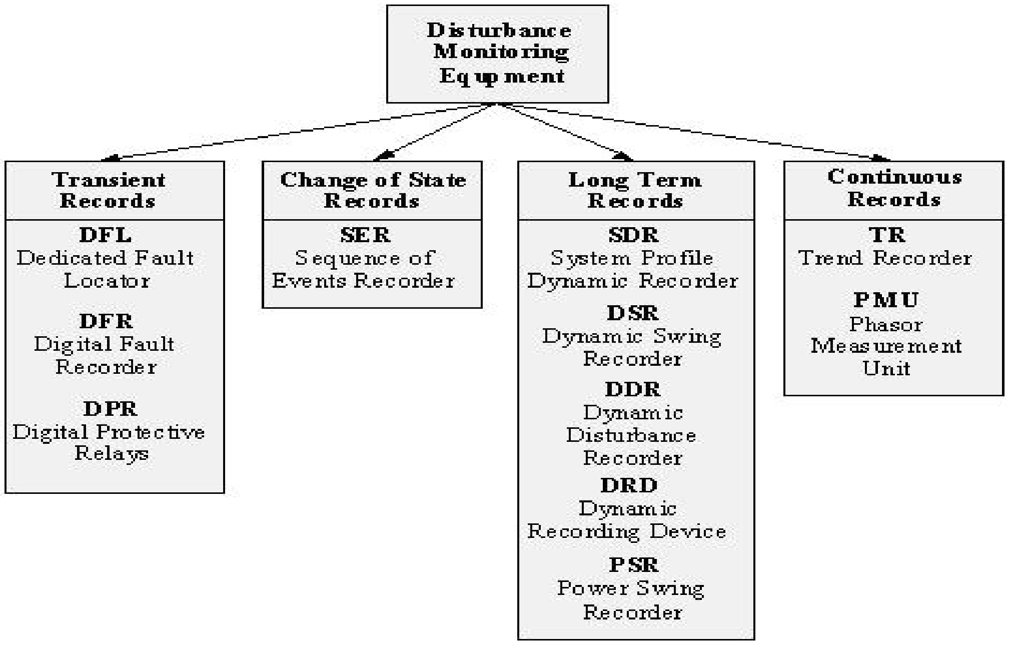


Figure 3.1: Disturbance recorders classification [22]

Fault events are usually short in duration as the fast transients appear, so it is necessary to use device with a sampling rate high enough to capture those changes in the corresponding current and voltage samples. Typical devices equipped with high-speed recording are dedicated fault locators (DFLs), digital protective relays (DPRs) or digital fault recorders (DFRs). Figure 3.2 is a transient event captured by high-speed recording, showing analog voltage and current waveforms.

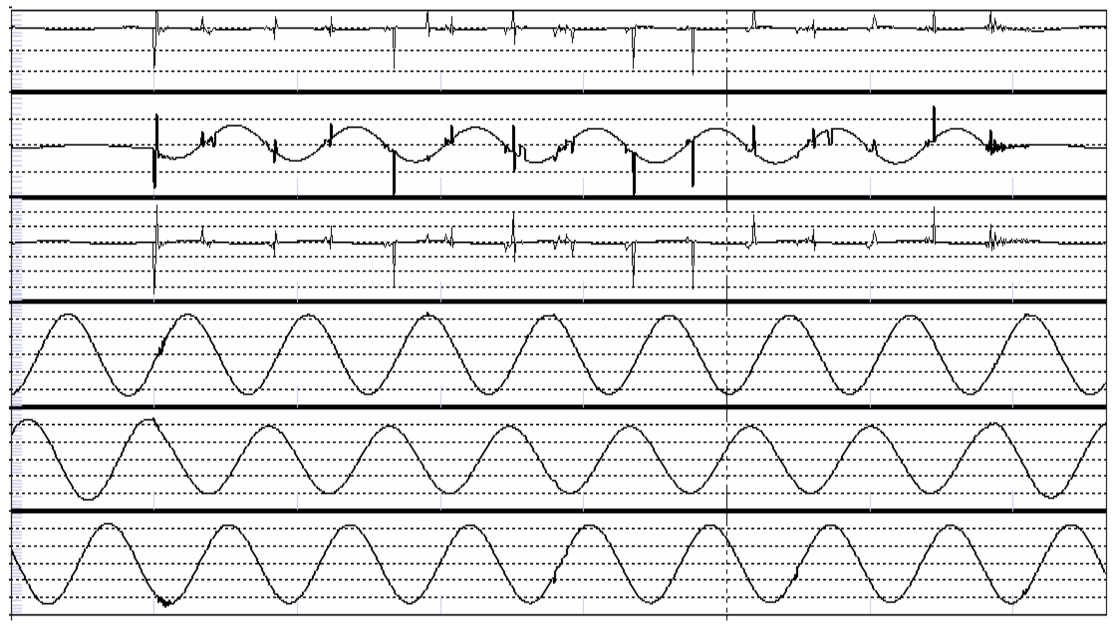


Figure 3.2: Transient record

DFLs are very flexible because the entire design can be optimized for FL application. However, DFLs are not so common because it is an expensive solution that accommodates only one function [2].

Protective relays are commonly used on all transmission lines. In the last years DPRs based on microprocessors are becoming very popular. These types of relays are capable of saving measured recordings and transferring them to remote locations using communication interfaces and they can be used for fault analysis. When considering using a DPR as a fault recorder the sampling rate at which records are stored, the length of the record, and the filtering applied in the capture of the relay must be evaluated. Many early relays filtered the analog data so that only the 60 hertz component of the waveform was captured, and then displayed using a smooth curve fitted algorithm [22]. Triggering of the recording function within the relay is programmable and based on the internal logic of the measuring elements within the device. In general DPRs may be programmed to recognize and record events in situations where they do not initiate a trip.

DFRs are commonly used in high-voltage transmission substations to record voltages and currents on transmission lines. They are usually triggered by abnormalities in measured signals caused by the fault or other disturbance occurrence in an area of observation. A DFR typically features directly measured analog channels, as well as event or binary channels. This allows the recorder to capture the time sequence of analog power system quantities, along with the breaker contacts, logic state changes, event contacts, etc. Triggering to start the capture of the data can typically be directly based on the changes in analog quantities, digital inputs, or logic. Information from DFRs can be used to confirm the occurrence of a fault, determine the duration of a fault, measure the magnitude of fault quantities of current and voltage, determine the location of a fault, define the nature or the type of fault, assess performance of relays, and assess circuit breaker performance.

The record length and sampling frequency have a major impact on the recording capability of the device with regard to memory constraints, maximum duration of a record, number of records that can be captured, etc. In the DFR, there is typically some form of memory management that allows for many records to be kept or transferred to a storage disk. In the case of the DPR, early devices may only allow for a single record to be stored, while newer devices may allow for several records to be captured before it either overwrites or stops recording. Philosophies with regard to reclosing and the capture and storage of this event may also affect the recording capabilities and memory requirements.

Once fault event signals are available it is necessary to understand implications of several stages of data conditioning that was applied to original signals. Care must be taken when choosing the appropriate device to capture high-speed data, as significant differences exist between DFRs and relays in terms of the filtering applied, which may be both analog and digital), the sampling rate, which determines the frequency response of the recorded information and the use of measurement windows, and the associated measurement algorithms that can have an impact on the data contained within the record.

Analog filtering for ac signals

Usually analog filters are used for implementing anti-aliasing feature on both the relays and DFRs. Anti-aliasing filtering eliminates higher frequencies that would otherwise overlap the lower portion of the spectrum due to the relatively low sampling rates of the IEDs with the respect to the frequency bandwidth of the transient waveforms. The cut-off frequency of the analog filter must be lower than or equal to half the sampling frequency according to the Nyquist-Shannon sampling theorem [22]. DFRs traditionally sample at 64 to 356 samples/cycle and have their cut-off filters set well above 1 kHz, yielding comparatively good spectral coverage. Relays traditionally sample from 4-32 samples/cycle, but relays are now available that sample at 64 to 128 samples/cycle. The protective relays, which main function is protection and control, use only the filtered phasor quantities or apply lower sampling rates. As a result, the signal spectrum effectively recorded by these devises is limited to few hundred hertz.

Digital filtering for ac inputs

The goal is that the recorded data have a high level of accuracy to allow for faithful representation of the original input signal. Applying digital filter applied to ac inputs impact the original data. In order to obtain input quantities for their protection and control algorithms, DPRs often perform digital pre-filtering prior to applying phasor estimation algorithms (such as the Fourier transform). Digital filters remove unwanted components of the applied signals, the dc component in particular. Digital relays tend to record sampled data after digital filtering. Newer digital relays operate similar to DFRs and record raw samples prior to digital filtering.

This is important to know in order to make sure that the stored information does not depend on any proprietary aspects of digital signal processing algorithms. Figure 3.3 illustrates the same event captured two ways directly from the relay and displayed by the vendor's display program [22]. The first capture uses 4 samples/cycle for digitally filtered data and the second uses 16 samples/cycle of the sampled data with no special filter. It can be noticed that the filtered data lags behind the sampled data (this time delay is the result of the filter) and does not show the dc offset. Filtered value does not show the true waveform (e. g. peak current). This may be acceptable for a relay under certain conditions, but can influence FL estimation if not taken into account.

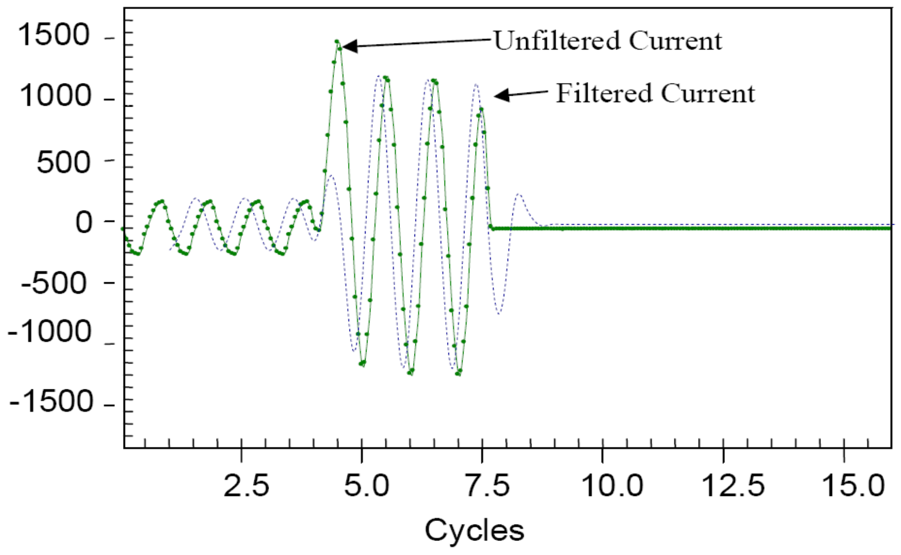


Figure 3.3: Unfiltered vs. filtered data [22]

Sampling rate

The sampling rate of the recording device impacts the accuracy of the data captured for later analysis. DFRs generally use the same time interval between samples, without respect to the actual system frequency. In some DFRs, an adaptive sampling rate may be applied. In this case the sampling rate will vary depending whether it was triggered to record the event of a system disturbance such as a power swing, or transmission line fault.

DPRs generally use a fixed sampling rate that varies with the system frequency. To increase accuracy of digital measurements DPRs track power system frequency to maintain a constant number of samples per cycle. This may result in variable spacing between the recorded samples. Some DPRs can be set to sample and record at a constant sampling rate, then re-sample the actual samples in the software to maintain a constant number of samples per cycle for protection and metering functions.

Measurement window

In order to capture transient waveform samples IEDs may use a concept of measurement window. This means that at minimum the first sampled value in the first window at the beginning of the record is not accurate (ramp up value). Likewise, the last sampled value in the last window of a transient waveform record is not accurate (ramp down value). Depending on the sample rate and window length, it is recommended that the first two (or more) samples at the beginning of the record be ignored and the last two (or more) samples at the end of the record be ignored for each analog input quantity. This makes slower sampling rates less desirable because: a) during analysis of all analog quantities, it is necessary to ignore the ramp up time of the quantity and the ramp down time of the quantity and b) if data is analyzed from more than one device, it is necessary to keep up with the ramp up and ramp down times for the different devices and use the proper one with the proper device. The use of faster sampling rates minimizes the ramp up and ramp down times, therefore making it no longer necessary to ignore these times in the analysis [22].

Here it is important to notice that in many one end FL algorithms they make assumption that fault resistance is negligible assuming that arc is likely to be short during the first cycle when relay is expected to react. In order to evade error that might be caused due to measurement window relays with high sampling rate should be used in these situations.

3. Retrieval of the Power Grid Switching Status

Power grid topology (switching status) describes connectivity of various components in a power system. Beside event recordings, FL algorithms may require knowledge of the switching status in order to process retrieved fault event measurements. This knowledge is related to a specific position from which the waveforms and contacts are measured and the information how the measurement positions were interconnected at the time of the fault occurrence. Therefore, the system topology must be known. Some FL algorithms use information about transmission line length and impedance, while some use power flow information in addition. Beside the network connectivity it is necessary to obtain information about power system component characteristics at a specific moment in time.

Most utilities have power system model information in a format that is defined by the PSS/E or some other Short Circuit program [19]. For example, using PSS/E program, status of the equipment can be updated according to the obtained real time status values. As said before, operator is able to track these changes in real time using Supervisory Control and Data Acquisition (SCADA) system. Through SCADA database, low-speed recording typically used to capture short term and long term disturbances is available. Captured data is typically at a rate between 2 times per cycle and 1 every 2 cycles and it is usually phasor or RMS data, not sampled data. Example of SCADA scans are shown in Figure 3.4. It shows a series of the 30 second data scans from SCADA remote terminal units over the period of 1 hour.

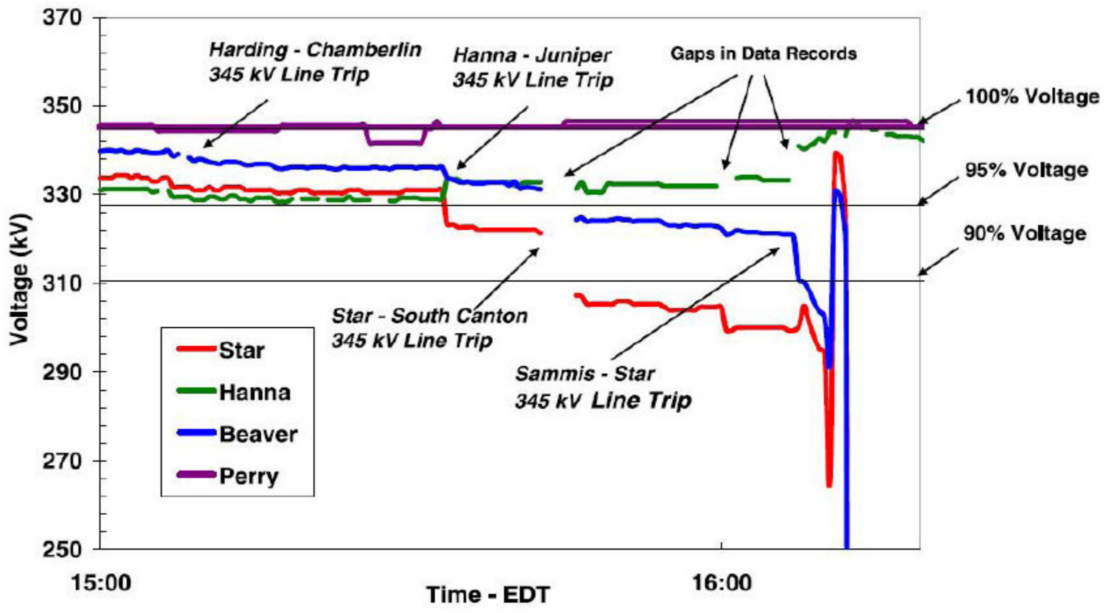


Figure 3.4: 30 seconds scan of SCADA data from 4 locations [22]

It is important to notice that SCADA database is fed by Remote Terminal Units (RTUs). The modern RTUs can be very sophisticated recording instruments that may have a recording performance of a DFR, and at the same time may produce a variety of pre-calculated quantities. However, RTUs are primarily designed to interface EMS SCADA database using mostly customized communication protocols and database formats [23]. Basically it is not possible to access recorded data before it is sent to a centralized location. The only way to retrieve those data beside SCADA system, which is available only in real time in a centralized location, is to use the archive. In order to retrieve pre fault power grid switching status during fault investigation EMS PI Historian data could be used. The PI Historian is capable of retrieving the load, branch and generator data scan in a period before and after the fault.

It is important to mention that information about system topology is extracted from circuit breaker (CB) contact signals which are available through SCADA database. The test results that were executed on Westinghouse R3 CB showed that contact noise can be detected in 70% of the total test cases [24]. This is due to the corrosion on the contacts, a common phenomenon in aged CBs. The fact that the B contact, one of two contacts available though SCADA, in 4 cases out of 6 has a bounce problem, and in 2 cases has excessive noise during the contact transition, leads to conclusion that by analyzing only contacts CB status in not reliably determined. To monitor CB status, a Circuit Breaker Monitor (CBM) device capable of recording signals from CB control circuit is introduced [24]. This makes possible evaluation of CB operation and determination of CB status (opened or closed). Main benefit of this approach is that more reliable information about CB status can be extracted and time stamp of each event is available. In the case devices are synchronized, information from different CBs can be compared on the same time scale.

Finally, one of the most important inputs from the power system model needed for FL estimation is transmission line impedance. System model in PSS/E format contains these values. Methods used for calculation are [3]:

- based on Carson's equations
- making direct measurement of the open and short circuit voltages and currents
- solving 2-port equations based on synchronous phasor measurements obtained from direct measurement from a digital relay or fault recorder.

By combining the above approaches, specially the first and third which do not require special equipment for testing, impedances can be easily obtained. It is important to notice that impedances depend on different factors, temperature, current magnitude, frequency etc [25]. This may cause the calculated impedances to deviate from real ones and this may influence accuracy of FL estimation.

4. Data Transfer and Storage Requirements

In the existing approach fault recordings are retrieved manually, which additionally increases time needed for fault analysis because data from different locations needs to be obtained. Also collected data are not automatically interchanged between different user groups, which makes user groups dependent on each other for the data exchange. In order to solve this problem architecture that uses central repository for data storage is needed.

New automated data analysis application is proposed in [26]. It collects and processes data from three different types of substation IEDs: DFRs, DPRs and CBMs. After local processing, recordings are automatically transferred to central repository. Figure 3.5 shows architecture of this software. In the rest of the thesis we will assume that such repository of DFR and CBM recordings is available to use by the applications. All IEDs should provide their recordings in COMTRADE format [27].

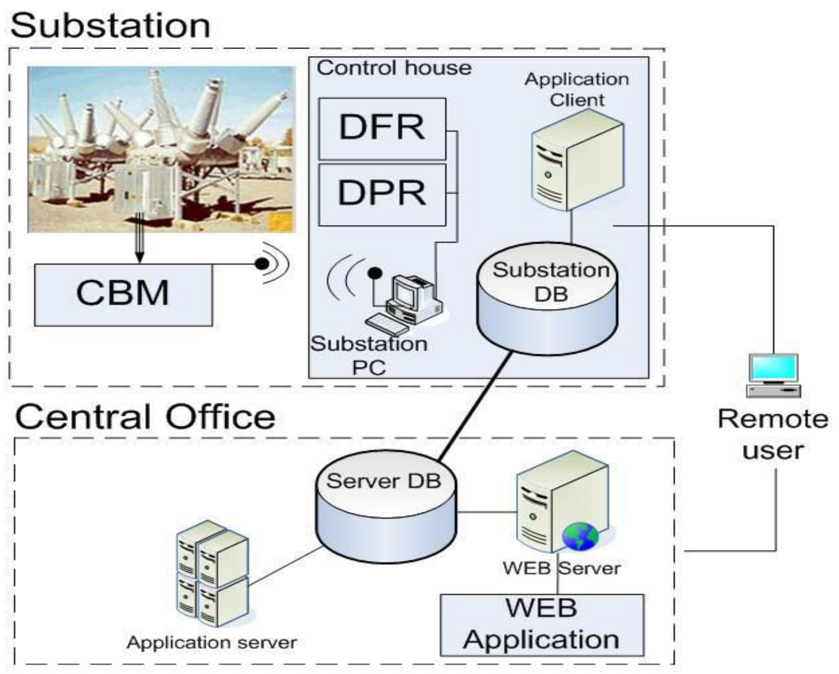


Figure 3.5: Architecture of substation automation software

5. Conclusion

This section discusses sources of input data needed for FL estimation. First, obtaining fault event data was presented. Devices used for collecting fault recordings are classified and possible limitations they might have were explained. Then tools for obtaining power grid switching status are analyzed. Finally, data transfer and storage requirements needed for centralized automated FL application, are presented. The conclusions reached in this section will be taken into account during designing architecture for obtaining data needed for optimal FL algorithm.

CHAPTER IV

NEW APPROACH

1. Introduction

Digital Protective Relays (DPRs) and some disturbance recorders produce a fault location (FL) output, which is available as soon as the record is processed without any manual analysis. The relay and recorder FL information in many cases may not be as accurate and can only serve as an estimate due to in feed, mutual coupling, and non-homogeneous line construction. The dispatcher or protection engineer using this information should be aware of these inaccuracies and learn how to make decisions under the uncertainties. The aim of automatic offline FL analysis is to recognize and compensate corresponding inaccuracies, and reduce outage time. As said before, the research presented in this thesis is reaching this goal by:

- Speeding up FL procedure through automation, where both data retrieval and fault analysis are automated.
- Improving accuracy by applying the most suitable algorithm based on available recordings in vicinity of the estimated fault location.

2. Speeding up Fault Location Procedure Through Automation

Automation of data retrieval process will enable the users to access collected data quickly from different locations as Figure 4.1 shows. Each user can get results as soon as

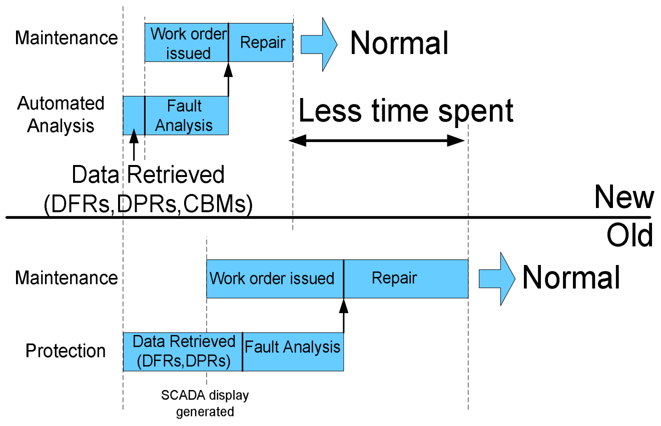


Figure 4.1: Comparison between new and old approach in case of permanent fault

automatic fault analysis is done. As a result any damage caused by the fault is quickly assessed, repaired and restoration is accelerated.

3. Recognizing Faulted Segment

Before any FL algorithm is applied it is necessary to determine whether fault was/is present on the line and estimate possible faulted sections. In Chapter I an existing approach in fault clearing procedure was already described. Part of that procedure is the fact that once relay recognizes the fault, it automatically sends trip command to corresponding Circuit Breakers (CBs). CBs have the purpose to automatically connect or disconnect different parts of the power system in order to isolate the faults and/or re-route the power flow. As already mentioned in Chapter I, by tracking status of CBs not only the faulted line can be revealed, but also the nature of the fault (temporary or permanent) can be determined. To demonstrate this we will consider small part of the network shown in Figure 4.2. Sequence of CB events seen from the substation on the left hand side when temporary fault is present on Line3 is shown on Figure 4.3 [28].

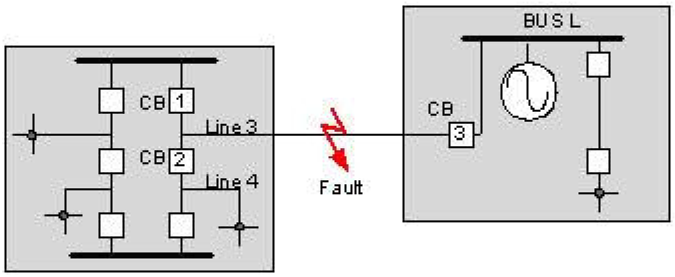


Figure 4.2: Example of fault present on transmission line

The proposed approach assumes that retrieving CBM device data at system level is feasible. In that case reliable information about CB status is obtained by the analysis and used for determining faulted section as proposed in [28].

Fault clearing is usually done by the bay equipment. It is not only important that the bay equipment operates individually as needed, but also that all the equipment functions in a coordinated way. Any abnormality in this procedure could lead to failure or could increase probability that next time the protection equipment will misoperate. Since CBM

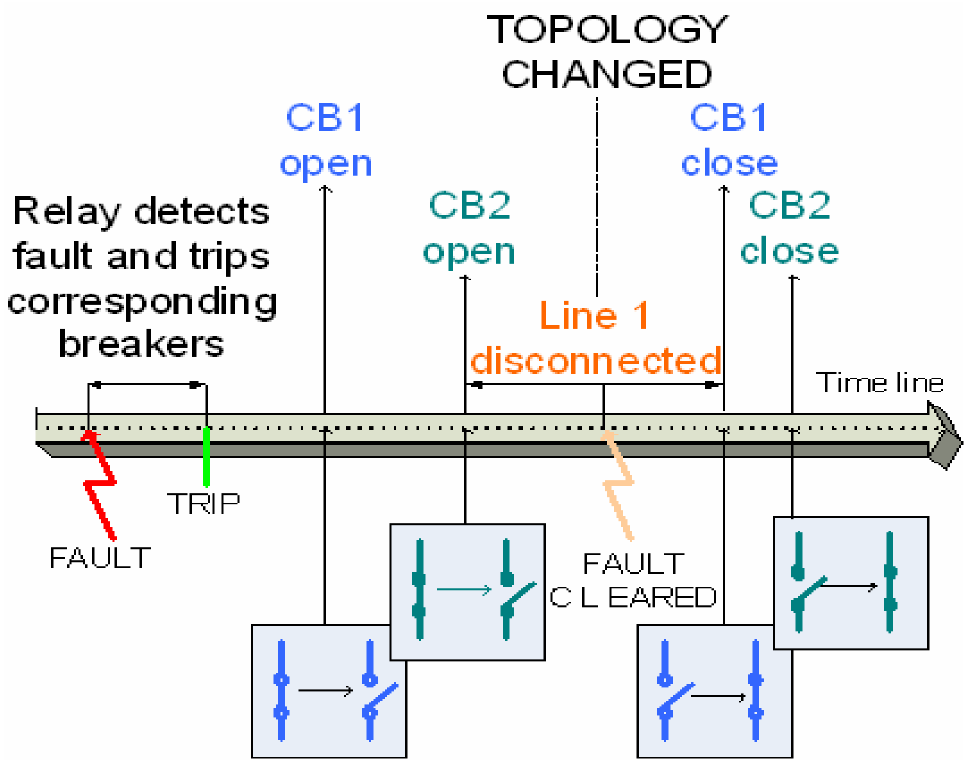
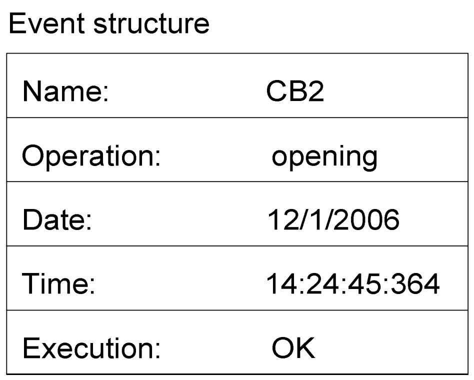


Figure 4.3: Sequence of circuit breaker events when fault is present

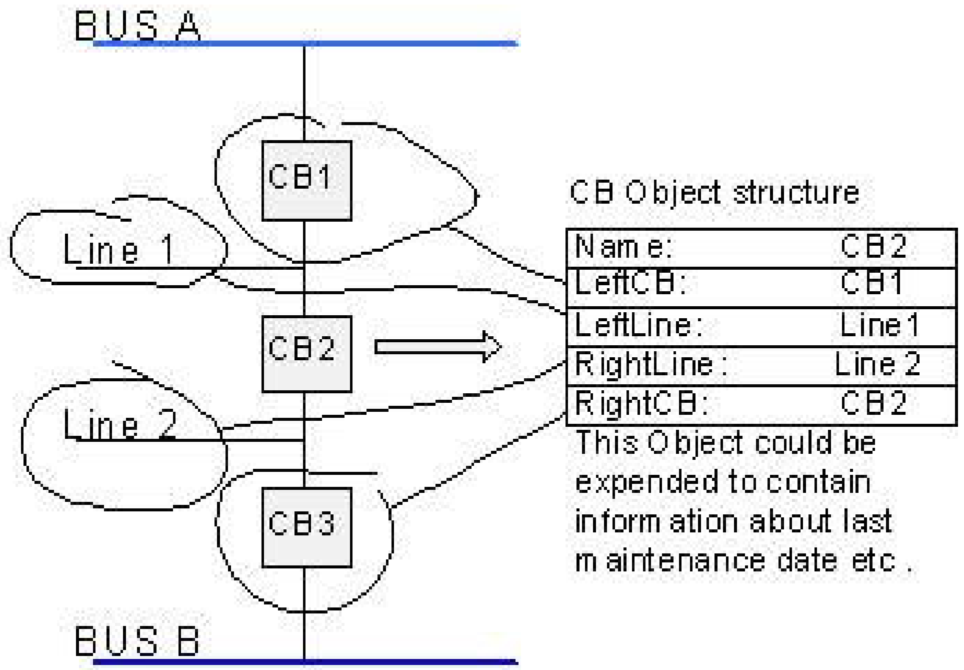
units are synchronized, extracted information is correlated in time domain. It shows: name of CB that was operated, description of operation (opening or closing), date, time of event, and description of execution (successful or not). Example is shown on Figure 4.4.


Event structure

Name:	CB2
Operation:	opening
Date:	12/1/2006
Time:	14:24:45:364
Execution:	OK

Figure 4.4: Extracted information about event from CBM device

It is possible to connect corresponding event files obtained from CBMs with the substation topology. Objects describing topology should be settled into database. When the topology is needed, it is retrieved from the database which is executed very fast using readily available tools. In the case of a breaker and a half bus topology information related to one CB object contains information as shown in Figure 4.5.



Name:	CB2
LeftCB:	CB1
LeftLine:	Line1
RightLine:	Line 2
RightCB:	CB2

This Object could be expended to contain information about last maintenance date etc.

Figure 4.5: Topology information about CBs that operated

Once the CB event file is available it is associated with CB's topology and sequence analysis is started. Every CB that operates and produces new event file is than matched against an existing one to compare whether they belong to the same sequence. State diagram of this approach is shown on Figure 4.6.

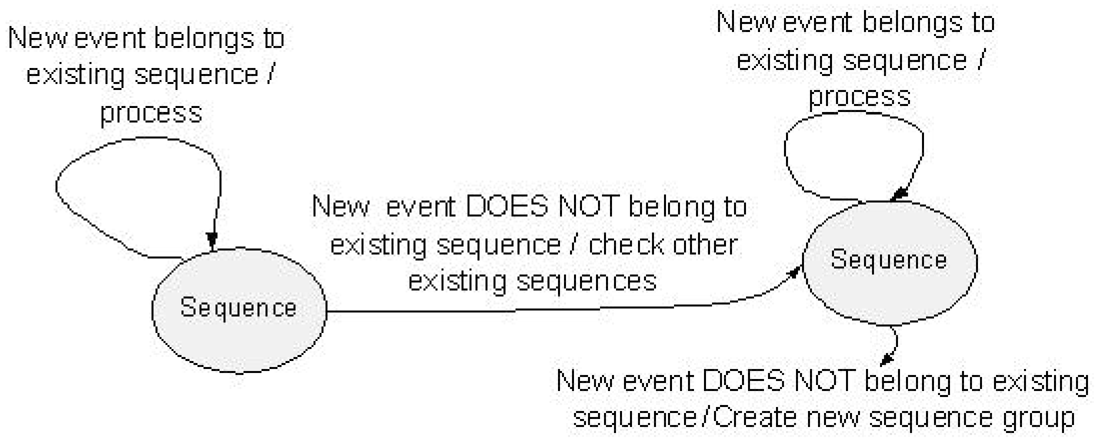

New event belongs to existing sequence / process
Sequence
New event DOES NOT belong to existing sequence / check other existing sequences
New event belongs to existing sequence / process
Sequence
New event DOES NOT belong to existing sequence/Create new sequence group

Figure 4.6: State diagram

Sequence is considered finished when the sequence timeout is achieved. This means that if after some time there was no any new change in the corresponding sequence it is considered finished. It is important to notice that usually 1-2 sequences appear at the same time, so all comparisons are done very fast. Using this approach the transmission line on which the fault is present, quality of sequence execution, and information whether sequence is manual or automatic can be determined. Block diagram of this approach is shown on Figure 4.7. List of branches that changed status as output of this analysis is used to update system model status and determine faulted line or section in the case that fault was present.

Knowledge about the current state of the system topology is very important for many power system applications like state estimation, fault location and alarm processor, which demonstrates the importance of the proposed approach for future improvement of existing tools.

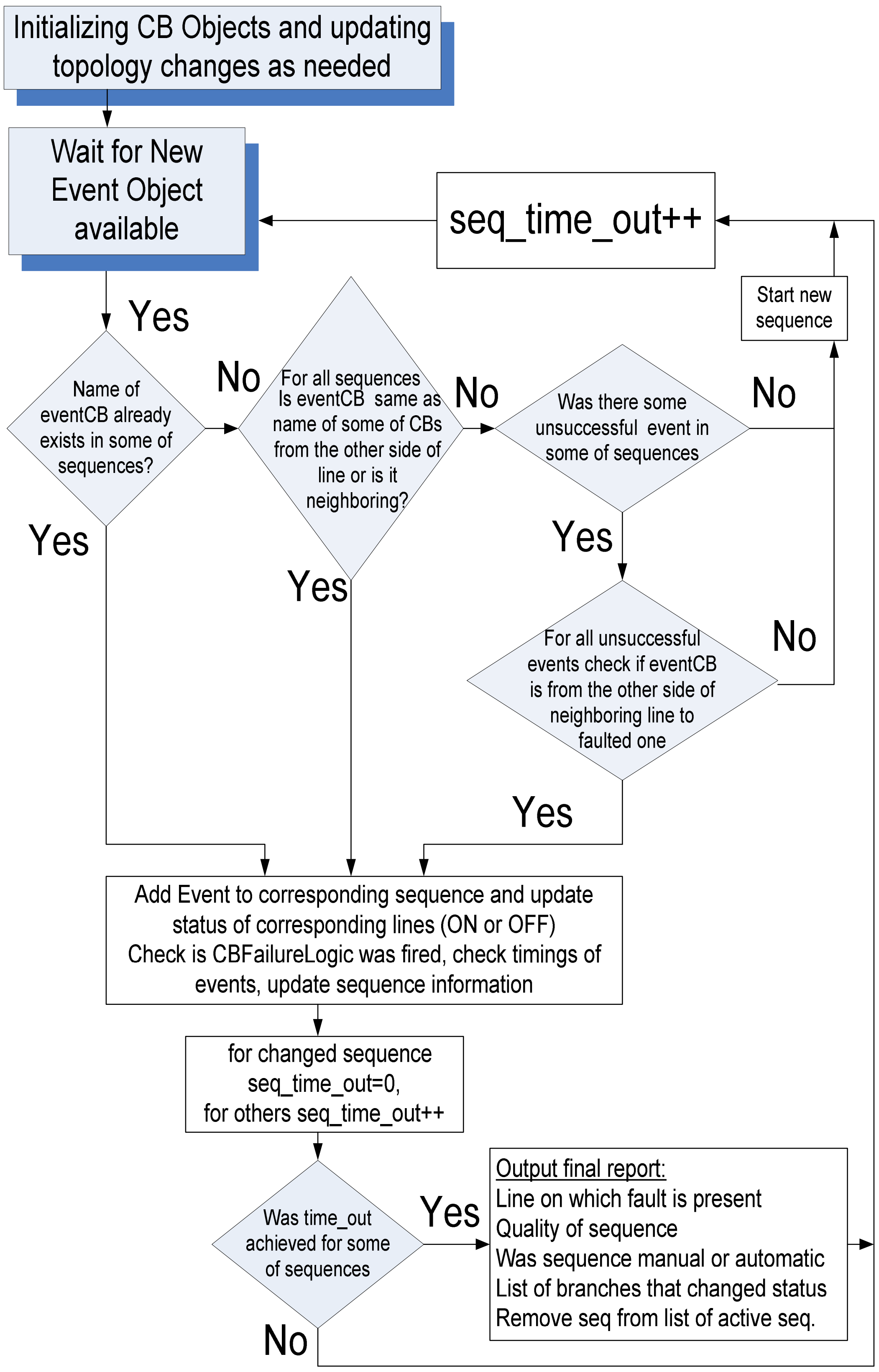

Initializing CB Objects and updating topology changes as needed
Wait for New Event Object available
seq_time_out++
Start new sequence
Yes
Name of eventCB already exists in some of sequences?
No
For all sequences Is eventCB same as name of some of CBs from the other side of line or is it neighboring?
No
Was there some unsuccessful event in some of sequences
No
Yes
Yes
Yes
For all unsuccessful events check if eventCB is from the other side of neighboring line to faulted one
No
Yes
Add Event to corresponding sequence and update status of corresponding lines (ON or OFF)
Check is CBFailureLogic was fired, check timings of events, update sequence information
for changed sequence seq_time_out=0, for others seq_time_out++
Was time_out achieved for some of sequences
Yes
Output final report:
Line on which fault is present
Quality of sequence
Was sequence manual or automatic
List of branches that changed status
Remove seq from list of active seq.
No

Figure 4.7: Block diagram of an algorithm for determining branches that changed status

4. Procedure for Optimal Fault Location Algorithm Selection

As already presented in Chapter II, beside measured fault recordings most FL algorithms need some additional data to meet the assumptions. In order to make correct FL estimation it is necessary to recognize what data are available and what assumptions are satisfied. A procedure for selection of a FL algorithm based on available information is proposed in [29]. This approach only covers two groups of FL techniques: algorithms based on phasors at one line terminal and algorithms that use voltages and currents from all line terminals. Neither the time-domain algorithms that use synchronized samples from two ends of the line nor the most common case when only sparse recordings are available is not covered at all in the proposed approaches. In [30] the authors reported experience based on real cases where Takagi based one-end algorithms [5, 6] and two-end negative sequence algorithm [16] were used for fault estimation. They compared results and shown importance of correct fault type determination, high sampling rate, length of event etc. They also have shown benefit of a two-end algorithm used for testing and generalized conclusion, but no comparison with other two-end algorithms was given. A comparative study was made between different FL algorithms where six one-end [32, 33, 34, 5, 7, 35] and six two-end [36, 17, 10, 37, 38, 13] algorithms were evaluated in [31]. Algorithms were compared with respect to the system infeed, line model used, mutual coupling impact, fault resistance impact. Performance of the methods presented in [31] may be affected by the implementation issues, but it was shown that algorithms that use the same basic principles behave the same.

In the case of system infeed two-end algorithm tests gave better results than one-end methods. For the one-end algorithms for close-in faults, the best results were obtained with the algorithms that approximate current distribution factor [34, 5]. The overall performance of one-end methods was the best for the algorithms with apparent impedance compensation [7, 35]. In the case of two-end algorithms in the presence of system infeed, the distributed parameter methods [17, 37] gave the best results.

In the case when dependence on the line model was tested, the one-end methods that approximate current distribution factor [34, 5] gave similar results showing that they are independent of the line model. The overall accuracy of these algorithms was not so high. Among one-end algorithms greater accuracy was obtained using the algorithms with apparent impedance compensation [7, 35], although they were affected by the line model. In general it was concluded that all the algorithms with lumped parameter models were affected. The two-end distributed parameter methods [17, 37] gave the best results.

During the tests for mutual coupling, all one-end methods were similarly affected, which was expected since all of them considered the line to be homogeneous and transposed. In the case of two-end algorithms it was noticed that their accuracy changed less when compared to the one-end methods. Two-end algorithms that use distributed parameter line model [17, 37] had the best performance again.

Finally, influence of the fault resistance was evaluated. With an increase in fault resistance measured fault currents become smaller which influences all the algorithms. Among the one-ended algorithms the best accuracy was obtained with algorithm considering the apparent impedance approach with compensation by using source impedances [7]. This was expected from the theoretical point of view because this algorithm compensates the most for the reactive component. In the case of the two end algorithms the best result was achieved with the distributed parameter method [17]. Although showing that the performance of FL algorithms varies, testing reported in [31] did not take into account the fact that assumptions made by FL algorithms are not always satisfied. Important analysis would be to show how the accuracy of an algorithm will change if the specific assumption is not satisfied.

In the rest of this section development of the procedure for selecting an optimal FL algorithm that should be used in order to obtain the most accurate estimate of FL is presented. The proposed solution should allow data from different sources to be retrieved and after the processing the most suitable FL algorithm should be decided. Based on Table 2.3 it is analyzed how input data influences the choice of the FL algorithm.

In the case of the one-end algorithms, the method using source impedances proposed in [7] is the most accurate because it compensates for both the errors caused by the current distribution factor and circuit loading factor. This algorithm needs pre fault data and accurate source impedances information from the remote terminal, which are not always available and could be varying. In the case accurate source impedances are not available the method using symmetrical components would be preferable, because it removes influence of mutual inductance, which is very important in the case of parallel lines. The one end algorithm using negative sequence [10] does not need the pre fault data and fault type classification, while method using symmetrical components [8] needs both. In the case of balanced faults the method described in [10] could introduce error due to the use of the negative sequence representation. For symmetric faults when source impedances are not available, one option might be to extend this method using other symmetrical components, so that positive sequence is used in the case of symmetrical faults if pre fault data are available. This way influence of mutual inductance is still removed. The other option is to use the impedance based Takagi method [4] or variation of this method depending whether pre fault data is available. It should be noticed that all the reviewed single end algorithms assume that faulted line is homogeneous.

In the case of two-end approaches, time based method is the most accurate when synchronized samples from two-terminal line are available and sampling rate is sufficiently high [11]. In the case that sampling rate is not sufficiently high, the method that uses distributed line model can give very accurate results [17]. In the case the samples are not synchronized, pre fault data are used for synchronizing phasors in this method. If pre fault data are also unavailable more robust algorithm that uses positive sequence can be used with both, synchronized and unsynchronized samples in the case of two-end lines [14]. This algorithm is immune to fault type, source impedance, fault resistance, mutual coupling and does not need the pre fault data. It uses positive sequence representation of the network. It should be noticed that using negative sequence might be more suitable in the case of unbalanced faults.

In the case of three terminal lines the negative-sequence impedance method [16] could be used for unbalanced faults. Benefit of this algorithm is that it is not computationally complex and it is immune to mutual inductance. It should be noticed that in [30] authors recommended averaging several results obtained using [16]. This algorithm in general might have a problem if the sampling rate is too low or fault is present for a short period. In the case of balanced faults when data from three ends are available the phasor based method that doesn’t use symmetrical components reported in [12] is the best choice. In the case that synchronized samples are available this method can be applied very easy. Since it uses the least square method (LSM) method it is capable to compensate significant noise in the measurements.

Finally, in the case of sparse measurements the waveform matching method using genetic algorithm is the only method capable of estimating FL [18]. Very important feature of this algorithm is that it can work when taps are present on line which is frequently the case in urban areas. This approach may be used to confirm results obtained using single and multi end algorithms, because it can be quite accurate if the faulted line section is known. After above analysis, the procedure for Optimal Fault Location (OFL) algorithm selection is generated. The OFL algorithm selection block diagram is shown on Figure 4.8. Corresponding sub blocks for choosing one-, two- and three- end method are shown on Figure 4.9, 4.10, and 4.11, respectively.

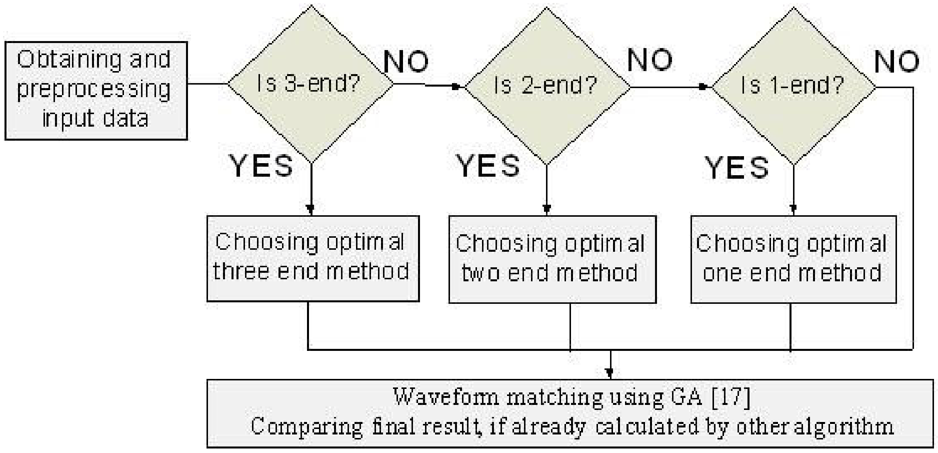

Obtaining and preprocessing input data
Is 3-end?
NO
Is 2-end?
NO
Is 1-end?
NO
YES
YES
YES
Choosing optimal three end method
Choosing optimal two end method
Choosing optimal one end method
Waveform matching using GA [17]
Comparing final result, if already calculated by other algorithm

Figure 4.8: Block diagram of procedure for OFL algorithm selection

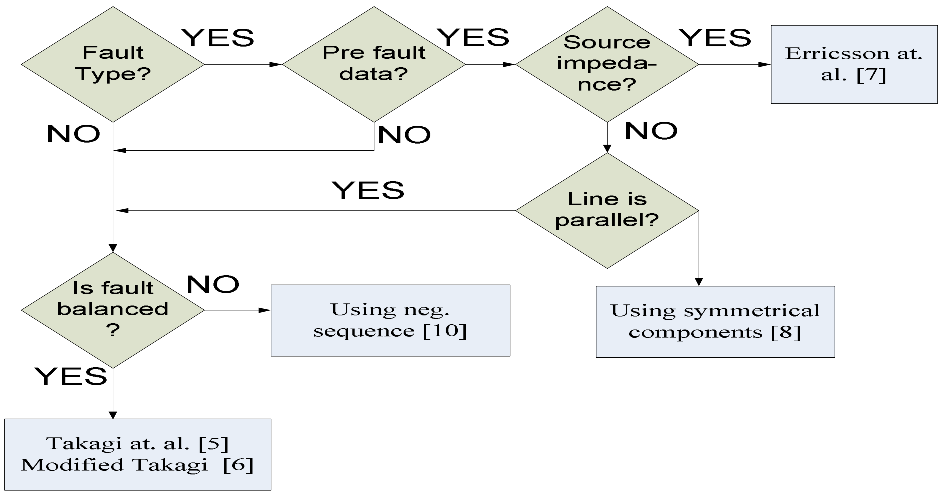


Figure 4.9: Block diagram of one-end optimal selection block

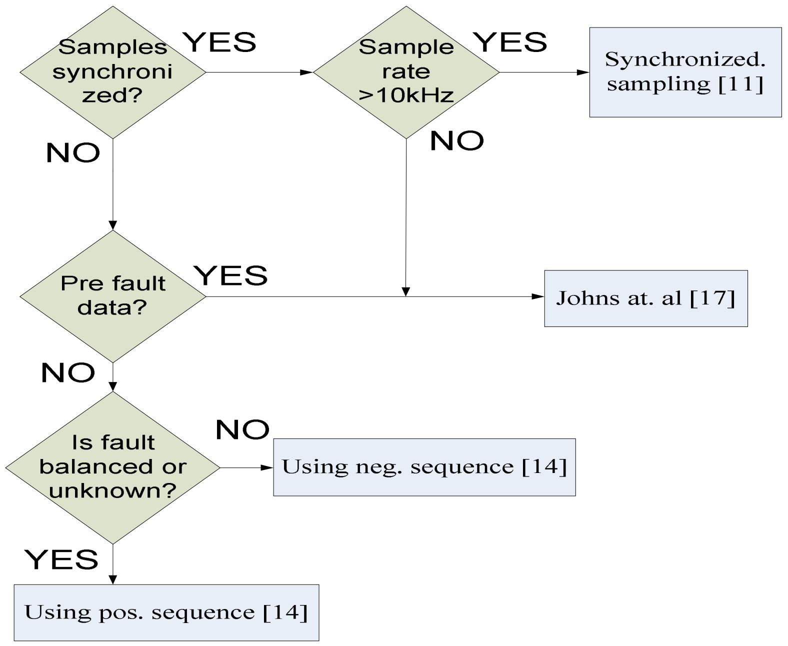


Figure 4.10: Block diagram of two-end optimal selection block

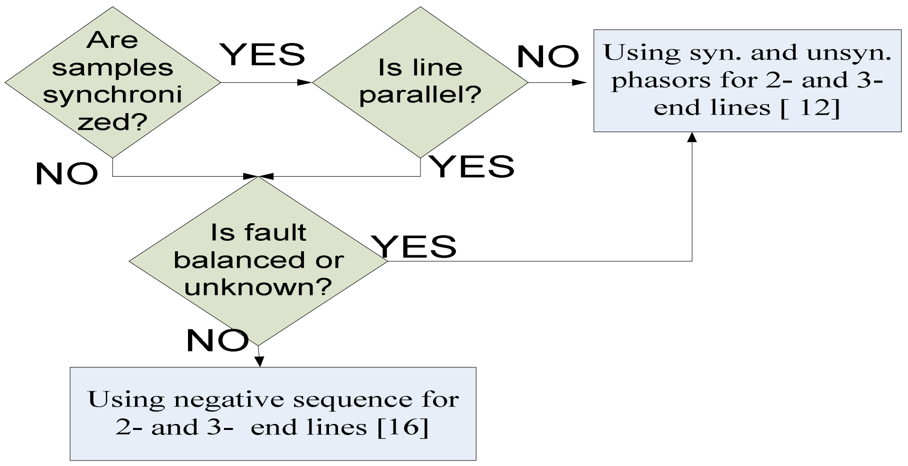

Are samples synchroni zed?
YES
Is line parallel?
NO
Using syn. and unsyn. phasors for 2- and 3- end lines [12]
NO
YES
Is fault balanced or unknown?
YES
NO
Using negative sequence for 2- and 3- end lines [16]

Figure 4.11: Block diagram of three-end optimal selection block

5. Conclusion

This chapter proposes new approaches in order to obtain more accurate FL estimation. First, improvements gained through automation are shown. Then, new approach for determining faulted section is described. This approach is unique in sense that it uses recordings of circuit breakers operation that are correlated in time domain, since recording devices were synchronized. Besides determining whether the fault is present this approach recognizes faulted section, sequence of events etc.

In the last section a procedure for OFL algorithm selection is presented. Few approaches that were taking into account different FL algorithms are commented. Than detail analysis with respect to the advantages and disadvantages of different FL methods is made. Finally, the block diagram of procedure for OFL algorithm selection is generated. The proposed method utilizes 11 FL algorithms in order to compensate for different assumptions and data, algorithms make and use. It is important to notice that most algorithms compute the result in several seconds. The most complicated computation is the waveform matching approach based on genetic algorithm which can last up to few minutes.

CHAPTER V

SYSTEM ARCHITECTURE

1. Introduction

The proposed approach is aimed at running automated fault location analysis. Since this is an off-line analysis, there is no need to estimate Fault Location (FL) in one or two cycle as is the case with relay's action. There is enough time to get measurements from different points in the system and enhance FL process. Centralized system architecture from implementation Figure 5.1 should be used.

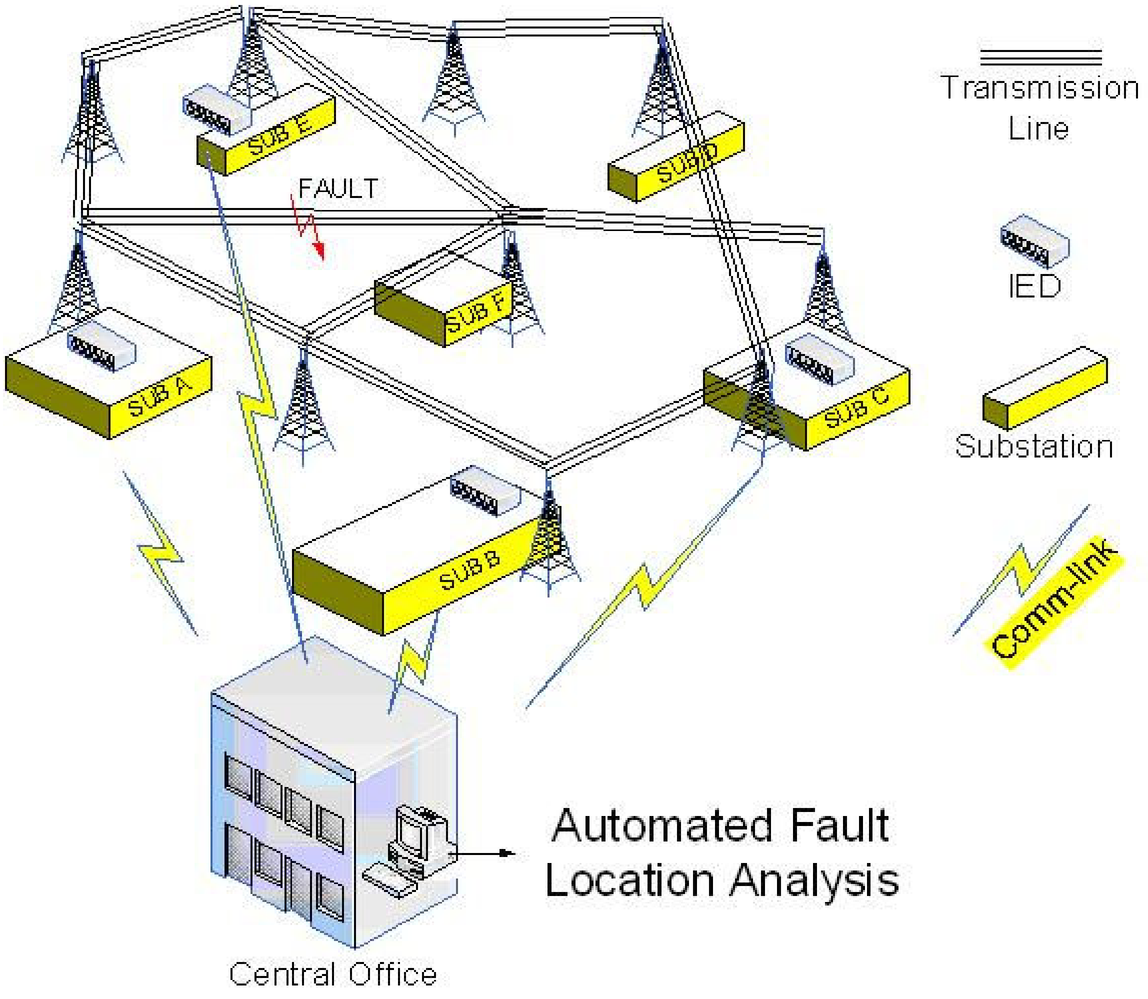


Figure 5.1: System-level architecture of proposed approach

Although this approach could be extended to the use of digital protective relays, phasor measurement units and other IEDs data, at this point it will be assumed that digital fault recorder (DFR) and circuit breaker monitor (CBM) data are the only additional data available in the central repository as described in chapter III (Figure 3.5).

2. Software Architecture

Since the proposed solution should be able to use various algorithms, different data must be provided in order to achieve the optimal performance of each algorithm. The architecture to be used for implementation of waveform matching FL method was proposed in [39]. It utilized only the data from DFRs and system model topology was updated using information about CB status from DFR files, while the generation and load data remained unchanged from one case to another. Architecture of the new approach is shown on Figure 5.2. Two modules may be recognized: fault location and external tools.

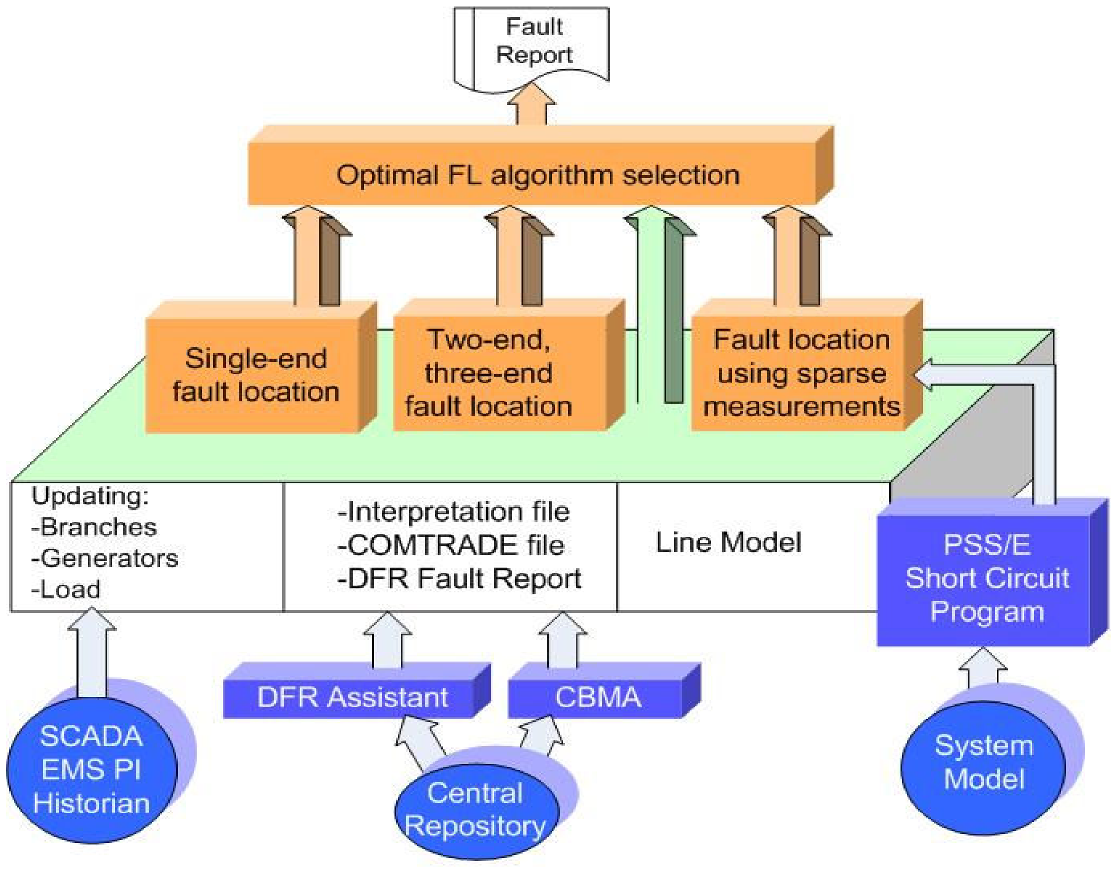


Figure 5.2: Architecture of proposed approach

Fault Location (FL)

FL module should update the power system status based on retrieved data, process new event files, select the most suitable fault location algorithm and execute it. Eleven FL algorithms are used as a possible selection: five one-, three two-, two three- end and one that uses sparse measurements. They are described in the previous chapters, together with the proposed procedure for optimal fault location algorithm selection and it will not be repeated here.

External tools

Since the accuracy of each algorithm is strongly influenced by the accuracy of input data it is necessary to provide reliable and accurate data and preprocessed information as an input to FL algorithms. Different external modules are used and their role in proposed architecture is described bellow:

a) Power system model is available in PSS/E format [19]. This model contains information about connection between system elements, service status of system elements, etc. It is updated before any calculation starts in order to reflect the system state prior to a fault. This is very important feature especially if topological changes take place in the mean time.

b) PSS/E Short Circuit program [19] should be accessed during fault calculation by some algorithms in order to run power flow and short circuit analysis automatically.

c) SCADA EMS PI Historian is used for obtaining the load, branch and generator data in order to update power system model before FL calculation starts.

d) DFR Assistant [40] provides new event recordings from central repository in COMTRADE format [27], as well as the preliminary fault report. Based on the operation of the relays, relay communication channels, and circuit breaker contact signals, an expert system has been developed to carry out the fault analysis [41, 42]. Since DFR assistant performs analysis using data from individual substations, it will generate a corresponding report. In the case that multiple DFRs were triggered due to a fault, DFR assistant will process each of them separately with out correlating whether they belong to the same fault event or not. The analysis report describes behavior of protection equipment, recognizes type of fault, possible faulty line and it is used by other algorithms as an input file. Automated FL procedure should be able to correlate the data in time and space and decide whether they belong to the same fault event. By using circuit breaker status data analysis from the system level, faulted lines could be identified and the selection could be compared with information from the corresponding DFR reports. If the DFR location, estimated faulted line from DFR Assistant report, and the timings of event match with corresponding faulted line recognized using analysis of the CB status data, automated analysis should pick the involved recordings, generated reports and continue with investigation.

e) CBM automated analysis application [24] provides new CB event recordings to a central repository in COMTRADE format [27], as well as the expert system report about the CB event. An application of the system wide data analysis that makes possible to track the circuit breaker switching sequences is demonstrated in [28]. This can be used for determining faulted section as presented in section IV, Figure 4.7. Additionally it is possible to utilize CBM data to determine precisely when the fault happened and tune the system model to that time instance by updating corresponding generator, load and topology values with the data obtained from EMS PI Historian archive.

3. Implementation Techniques for Procedure for Optimal Fault Location Algorithm Selection

Since there are many parameters that influence FL investigation it is necessary to provide easy way for changing the defined rules and parameters during development and testing of the procedure for OFL algorithm selection. To enable that, a decision tree is used for implementing this algorithm. The idea is to implement decision engine using two modules: importing initial data and forming binary decision tree. It can be seen that first module provides the operands and second module defines how the operands are manipulated. In order to make a complete algorithm transparent and easily readable, both modules will be implemented using Extensible Markup Language (XML) [43]. XML provides a text-based means for describing and applying a tree-based structure to the information.

For obtaining initial data used by FL algorithms, different parts of the program are used. For example information about parameters of available recordings is obtained after processing input waveforms, while fault type is read directly from DFR Assitant's fault report. Type of algorithm with respect to the measurements placement (one-, two-, three-end or sparse) is decided after processing system topology, CB status and sources of available recordings. In order to make the initial data transparent to possible changes of how they are retrived and calculated, unique XML object of initial data is implemented.

Next step was to implement easily readable and transposable decision tree. Each node of this tree is represented with operand 1, operand 2, and an operation between them, pointer to the next node if operation is satisfied and pointer to another node if operation is not satisfied. It is important to notice that the names used as input operands for any node correspond to the name of some attributes in initial data object.

Once initial data object and decision tree are created, both initial data object and decision tree will be imported into the program by simply calling the processing engine. The engine will process the decision tree in a binary format node by node until it comes to the last leaf of the tree. Each node is processed by retrieving value of the operand 1 and operand 2 from initial data object and then comparing them according to the specified operation. Depending whether corresponding operation is satisfied or not the next node is chosen. Figure 5.3 demonstrates this approach.

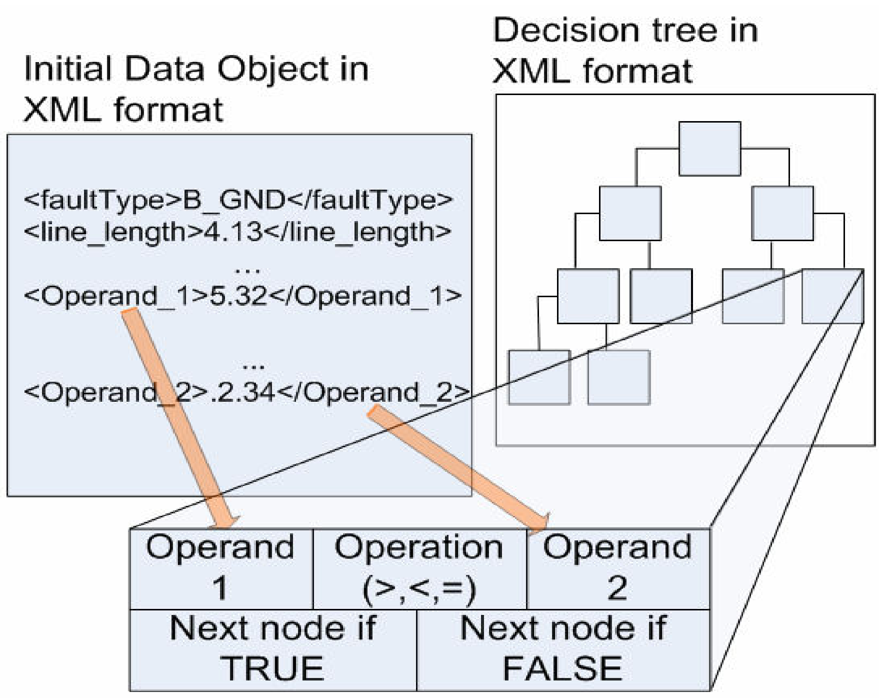

Initial Data Object in XML format
Decision tree in XML format
<faultType>B_GND</faultType>
<line_length>4.13</line_length>
...
<Operand_1>5.32</Operand_1>
...
<Operand_2>.2.34</Operand_2>
Operand 1
Operation (>,<,=)
Operand 2
Next node if TRUE
Next node if FALSE

Figure 5.3: Decision three implementation

It should be noticed that in some cases multiple algorithms are applicable and it would be interesting to check how averaging results from different FL estimations using different weight functions for different algorithms could influence the results.

4. Conclusion

This chapter describes system architecture of the proposed procedure for OFL algorithm selection solution. First, centralized system wide concept that is needed for automated analysis is presented. Then software architecture of proposed FL analysis is described. How the variety of data sources is used during analysis is discussed. Finally implementation of procedure for OFL algorithm selection is discussed. In order to enable easy way for further testing and enhancing procedure for OFL algorithm selection, implementation using decision tree is proposed and it is presented in the last section.

CHAPTER VI

EXPERIMENTAL RESULTS

1. Introduction

During the development of the software, tests on different code segments were first done using simulated fault data. Software was developed using Java programming language [44]. Once the overall automated fault location software prototype was done it was tested again using simulated data. This proved the feasibility of the approach. After fatal bugs were corrected, more extensive tests using real fault data from the field were done to verify the practical value of the algorithm and software. The tests were carried out using the real life transmission system as an example.

2. Evaluation Criteria

Examples of the open issues that should be evaluated are: using varying number of DFR files, specifying the search region, using preprocessed fault location estimation, using different quantities for the match between measured and simulated data, evaluating differences in the accuracy when different input data are available and different assumption are satisfied etc. These different options may produce different results. In order to efficiently analyze the results and find out the most appropriate parameters of FL algorithms it is necessary to carefully plan testing. The following efforts have been identified for conducting test activities:

- *Testing the fault location software using fault data collected from real electric power system*

  The fault location software prototype was developed and set-up for specific electric power system. This system has thirty-three substations equipped with digital fault recorders (DFRs). An automated system capable of processing, analyzing and archiving DFR data is being installed. The short-circuit model of this system has about 5000 buses and 7000 lines. All of this makes this system well suited for comprehensive algorithm/software testing.

- *Analyzing the performance of the fault location algorithm/software and proposing potential improvements*

First activity in this task is to perform a comprehensive analysis of the results obtained during tests using available fault case data. During this analysis, system one-line diagrams, system models and system data are extensively used. Fault location information furnished by a given utility is used as a benchmark.

Second activity is related to identifying the potential improvements in the software. The improvements may be related to the various elements of the fault location procedure: preparation of the input data, data preprocessing, FL algorithm itself, etc. This activity includes a detailed analysis of the steps of the currently implemented solution. Finally, the improvements are recognized and implemented.

3. Test Scenarios

Test activities are related to the data collected from a real life electric power system. Currently, total number of available fault cases is 15. In order to get confidence in the results each case will be run at least five times and corresponding errors will be calculated. After running the same case several times an average error is calculated for each group of settings. Measuring fault location with same settings five times we will call the *measurement cycle* in the future.

Table 6.1: Format of table with results

Case1-1	Err(miles)
Calculated fault location 1	Distance from fault location 1
Calculated fault location 2	Distance from fault location 2
…	
Calculated fault location n	Distance from fault location n
$\frac{\sum Err}{n}$ = average error= average distance from real fault location	
Parameters used as settings for fault location program	

The parameters used as settings for FL algorithms are manually set during testing. It is important to notice that in the future the program should work automatically, which means that is should be able to change settings automatically depending on available information. The results will be systemized and analyzed in order to recognize possible improvements. In this phase, tests are done without the EMS PI Historian and Circuit

Breaker Monitoring (CBM) input because this input for the available fault cases was not furnished by a given utility.

4. Results

Out of 15 tested cases eight of them will be described in detail in this section. For other 7 cases accurate FL was not furnished by utility so it was not possible to evaluate the obtained results. Each test case will start with brief description of the event and then the test results under different FL algorithm parameters and corresponding conclusions will be presented.

Case 1

Event data

Event Date/Time:	7-19-2000, 05:15:34
Event Type:	Phase A-B fault
Fault Location:	Ckt. 84, in Warren substation (Known Fault Location)
Triggered DFRs:	Cedar Bayou , South Channel

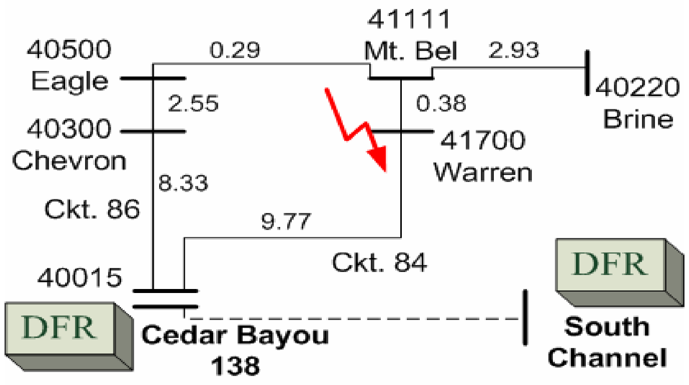


Figure 6.1: Faulted area in case 1

In this case DFR Assistant that is used for recognizing faulted section wrongly marks circuit 86 as one affected. If additional data from SCADA and CBM were available a correct faulted line would have been recognized. Two tests will be run. One comparison of the results will be done when correct circuit 84 is marked as affected and the other comparison will be done when wrong circuit 86 is marked as affected. Each case will be run at least five times and error will be calculated. After running the same case several times, an average error is calculated for each group of settings. Test results are shown in Table 6.2. The selection algorithm properly recognized that sparse measurement algorithm is the only one applicable in this situation.

Table 6.2: Test results – Case 1

Case1-1	Err(from Warvue)
0.018 from Warvue toward 40015	0.018
0.317 from Mt. Bel toward Warvue	-0.063
0.197 from Mt. Bel toward Warvue	-0.183
0.138 from Warvue toward 40015	0.138
0.018 from Warvue toward 40015	0.018
$\frac{\sum Err}{n}$ = 0.0144 from Mt. Bel All V and C matched, Ckt 84 as faulted out_iter 2, n_parent 30, resist 0.8, matching method Phase, matched values Mag	
Case1-2	Err(from Warvue)
0.178 from Warvue toward 40015	0.178
0.26 from Warvue toward 40015	0.26
0.357 from Mt. Bel toward Warvue	-0.023
0.117 from Mt. Bel toward Warvue	-0.263
0.3 from Mt. Bel toward Warvue	-0.08
$\frac{\sum Err}{n}$ =0.0144 from Warvue All V and C matched, Ckt 84 as faulted out_iter 1, n_parent 30, resist 0.8, matching method Phase, matched values Mag	
Case1-3	Err(from MtBel8)
1.544 from Brine_ toward Mt_Bel	-1.39
2.88 from Cedarw toward Chevron	8.29
1.68 from Brine_ toward Mt. Bel	-1.25
1.01 from Chevron toward Eagle	1.61
0.164 from Enprod toward Cities	-5.384
$\frac{\sum Err}{n}$ =0.375 from Mt. Bel $\frac{\sum Err}{n}$ + 0.38(dist Mt. Bel –Warvue)=0.75 miles Only All C matched, Ckt 86 as faulted out_iter 1, n_parent 30, resist 0.8, matching method Seq, matched values Phasor	
0.0354 from Brine→ Err6=-2.895 $\frac{\sum Err}{n}$ =0.2 from Mt. Bel $\frac{\sum Err}{n}$ + 0.38(dist Mt. Bel –Warvue)=0.58 miles	

In the case when correct circuit is assumed as the input parameter, an average error is very small. If we change one parameter of the genetic algorithm, same average values of the error are achieved. In the case when circuit 86 is chosen as being affected, individual results oscillate very much. Error of individual result may be big. If we average those errors, they suppress each other so that the average error becomes very small. This behavior can easily be concluded from Figure 6.2, which visually presents test results for Case1-3 from Table 6.2.

Table 6.3: Comparison of results - Case 1

Ckt 84 as faulted, All V and C matched	Err (miles)
out_iter 2, n_parent 30, resist 0.8, matching method Phase, matched values Mag	0.0144
out_iter 1, n_parent 30, resist 0.8, matching method Phase, matched values Mag	0.0144
Ckt 86 as faulted, Only All C matched	
out_iter 1, n_parent 30, resist 0.8, matching method Seq, matched values Phasor	0.75
	0.58

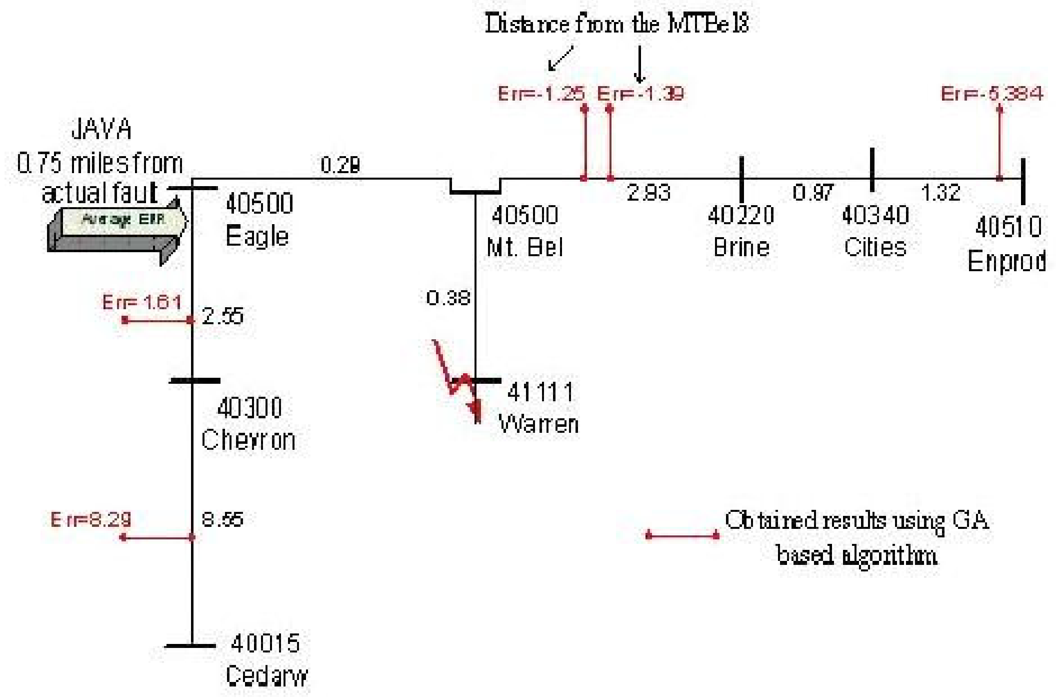


Figure 6.2: Test results when wrong circuit 86 is assumed as faulted

Case 2

Event data

Event Date/Time: 7-18-2000, 20:40:16

Event Type: Phase B-GND fault

Fault Location: Ckt. 21, 3.32 miles from Witter (Known Fault Location)

Triggered DFRs: Greens Bayou 138S.R. Bertron

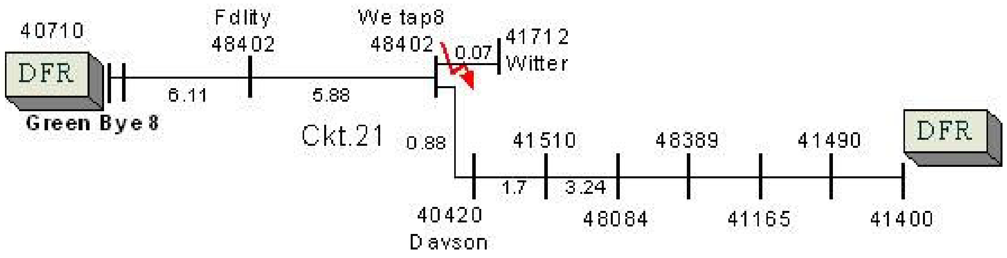


Figure 6.3: Faulted area in case 2

Table 6.4: Test results – Case 2

Case2-1	Err (miles)
1.41 from Fdlity (Green_Bye_8)	3.89
5.68 from Fdlity (WE_Tap_8)	-3.2
0.47 from Davson	3.83
2.66 from Fdlity (WE_Tap_8)	-0.18
0.94 from Fdlity (WE_Tap_8)	1.54

$$\frac{\sum Err}{n} = 1.176 \text{ from fault location (3.32 miles from WITTER substations)}$$

Event 679 and 682, All V and C matched, Ckt 21 as faulted
out_iter 1, n_parent 30, resist 0.8, matching method Phase, matched values Mag

By calculating average value, the measurement cycle error is reduced for 50%. This will be discussed more in case 6.

Case 3

Event data

Event Date/Time: 07-12-2000 20:55:58

Event Type: Phase B-GND fault

Fault Location: Ckt. 94, 3 miles from Metals (Known Fault Location)

Triggered DFRs: Greens Bayou 138

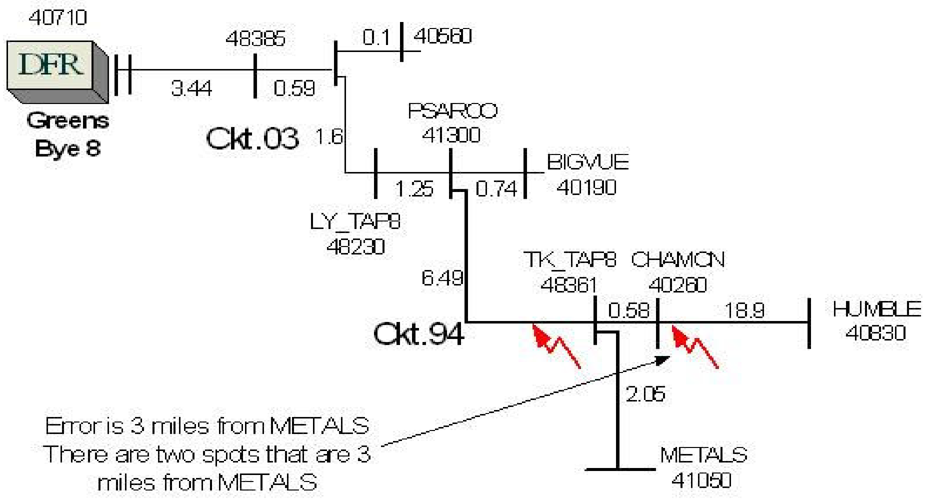


Figure 6.4: Faulted area in case 3

In the case when Ckt 03 is marked as faulted the program shows constantly that fault is around PSARCO bus. That bus connects Ckt 03 and Ckt 94. Results vary consistently in this case which can be seen from Table 6.5. The best results we get are when the fault resistance is 0.4 and additional Ckt 94 is marked as faulted. The results vary quite a bit with a change in fault resistance. It is very important to find an efficient way to automatically tune this parameter from the available fault information in real time.

Table 6.5: Test results – Case 3

Case3-1	Err (assume error 3 miles from Metals)
0.43 from Bigvue	-0.31(Ref: PSARCO)
0.21 from PSARCO	0.21(Ref: PSARCO)
0.24 from Bigvue	-0.5(Ref: PSARCO)
0.73 from Bigvue	-0.01(Ref: PSARCO)
0.57 from Bigvue	-0.17(Ref: PSARCO)
1. $\frac{\sum Err}{n}$ = |-0.056| → Err=|-0.056| + 5.45 from fault location (3 miles from Metals)=5.5 Event 627, All C and V matched, Ckt 03 as faulted out_iter 1, n_parent 30, resist 0.8, matching method Phase, matched values Mag	
Case3-2	Err (assume error 3 miles from Metals)
0.06 from Bigvue	-0.68(Ref: PSARCO)
0.0012 from PSARCO	0.0012(Ref: PSARCO)

Table 6.5(Continued)

0.18 from Bigvue	-0.56(Ref: PSARCO)
7.4 E-5 from Bigvue	-0.74(Ref: PSARCO)
0.335 from Bigvue	-0.405(Ref: PSARCO)
$\frac{\sum Err}{n}$ = |-0.476| → Err=-0.476|+5.45 from fault location (3 miles from Metals)= 4.97 Event 627, All C matched, none V, Ckt 03 as faulted out_iter 1, n_parent 30, resist 0.8, matching method Phase, matched values Mag	
Case3-3	Err (assume error 3 miles from Metals)
0.44 from Chamon	-0.07
3.74 from PSARCO	1.8
0.64 from Chamon	-0.27
2.21 from Chamon	-1.84
0.64 from Chamon	-0.27
1. $\frac{\sum Err}{n}$ = 1.8 2. $\frac{\sum Err}{n}$ =-0.456 Event 627, All V and C matched, Ckt 03 and 94 as faulted out_iter 1, n_parent 30, resist 0.4, matching method Phase, matched values Mag	
Case3-4	Err (assume error 3 miles from Metals)
7.96 from Chamon	-7.59
1.035 from Chamon	-0.665
3.4 from Chamon	-3.03
5.72 from PSARCO	-0.18
2.19 from Chamon	-1.87
1. $\frac{\sum Err}{n}$ = 0.18 2. $\frac{\sum Err}{n}$ =-3.29 Event 627, All V and C matched, Ckt 03 and 94 as faulted out_iter 1, n_parent 30, resist 0.8 , matching method Phase, matched values Mag	

Table 6.6: Comparison of results – Case 3

Ckt 03 as faulted	Err
All V and C matched, out_iter 1, n_parent 30, resist 0.8, matching method Phase, matched values Mag	5.5
All C matched, out_iter 1, n_parent 30, resist 0.8, matching method Phase, matched values Mag	4.97
Ckt 03 and ckt 94 as faulted	Err (miles)
All V and C matched out_iter 1, n_parent 30, resist 0.4, matching method Phase, matched values Mag	1. $\frac{\sum Err}{n}$ =1.8 2. $\frac{\sum Err}{n}$ =-0.456
All V and C matched out_iter 1, n_parent 30, resist 0.8, matching method Phase, matched values Mag	1. $\frac{\sum Err}{n}$ =-2.1 2. $\frac{\sum Err}{n}$ =-3.29

Case 4

Event data

Event Date/Time:	08-23-2000 10:05:50
Event Type:	Phase B-GND fault
Fault Location:	Exxon Ckt. 03, 2.5 miles from SRB 138
Triggered DFRs:	SRB 138 , Cedar Bayou ,South Channel

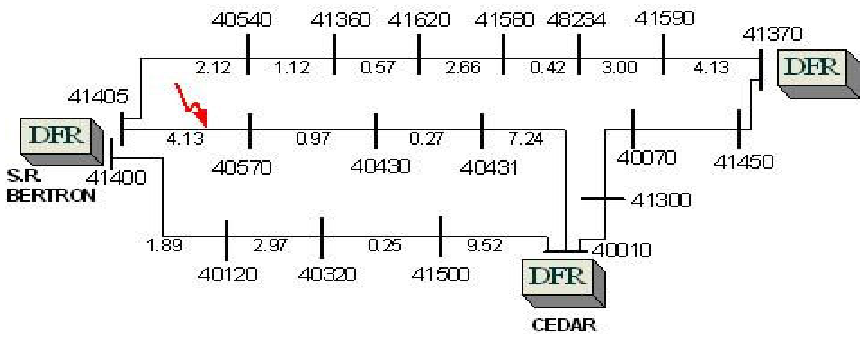


Figure 6.5: Faulted area in case 4

In this case, the fault type is phase B_GND fault. DFR marks correct circuit 03 as the affected one. There are available data from three DFRs. The closest one is 2.5 miles from the fault and there are no taps between closest DFR and fault location. Test results are shown in Table 6.7. Comparison of the two applications is made in Table 6.8 and related conclusions are provided as well.

Table 6.7: Test results – Case 4

Case4-1	Err (assume error 2.5 miles from SRB 138)
2.15 from EXXON	0.51
2.07 from EXXON	0.44
2.31 from EXXON	0.67
1.32 from EXXON	-0.32
1.97 from EXXON	0.34
$\frac{\sum Err}{n}$ = 0.328 from fault location (1.63 from EXXON) Event 627, All C matched, none V, Ckt 03 as faulted out_iter 1, n_parent 30, resist 0.4, matching method Phase, matched values Mag	

Table 6.7(Continued)

Case4-1	Err (assume error 2.5 miles from SRB 138)
1.63 from EXXON	0
3.03 from EXXON	1.4
3.95 from EXXON	2.3
1.79 from EXXON	0.16
1.39 from EXXON	-0.25
$\frac{\sum Err}{n}$=0.728 from fault location (1.63 from EXXON) Event 316, All C and V matched, Ckt 03 as faulted out_iter 1, n_parent 30, resist 0.8, matching method Phase, matched values Mag	
Case4-1	Err (assume error 2.5 miles from SRB 138)
1.39 from EXXON	1.24
1.68 from EXXON	0.05
3.01 from EXXON	1.38
2.57 from EXXON	0.94
2.21 from EXXON	0.58
$\frac{\sum Err}{n}$=0.838 from fault location (1.63 from EXXON) Event 316 and Event 318, All C and none V matched, Ckt 03 as faulted out_iter 1, n_parent 30, resist 0.4, matching method Phase, matched values Mag	
Case4-1	Err (assume error 2.5 miles from SRB 138)
0.47 from EXXON	-1.16
2.7 from EXXON	1.07
1.87 from EXXON	0.23
1.79 from EXXON	0.15
2.85 from EXXON	1.22
$\frac{\sum Err}{n}$=0.302 from fault location (1.63 from EXXON) Event 316 and Event 318, All C and none V matched, Ckt 03 as faulted out_iter 1, n_parent 30, resist 0.8, matching method Phase, matched values Mag	

The best results are achieved when information from two DFRs is used. It is important to notice that adding additional event file from the second DFR, as well as the fault resistance parameter, had the biggest influence on test results. In the case when only one file was used, the results were the same independent of value of fault resistance parameter. Also it should be noticed the best result was achieved when matching only currents is used because voltages were not recorded correctly in this case.

Table 6.8: Comparison of results – Case 4

Ckt 03 as faulted	Err
Event 316, All C, none V matched, out_iter 1, n_parent 30, resist 0.4, matching method Phase, matched values Mag	0.328
Event 316, All C and V matched, out_iter 1, n_parent 30, resist 0.8, matching method Phase, matched values Mag	0.728
Event 316 and Event 318, All C, none V matched, out_iter 1, n_parent 30, resist 0.4, matching method Phase, matched values Mag	0.838
Event 316 and Event 318, All C, none V matched, out_iter 1, n_parent 30, resist 0.8, matching method Phase, matched values Mag	0.302

Case 5

Event data

Event Date/Time:	06-03-2000, 18:29:23
Event Type:	Phase B-GND fault
Fault Location:	Westfield Ckt 81, 2 miles from Rayford toward Rayford tap
Triggered DFRs:	Tomball

In this case, the fault type is phase B_GND fault. DFR marks correct circuit Westfield ckt 81 as the affected one. The model with the taps is taken into account as Figure 6.6 shows, but some model errors might be present as stated by the utility. In this case matching all the currents without matching voltages gives the best results. Like in the previous case this parameter had influence. Error is high and it should be studied further, although it might be related to the fact that system model was not correct.

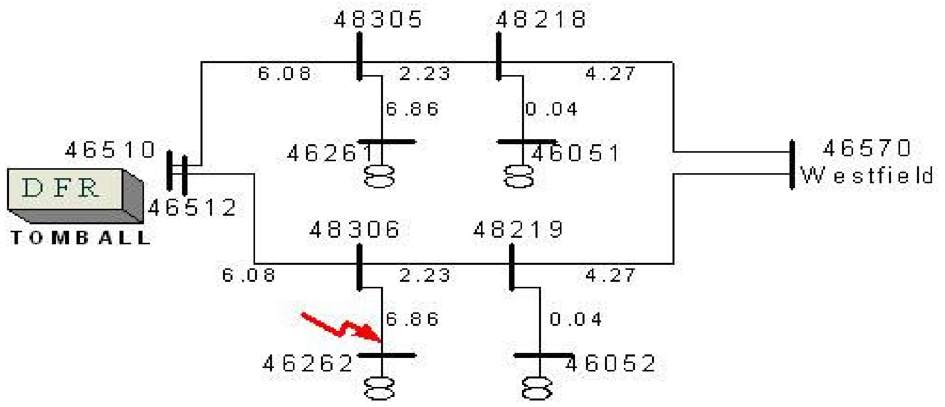


Figure 6.6: Faulted area in case 5

Table 6.9: Test results – Case 5

Case5-1	Err
3.01from Rayford	1.01
3.01 from Rayford	1.01
5.61 from Rayford	3.61
4 from Rayford	2
2.53 from Rayford	0.53
$\frac{\sum Err}{n}$ = 1.632 from fault location (2 miles from Rayford toward Rayford tap) Event 365, All V and C matched, Ckt 81 as faulted out_iter 1, n_parent 30, resist 0.4, matching method Phase, matched values Mag	
Case5-2	Err
1.86 from Rayford	-0.14
2.55 from Rayford	0.55
2.78 from Rayford	0.78
2.02 from Rayford	0.02
4 from Rayford	2
$\frac{\sum Err}{n}$ =0.642 from fault location (2 miles from Rayford toward Rayford tap) Event 365, All C matched, none V, Ckt 81 as faulted out_iter 1, n_parent 30, resist 0.4, matching method Phase, matched values Mag	
Case5-3	Err
0.72 from Rayford	-1.28
3.16 from Rayford	1.16
4.61 from Rayford	2.61
2.02 from Rayford	0.02
4.23 from Rayford	2.23
$\frac{\sum Err}{n}$ =0.948 from fault location (2 miles from Rayford toward Rayford tap) Event 365, All C matched, none V Ckt 81 as faulted out_iter 1, n_parent 30, resist 0.8, matching method Phase, matched values Mag	

Table 6.10: Comparison of results – Case 5

Event 365, Ckt 81 as faulted	Err
All V&C, resist=0.4	1.632
All C, none V, resist=0.4	0.642
All C, none V, resist=0.8	0.948

Case 6

Event data

Event Date/Time:	06-15-2000, 16:12:34
Event Type:	Line_fault (C_GND)
Fault Location:	Westfield Ckt 81, 2.8 mile from Westfield
Triggered DFRs:	Tomball

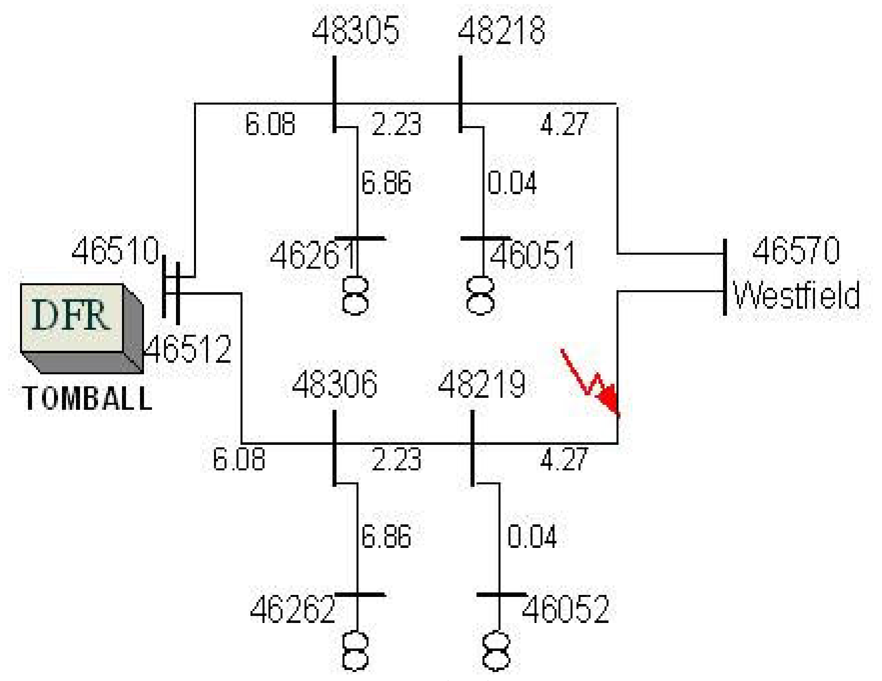


Figure 6.7: Faulted area in case 6

DFR marks correctly the cir. 81 as the affected one. Test results are shown in Table 6.11.

Table 6.11: Test results – Case 6

Case7-1	Err
0.32 from 48219	1.79
0.04 from 48219	1.51
0.17 from 48219	1.64
2.14 from Westfield	-0.66
3.67 from Westfield	0.87
$\frac{\sum Err}{n} = 1.03$ from fault location (2.8 miles from Westfield) Event 474, All C and none V matched, Ckt 81 as faulted out_iter 1, n_parent 30, resist 0.8, matching method Phase, matched values Mag	
Case7-2	Err
2.66 from Westfield	-0.14
3.8 from Westfield	1
2.49 from Westfield	-0.31
3.83 from Westfield	1.03
2.91 from Westfield	0.11
$\frac{\sum Err}{n} =0.338$ from fault location (2.8 miles from Westfield) Event 474, Only branch 48219-46570, Ckt 81 as faulted out_iter 1, n_parent 30, resist 0.8, matching method Phase, matched values Mag	
Case7-3	Err
0.24 from 48219	1.71
2.75 from Westfield	-0.05
3.06 from Westfield	0.26
1.45 from Westfield	-0.55
4.69 from Rayford	5.87
3.67 from Westfield	0.87
2.83 from Westfield	0.03
2.53 from Westfield	-0.29
3.36 from Westfield	0.56
4.05 from Westfield	1.25
1. $\frac{\sum Err}{n} =1.448$ from fault location (2.8 miles from Westfield) 2. $\frac{\sum Err}{n} =0.484$ from fault location (2.8 miles from Westfield) Event 474, All C matched, none V Ckt 81 as faulted out_iter 1, n_parent 30, resist 0.4, matching method Phase, matched values Mag	

A comparison of test results is given in Table 6.12 and conclusions are made.

Table 6.12: Comparison of results – Case 6

	Err
All C noneV, resist=0.8	1.03
All C non V, resist=0.4	1.448
	0.484
Matched only currents from section 48219-46570, resist=0.8	0.338

In the case when many taps are present on the faulted circuit, the results oscillate much. It can be seen from Table 6.11 that within one cycle the results could differ more than 5 miles between each other. It is necessary to mark the faulted section of a circuit ahead of the time. Otherwise the results may be totally random and unpredictable when running a couple of measurement cycles on the case with fixed parameters. The best results are achieved when the specific faulted section is marked Table 6.12 shows.

Case 7

Event data

Event Date/Time:	07-31-2000, 14:57:49
Event Type:	Phase A-GND fault
Fault Location:	Ckt. 44, 66 miles from STP (Calculated Fault Location)
Triggered DFRs:	STP

The line is very long and although there are no taps, the results are not as accurate as it can be seen from Tables 6.13 and 6.14. Even averaging the results did not help in this case. The reasons for the inaccuracy are unknown and this case should be studded more. It can be noticed that results oscillate very much and in the best case an average error is about 7 miles. It should be noticed that we get better average error when fault resistance is set to be 0.8 and as the matching method we use phasor and sequence values. Individual results in one measurement cycle in this case differ for more then 20 miles. This is not acceptable.

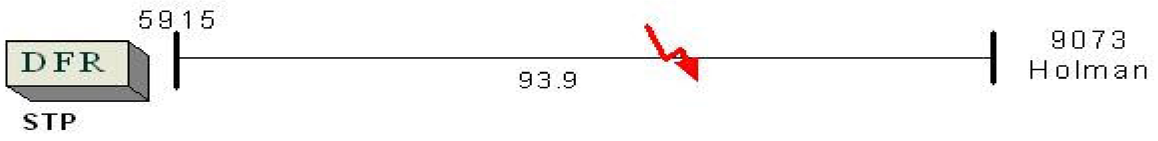


Figure 6.8: Faulted area in case 7

Table 6.13: Test results – Case 7

Case9-1	Err (error 66 miles from STP)
38.66 from STP toward Holman	27.34
22.46 from STP toward Holman	43.54
14 from STP toward Holman	52
2.57 from STP toward Holman	63.42
5.15 from STP toward Holman	60.85
$\frac{\sum Err}{n}$ = 49.43 from fault location (66 miles from STP) Event 805, All C matched, none V, Ckt 44 as faulted out_iter 1, n_parent 30, resist 0.4, matching method Phase, matched values Mag	
Case9-2	Err (error 66 miles from STP)
20.25 from STP toward Holman	45.75
12.94 from STP toward Holman	53.06
36.82 from STP toward Holman	29.18
11.78 from STP toward Holman	54.22
15.83 from STP toward Holman	50.17
$\frac{\sum Err}{n}$ =46.476 from fault location (66 miles from STP) Event 805, All C and none V matched, Ckt 44 as faulted out_iter 1, n_parent 30, resist 0.8, matching method Phase, matched values Mag	
Case9-3	Err (error 66 miles from STP)
64.072 from STP toward Holman	1.928
51.18 from STP toward Holman	14.82
5.15 from STP toward Holman	60.85
30.19 from STP toward Holman	35.81
23.57 from STP toward Holman	42.43
90.21 from STP toward Holman	-24.21
67.39 from STP toward Holman	-1.39
81.01 from STP toward Holman	-15.01
48.607 from STP toward Holman	17.39
9.2 from STP toward Holman	56.8
1. $\frac{\sum Err}{n}$ =31.16 from fault location (66 miles from STP) 2. $\frac{\sum Err}{n}$ =6.7 from fault location (66 miles from STP) Event 805, All C and none V matched, Ckt 44 as faulted out_iter 1, n_parent 30, resist 0.4, matching method Seq, matched values Phasor	
Case9-4	Err (error 66 miles from STP)
56.34 from STP toward Holman	9.66
10.31 from STP toward Holman	55.69
47.87 from STP toward Holman	18.13
90.954 from STP toward Holman	-24.954

Table 6.13 (Continued)

18.04 from STP toward Holman	47.96
45.29 from STP toward Holman	20.71
28.72 from STP toward Holman	37.28
92.79 from STP toward Holman	-26.79
19.15 from STP toward Holman	46.85
81.011 from STP toward Holman	-15.011
1. $\frac{\sum Err}{n}$ = 21.29from fault location (66 miles from STP) 2. $\frac{\sum Err}{n}$ =12.6 from fault location (66 miles from STP) Event 805, All C and none V matched, Ckt 44 as faulted out_iter 1, n_parent 30, resist 0.8, matching method Seq, matched values Phasor	

Table 6.14: Comparison of results – Case 7

	Err
Phase and Mag, resist=0.4, Ckt 44	49.43
Seq and Phasor, resist=0.4, Ckt 44	31.16
	6.7
Phase and Mag, resist=0.8, Ckt 44	46.476
Seq and Phasor, resist=0.8, Ckt 44	21.29
	12.6

In this case the DFR Assistant made an error of only 0.5 miles. This clearly shows benefits when one using the most suitable algorithm for a given case. The optimal selection algorithm recognized that one-end fault location algorithm should be used. After this method was used the obtained error was 0.8 miles.

Case 8

Event data

Event Date/Time:	08-04-2000, 12:53:56
Event Type:	Phase B-GND fault
Fault Location:	Ckt. 95, 3.8 miles from Hardy toward Bertwood
Triggered DFRs:	Greens Bayou

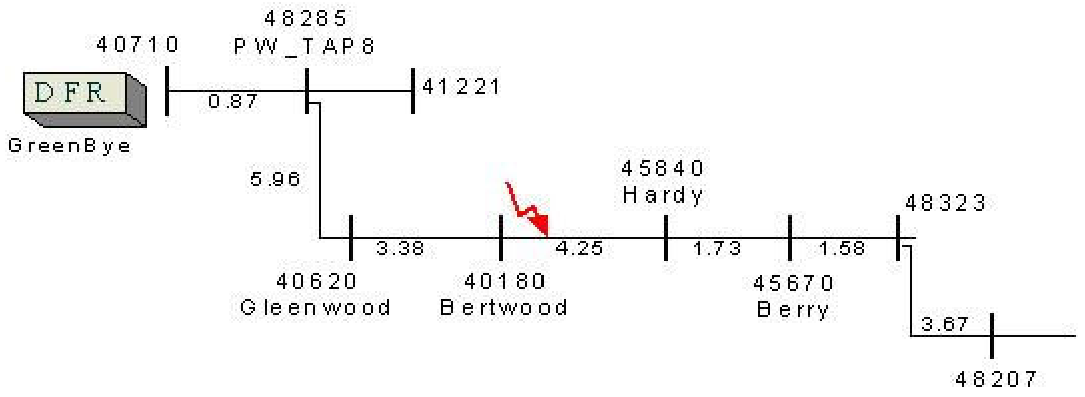


Figure 6.9: Faulted area in case 8

Table 6.15: Test results – Case 8

Case10-1	Err (error 3.8 miles from Hardy)
2.05 from Bertwood toward Gleenwood	2.5
0.04 from Gleenwood toward Parkway	8.12
0.65 from Bertwood toward Gleenwood	1.1
0.00 from Bertwood toward Gleenwood	0.45
0.28 from Bertwood toward Gleenwood	0.73
1.12 from Bertwood toward Gleenwood	1.57
$\frac{\sum Err}{n}$ = 2.41 from fault location (0.45 from Bertwood) Event 860, All C matched, none V, Ckt 95 as faulted out_iter 1, n_parent 30, resist 0.4, matching method Phase, matched values Mag	
Case10-2	Err (error 3.8 miles from Hardy)
2.52 from Bertwood toward Gleenwood	2.97
2.99 from Bertwood toward Gleenwood	3.44
1.12 from Bertwood toward Gleenwood	1.57
0.65 from Bertwood toward Gleenwood	1.1
0.73 from Bertwood toward Hardy	-0.28
$\frac{\sum Err}{n}$ =1.76 from fault location (0.45 from Bertwood) Event 860, All C and none V matched, Ckt 95 as faulted out_iter 1, n_parent 30, resist 0.8, matching method Phase, matched values Mag	
Case10-3	Err(error 3.8 miles from Hardy)
0.6 from Gleenwood toward Parkway	4.43
0.51 from Berry toward Hardy	-5.01
1.45 from Gleenwood toward Parkway	5.28
0.02 from 48207 toward 48323	-10.76
3.54 from Bertwood toward Hardy	-3.09
1.95 from Bertwood toward Hardy	-1.45
2.8 from Bertwood toward Hardy	-2.35

Table 6.15 (Continued)

5.19 from Gleenwood toward Parkway	9.02
$\frac{\sum Err}{n}$=-0.49 from fault location (0.45 from Bertwood) Event 860, All C and none V matched, Ckt 95 as faulted out_iter 1, n_parent 30, resist 0.4, matching method Seq, matched values Phasor	
Case10-4	Err (error 3.8 miles from Hardy)
0.36 from Bertwood toward Hardy	0.09
0.84 from 46002 toward 48207	-12.95
4.72 from Gleenwood toward Parkway	8.55
2.71 from Bertwood toward Gleenwood	3.16
5.38 from Gleenwood toward Parkway	9.21
$\frac{\sum Err}{n}$ = -1.612 from fault location (0.45 from Bertwood) Event 860, All C and none V matched, Ckt 95 as faulted out_iter 1, n_parent 30, resist 0.8, matching method Seq, matched values Phasor	

Table 6.16: Comparison of results – Case 8

	Err
Phase and Mag, resist=0.4, Hardy Ckt 95	2.41
Seq and Phasor, resist=0.4, Hardy Ckt 95	-0.49
Phase and Mag, resist=0.8, Hardy Ckt 95	1.76
Seq and Phasor, resist=0.8, Hardy Ckt 95	-1.612
Only Bertwood-Hardy section used for matching, Phase and Mag, resist=0.4	-0.176

Once again the best results are achieved when fault section of the circuit is marked ahead of the time. This only confirms that it is very important to know a relatively narrow part of the circuit where the fault could be in order to apply FL method that uses GA.

5. Conclusion

From the evaluation it can be seen that in most cases only sparse measurement FL algorithm is applicable. Since other inputs beside the DFR Assistant files are not available, it can not be estimated how much the results are affected by an inaccurate system model. In few cases it is shown that results are drastically improved if the faulted section, especially if many taps are present on the line, is known. When the faulted section is known the accuracy of this method is 0.2 miles in some cases. In most of cases averaging the results proves to be effective. This could be explained by the fact that the GA algorithm places possible faulted locations randomly at the beginning of the processing. Later on it finds out new locations by crossover and mutation. In the case when calculation is repeated several times, every time different beginning points are generated. This produces greater number of possible faulted points, and causes longer processing. The processing time of this algorithm depends on the number of DFR files used. In the case when two recordings are available it takes about 3 minutes for calculation. This amount is mostly influenced by the need to access the external application (PSS/E Short circuit program) several times during processing. This algorithm is strongly influenced by the assumed fault resistance.

CHAPTER VII

CONCLUSIONS

1. Summary

This thesis proposes improvements in the Fault Location (FL) procedure currently used in practice. It provides solution for obtaining more accurate and robust FL estimation by:

- Applying the most suitable algorithm for the available recordings that correspond to a given fault event, which provides the most accurate results.
- Using automation, which speeds up the centralized FL procedure and makes the program robust and immune to human errors.

The obstacles of current approaches for offline fault analysis are shown in the first chapter. They affect efficiency of utility personnel, as well as speed and accuracy of FL investigation. The criterion used for classifying FL algorithms is introduced and then detailed analyses of FL methods is performed in the second chapter. This analysis results in a table where different features of the algorithms are compared from the theoretical stand point. The sources of input data needed for FL estimation are evaluated in the third chapter. Possible limitations that algorithms might have when subjected to incomplete data are explained. Finally, automatic centralized data retrieval concept is presented. The impact of the automated procedure on the overall performance of fault analysis is explained in the fourth chapter. A detail algorithm for determining faulted line or section is introduced. This approach is unique in a sense that it uses recordings of circuit breaker operation that are correlated in the time domain once the recording devices are synchronized. Besides determining whether the fault is present, this approach also recognizes the faulted section, sequence of events etc. After that a procedure for an Optimal Fault Location (OFL) algorithm selection is presented. Detailed analysis with respect to advantages and disadvantages of the different FL methods is carried out and block diagram of procedure for OFL algorithm selection is generated. The architecture of the new approach is described and the role of proposed external tools is explained in the fifth chapter. A software implementation of the procedure for OFL algorithm selection is also proposed. It utilizes implementation using binary tree so that the algorithm can be easily changed and tested using various scenarios. Finally, test results based on real cases are presented in the sixth chapter. Unfortunately, most of the fault cases are applicable only for the sparse measurement algorithm so it is not possible to evaluate other algorithms using these cases.

2. Contribution

In summary, this thesis has been focusing on automated FL procedure. Two new procedures have been proposed one for determining faulted section based on the data from circuit breaker monitoring devices and the other for selection of the most suitable FL algorithm. Detailed analysis with respect to advantages and disadvantages of the different FL methods is carried out during the development of procedure for OFL algorithm selection. Software prototype of the proposed automated FL analysis is developed using Java programming language. Current implementation of the procedure for OFL algorithm selection is presented in Appendix A. For future implementation of this procedure, binary tree is proposed. Binary tree approach is implemented using decision engine capable of forming and searching a binary tree. Elements of the tree are imported from XML files. This enables procedure for OFL algorithm selection to be easily changed through XML file without any knowledge of programming. An example of this principle is presented in Appendix B. During evaluation of some of the proposed FL algorithms, improvements of existing methods were noticed and implemented. Complete application is implemented to work automatically. Fault location analysis is automatically triggered by appearance of new event files in a specified folder. Once FL is calculated, result is visually shown through newly developed graphical views. Advanced representation of results enables user to notice fault on one line system diagram and access corresponding views of faulted area and involved equipment.

3. Benefits

The following is a summary of the benefits achieved with this solution:

- System operators: Their main tool today is the SCADA system. Additional information used in the proposed approach is automatically obtained using additional data from substation IEDs. This speeds up the decision made by operators in restoring the system.

- Protection engineers: Instead of spending a lot of time processing IED data manually, this group will be unburdened from the routine analysis tasks that are performed automatically and they will able to concentrate on complicated cases that require their involvement.
- Maintenance staff: Automatic analysis immediately provides the information about the fault location so that this group is able to take some actions right away instead of waiting for instructions from other groups. This significantly reduces the time spent on fault repair and system restoration.

4. Future Work

Modular structure of the proposed approach makes it possible to easily extend architecture with new techniques as they become available. It is very important to notice that many additional data are measured all over power system and by automated collection of data into central repository redundancy and accuracy of input data can be drastically improved. Examples of additional data that could be incorporated are data from DPRs, PMUs, metering devices, etc. Although proposed in the new approach, the SCADA EMS PI Historian and circuit breaker monitoring data are still not available, so they were not utilized in the current software prototype. It is expected that this data will strongly influence accuracy of waveform matching algorithm due to the ability to update system model by using data from the mentioned tools. Test results show that in almost all real life cases that were available only the waveform matching algorithm was applicable. That is a reason why additional cases should be simulated using ATP or some other tools in order to test one-, two- and three- end FL algorithms. Only few one- and two- end FL algorithms were developed in the current version of software, while proposed procedure for OFL algorithm selection utilizes eleven algorithms.

Some non conventional improvements would be achieved using lightning detection data (LDD) [45]. A lightning detection system uses GPS for time synchronizing, which makes it possible to use LDD data to improve analysis of the timing associated with the fault clearing sequence. In addition, enhancements in the proposed FL location approach could be achieved by providing user critical information through visualization. Examples of this activity would be providing visual information about FL of faulted area using satellite image or GIS [46]. An example of applying the GIS is showing the fault events archive data using geographic images. This would reveal locations in system where faults are more frequent, which means the equipment will be more stressed and be more likely to be a cause for presence of permanent faults.

REFERENCES

[1] J. L. Blackburn, *Protective Relaying: Principles and Applications*, 2nd edition. New York: Marcel Dekker, 1998.

[2] M. Kezunovic, B. Perunicic, *Fault location: Wiley Encyclopedia of Electrical and Electronics Terminology*, vol. 7, pp. 276-285, New York: John Wiley, 1999.

[3] IEEE Inc., "IEEE Guide for determining fault location on AC transmission and distribution lines," *IEEE Std. C37.114-2004*, Jun 2005.

[4] M. Kezunovic, "Monitoring of power system topology in real-time," *Hawaii International Conference on System Sciences HICSS-39*, Poipu, Kauai, Jan 2006, pp.244b-254b.

[5] T. Takagi, Y. Yamakoshi, M. Yamaura, R. Kondow, and T. Matsushima, " Development of a new type fault locator using the one-terminal voltage and current data," *IEEE Trans. on Power App. & Sys.*, vol. PAS-101, no. 8, pp. 2892-2898, Aug 1982.

[6] E. O. Schweitzer, III, "Evaluation and development of transmission line fault locating techniques which use sinusoidal steady- state information," in *Pro. of the 9th Annual Western Protective Relay Conference*, Spokane, WA, Oct 26-28, 1982, pp. 1-10

[7] L. Ericsson, M. Saha, and G.D. Rockefeller," An accurate fault locator with compensation for apparent reactance in the fault resistance resulting from remote-end in feed," *IEEE Trans. Power App. & Sys.*, vol. PAS-104, no. 2, pp. 422-436, Feb 1985.

[8] D. L. Waikar, A. C. Liew, and S. Elangovan, "Design, implementation and performance evaluation of a new digital distance relaying algorithm," *IEEE Transactions on Power Systems*, vol. 11, no. 1, pp. 448 – 456, Feb 1996.

[9] Y. Liao and S. Elangovan, "Digital distance relaying algorithm for first zone protection for parallel transmission lines," *IEE Proceedings-Part C: Generation, Transmission and Distribution*, vol. 145, no. 5, pp. 531 – 536, Sep 1998.

[10] D. Novosel, D.G. Hart, E. Udren, and A. Phadke, "Accurate fault location using digital relays," *ICPST Conference*, Beijing ,China, Oct 1994, pp. 1120-1124.

[11] M. Kezunovic, B. Perunicic, and J. Mrkic, "An accurate fault location algorithm using synchronized sampling," *Electric Power Systems Research Journal*, vol. 29, no. 3, pp. 161-169, May 1994.

[12] A.A. Gigris, D.G. Hart, and W.L. Peterson, "A new fault location technique for two- and three-terminal lines," *IEEE Trans. on Power Delivery*, vol. 7, no. 1, pp. 98-107, Jan 1992.

[13] M. S**.**Sachdev and R. Agarwal, "A technique for estimating line fault locations from digital impedance relay measurements," *IEEE Transactions on Power Delivery*, vol. PWRD-3, no. 1, pp. 121-129, Jan 1988.

[14] D. Novosel, D.G. Hart, E. Udren, J. Garitty," Unsynchronized two-terminal fault location estimation ," *IEEE Transactions on Power Delivery*, vol. 11, no. 1, pp.130 – 138, Jan 1996.

[15] Y. Liao, and S. Elangovan, "Improved symmetrical component-based fault distance estimation for digital distance protection," *IEEE Proceedings-Generation, Transmission and Distribution*, vol. 145, no. 6, pp. 739 – 746, Nov 1998.

[16] D. A. Tziouvaras, J. Roberts, and G. Benmouyal, " New multi-ended fault location design for two- or three-terminal lines," *Development in Power System Protection*, Amsterdam, Netherlands, Apr 9-12, 2001.

[17] A. Johns, and S. Jamali, "Accurate fault location technique for power transmission lines," *IEE Proceedings*, vol. 137, pt. C, no. 6, pp. 395-402, Nov 1990.

[18] M. Kezunovic, and Y. Liao, "Fault location estimation based on matching the simulated and recorded waveforms using genetic algorithms," *Developments in Power System Protection*, in Seventh International Conference on (IEE), Amsterdam, The Netherlands, pp. 399 – 402, Apr 2001.

[19] Siemens Power Transmission & Distribution Inc., PTI, Schenectady, NY, USA, Online: http://www.pti-us.com/pti/software/psse/sitemap.cfm.

[20] M. Kezunovic, and M. Knezev, "Fault location using sparse IED recordings," in *14th Intelligent System Applications to Power Systems - ISAP 2007*, Koahsiung, Taiwan, Nov 2007, pp. 1-6.

[21] D. J. Lawrence and D. L. Waser, “Transmission line fault location using digital fault recorders,” *IEEE Transactions on Power Delivery*, vol. 3, no. 2, pp. 496-502, Apr 1998.

[22] IEEE Inc., “Considerations for use of disturbance recorders,” *A Report to the System Protection Subcommittee of the Power System Relaying Committee of the IEEE Power Engineering Society*, Dec 2006.

[23] M. Kezunovic, C.C. Liu, J. McDonald, and L.E. Smith, “IEEE tutorial on automated fault analysis,” *IEEE PES*, 2000, pp. 5-9.

[24] M. Kezunovic, Z. Ren, G. Latisko, D.R. Sevcik, J. Lucey, W. Cook, and E. Koch, "Automated monitoring and analysis of circuit breaker operation,” *IEEE Transactions on Power Delivery*, vol. 20, no. 3, pp. 1910-1918, Jul 2005.

[25] J. D. Glover and M. Sarma, *Power system analysis and design*, 2nd edition, Boston, MA: PWS Publishing Company, 1993.

[26] M. Kezunovic, "Future uses of substation data," in *7th International Conference on Advances in Power System Control, Operation and Management - APSCOM 2006*, Hong Kong, Nov 2006, pp. 1-6.

[27] IEEE Inc., 1999 “IEEE standard common format for transient data exchange (COMTRADE) for power systems,” *IEEE Std. C37.111-1999*, Mar 1999.

[28] M. Knezev, Z. Djekic, and M. Kezunovic, ”Automated circuit breaker monitoring,” *IEEE PES GM 2007*, Tampa, FL, Jun 2007, pp. 1-6.

[29] A. Girgis, and M. Johns, ”A hybrid system for section identification, fault type classification and selection of fault location algorithms,” *IEEE Transactions on Power Delivery*, vol. 4, no. 2, pp. 978-985, Apr 1989.

[30] K. Zimmerman, and D. Costello, “Impedance-based fault location experience,” in *2006 IEEE Rural Electric Power Conference*, Albuquerque, NM, Apr 2006.

[31] T. A. Kawady, “Fault location estimation in power systems with universal intelligent tuning,” Ph.D. dissertation in Electrical Engineering , Technical University of Darmstadt (TUD), Darmstadt, Germany, Feb 2005.

[32] M. T. Sant, M. Tech and Y. G. Paithankar, “On line digital fault locator for overhead transmission line,” *IEE Proceedings*, vol. 126, pp. 1181-1185, 1979.

[33] A. Wiszniewski, “Accurate fault impedance locating algorithm,” *IEE Proceedings*, vol. 130, pt. C, pp.331-314, 1993.

[34] T. Takagi, Y. Yamakoshi, J. Baba, K. Uemura and T. Sakaguchi, “A new algorithm for EHV/UHV transmission lines: Part-II Laplace transform method,” *IEEE PES Summer Meeting*, paper number SM 411-8, 1981.

[35] A. Girgis and E. Makram, “Application of adaptive Kalman filtering in fault classification, distance protection and fault location using microprocessors,” *IEEE Trans. on Power Systems*, vol. 3, no. 1, pp. 301-309, Feb 1988.

[36] B. Jeyasura and M. A. Rahman, “Accurate fault location of transmission lines using microprocessors,” in *4th International Conference on Developments in Power System Protection,* , New Delphi, India, 1988, pp. 13-17.

[37] J. Jiang, J. Yang, Y. Lin, C. Liu and J. Ma, “An adaptive PMU based fault detection/location technique for transmission lines. P. I: Theory and algorithms,” *IEEE Transactions. on Power Delivery*, vol. 15, no. 4, pp. 1136-146, Oct 2000.

[38] V. Cook, “Fundamental aspects of fault location algorithms used in distance protection,” *IEE Proceedings*, vol. 133, pt. C, pp.359-366, 1986.

[39] Y. Liao, “Automated analysis of power quality data and transmission line fault location,” Ph.D. Dissertation, Texas A&M University, College Station, 2000.

[40] Test Laboratories International, Inc.: “DFR Assistant - software for automated analysis and archival of DFR records with integrated fault location calculation,” [Online]. Available: http://www.tliinc.com.

[41] M. Kezunovic, I. Rikalo, C.W. Fromen, D.R.Sevcik, S.M. McKenna, D.Hamai, W.M. Carpenter, and S.L. Goiffon “Automated fault analysis using intelligent techniques and synchronized sampling ,” in *Proceedings of the CIGRE General Session, Paris, France*, pp. 1-8, Sep 1998.

[42] M. Kezunovic, “Automation of fault analysis using DFR data ,” in *Proceedings of the 60th American power Conference*, Chicago, IL, pp. 91-98, Apr 1998.

[43] *Extensible Markup Language (XML) 1.0* (fourth edition), W3C Recommendation, http://www.w3.org/TR/xml/, Aug 2006.

[44] Java Programming Language: http://java.sun.com/.

[45] R. E Orville,G. R. Huffines, W. R. Burrows, R. L. Holle, and K. L. Cummins, "The north American lightning detection network (NALDN) - First Results: 1998-2000," *Mon. Wea. Rev*., vol. 130, pp. 2098-2109, 2002.

[46] D. Maguire, M. Batty, and M. Goodchild, *GIS, spatial analysis, and modeling*, 1st edition, Redlands, CA: ESRI Press, 2005.

APPENDIX A

algo.java

```
package FL;

import java.io.IOException;
import java.util.List;
import java.io.File;
import GA.*;
import Interface.DfrInfo;
/**
 * Selection of optimal algorithm
 * @author  Maja Knezev
 *
 * */
public class algo {

        /**
         *   algotype can be:
         *  "one"(one-end),"two"(two-end),"three"(three-end),"ga"(genetic algorithm)
         */
        public String algotype="";
        onedata v_one=new onedata();
        twodata v_two=new twodata();
        threedata v_three=new threedata();
        gadata v_ga=new gadata();
        /**one if selected*/
        int isone=0, istwo=0, isthree=0;
        int busP=0, busQ=0, busS=0;

        /**pathfile contains all the files and paths*/
        algo(String pathfile) {

                /** defining files */
                setpath files = new setpath(pathfile);

                /** defining v_pti */
                ptifilepro v_pti = new ptifilepro(pathfile);

                /** defining v_DI */
                DfrInfo v_DI = new DfrInfo();
                v_DI.SetValue(pathfile);

                /** defind v_dtf */
                DfrToFl v_dtf = new DfrToFl(pathfile,(float)0.0,(float)0.0);
                v_dtf.SetValue();
                v_dtf.toperunit();

                /** Evaluating whether one-, two- or three- end recordings are available*/
                select1(v_pti, v_dtf, v_DI);
                select2(v_pti, v_dtf, v_DI);
                select3(v_pti, v_dtf, v_DI);

                System.out.println("\n"+isone+" "+istwo+" "+isthree);

                getlocation(files, v_pti, v_dtf, v_DI, isone, istwo, isthree, algotype,pathfile);
}
```

```
/**judge whether the single-end or parallel line algo is selected*/
void select1(ptifilepro v_pti, DfrToFl v_dtf, DfrInfo v_DI){

        int k, m, n, isdou;
        boolean  a1, a2;
        int[] index=new int[2];
        isone = 0;
         /**judge whether the faulty branch is one of the parallel lines*/
        for ( k=0; k<v_DI.Getnofbinternal(); k++) {

                v_one.ckt1=v_DI.Getfptickin().get(k);
                v_one.ftype=v_DI.Getft(1) ;
                isdou = 0;
                for (m=0; m<v_dtf.encurrent; m++){

                        1 = (v_dtf.eiptino1.get(m).equals(v_DI.Getfptino1in().get(k)));
                        a2 = (v_dtf.eiptino2.get(m).equals(v_DI.Getfptino2in().get(k)));
                        if ( a1 && a2 ) {

                            index[isdou] = m; //to make index range from 0 to encurrent-1
                            /**if the two nodes of faulty branch match those of the branches whose
                            currents  are monitored, increase isdou by one*/
                            isdou ++;
                        }
                }
                if (isdou > 1) {

                        for (m=0; m<v_dtf.envoltage; m++){
                        //current flows from 1.2
                        a1 = (v_dtf.evptino.get(m).equals( v_DI.Getfptino1in().get(k)));
                        if (a1) {
                                isone = 1;
                                v_one.pti_p = v_DI.Getfptino1in().get(k); //endow value to onedata
                                v_one.pti_q = v_DI.Getfptino1in().get(k);
                                for (n=0; n<3; n++)
                                   v_one.vp.set(n,new Complex(v_dtf.GetevDurPhasor(m+1).get(n)));

                                // index[0] indexes the faulty line
                                if(v_dtf.eiptick.get(index[0]).equals(v_DI.Getfptickin().get(k))){

                                v_one.ckt2=v_dtf.eiptick.get(index[1]);
                                for (n=0; n<3; n++) //faulty branch currents are put first
                                v_one.ip.set(n,
                                              new Complex(v_dtf.GeteiDurPhasor(index[0]+1).get(n)));

                                 for (n=0; n<3; n++)
                                 v_one.ip.set(n+3,
                                               new Complex( v_dtf.GeteiDurPhasor(index[1]+1).get(n)));
                                }else{

                                v_one.ckt2=v_dtf.eiptick.get(index[0]);
                                for (n=0; n<3; n++)
                                v_one.ip.set(n, v_dtf.GeteiDurPhasor(index[1]+1).get(n));
                                for (n=0; n<3; n++)
                                v_one.ip.set(n+3,v_dtf.GeteiDurPhasor(index[0]+1).get(n));
                                }
```

```
                                        break;
                                        }
                                }
                        }
        if ( (isone == 1) && (v_DI.Getnoft() == 1) ) break;
        }

        if (isone == 0) {

        //judge whether the faulty branch is one of the parallel lines
        for ( k=0; k<v_DI.Getnofbinternal(); k++) {

                v_one.ckt1=v_DI.Getfptickin().get(k);
                v_one.ftype=v_DI.Getft(1) ;
                isdou = 0;
                for (m=0; m<v_dtf.encurrent; m++){

                        a1 = (v_dtf.eiptino1.get(m).equals( v_DI.Getfptino2in().get(k)));
                        a2 = (v_dtf.eiptino2.get(m).equals( v_DI.Getfptino1in().get(k)));
                        if ( a1 && a2 ) {

                             index[isdou] = m; //to make index range from 0 to ncurrent-1
                            /**if the two nodes of faulty branch match those of the branches whose
                            currents are monitored, increase isdou by one*/
                             isdou ++;
                        }
                }
                if (isdou > 1) {

                        for (m=0; m<v_dtf.envoltage; m++){
                            //current flows from 2.1
                             a1 = (v_dtf.evptino.get(m).equals( v_DI.Getfptino2in().get(k)));
                             if (a1) {
                                  isone = 1;
                                  v_one.pti_p = v_DI.Getfptino2in().get(k); //endow value to onedata
                                  v_one.pti_q = v_DI.Getfptino1in().get(k);
                                  for (n=0; n<3; n++)
                                   v_one.vp.set(n,new Complex(v_dtf.GetevDurPhasor(m+1).get(n)));

                                   if (v_dtf.eiptick.get(index[0]).equals(v_DI.Getfptickin().get(k))) {

                                       v_one.ckt2=v_dtf.eiptick.get(index[1]);
                                       for (n=0; n<3; n++)
                                        v_one.ip.set(n,
                                                new Complex(v_dtf.GeteiDurPhasor(index[0]+1).get(n)));
                                       for (n=0; n<3; n++)
                                        v_one.ip.set(n+3,
                                               new Complex( v_dtf.GeteiDurPhasor(index[1]+1).get(n)));
                                  }else{
                                      v_one.ckt2=v_dtf.eiptick.get(index[0]);
                                       for (n=0; n<3; n++)
                                      v_one.ip.set(n,
                                              new Complex( v_dtf.GeteiDurPhasor(index[1]+1).get(n)));
                                       for (n=0; n<3; n++)
                                     v_one.ip.set(n+3
                                              new Complex(v_dtf.GeteiDurPhasor(index[0]+1).get(n)));
                                }
                                break;
                                }
                            }
                        }
                if ( (isone == 1) && (v_DI.Getnoft() == 1) )break;
                }
        }
        if ( (isone == 1) && (v_DI.Getnoft() == 1) ){

                v_pti.busno2impe(v_one.pti_p, v_one.pti_q, v_one.ckt1);
                v_one.impe.add(new Complex(v_pti.rz, v_pti.xz));
                v_one.impe.add(new Complex(v_pti.r, v_pti.x));
                v_one.impe.add(new Complex(0, 0)); //assumed as zero
                v_pti.busno2impe(v_one.pti_p, v_one.pti_q, v_one.ckt2);
                v_one.impe.add(new Complex(v_pti.rz, v_pti.xz));
                v_one.impe.add(new Complex(v_pti.r, v_pti.x));
                v_one.line = v_pti.busno2len( v_one.pti_p, v_one.pti_q);
        }else{
                isone =0;
        }
}
/**     judge whether the two-end algo is selected*/
void select2(ptifilepro v_pti, DfrToFl v_dtf, DfrInfo v_DI){

        int k, m, n, count;
        boolean  a1=false, a2=false, a3;
        int[] index= new int[2];
        istwo = 0;
        for ( k=0; k<v_DI.Getnofbinternal(); k++){

                count = 0;
                /**find terminal p*/
                for (m=0; m<v_dtf.encurrent; m++) {

                        a1 = (v_dtf.eiptino1.get(m).equals( v_DI.Getfptino1in().get(k)));
                        a2 = (v_dtf.eiptino2.get(m).equals(v_DI.Getfptino2in().get(k)));
                        a3 = v_dtf.eiptick.get(m).equals(v_DI.Getfptickin().get(k));

                        if ( a1 && a2 && a3){
                                count ++;
                                index[0] = m;
                                break;
                        }
                }
                /**find terminal q*/
                for (m=0; m<v_dtf.encurrent; m++) {

                        a1 = (v_dtf.eiptino1.get(m).equals( v_DI.Getfptino2in().get(k)));
                        a2 = (v_dtf.eiptino2.get(m).equals( v_DI.Getfptino1in().get(k)));
                        a3 = v_dtf.eiptick.get(m).equals( v_DI.Getfptickin().get(k));

                        if ( a1 && a2 && a3){
                                count ++;
                                index[1] = m;
                                break;
```

```
                    }
            }

            /**check whether p or q's voltages are monitored*/
            for (m=0; m<v_dtf.envoltage; m++) {

                    a1 = ( v_dtf.evptino.get(m).equals( v_DI.Getfptino1in().get(k)));
                    if (a1==true) break;
            }
            /**check whether p or q's voltages are monitored*/
            for (m=0; m<v_dtf.envoltage; m++) {

                    a2 = ( v_dtf.evptino.get(m).equals( v_DI.Getfptino2in().get(k)));
                    if (a2==true) break;
            }

            if ( (count > 1) && a1 && a2 ) {

                    istwo = 1;
                    v_two.pti_p = v_DI.Getfptino1in().get(k);
                    v_two.pti_q = v_DI.Getfptino2in().get(k);
                    v_two.ckt=v_DI.Getfptickin().get(k);
                    v_two.ftype=v_DI.Getft(1);

                    v_two.vp_pre.clear();
                    v_two.vp_pos.clear();
                    v_two.ip_pre.clear();
                    v_two.ip_pos.clear();
                    v_two.vq_pre.clear();
                    v_two.vq_pos.clear();
                    v_two.iq_pre.clear();
                    v_two.iq_pos.clear();

                    for (n=0; n<3; n++){

                            v_two.vp_pre.add(v_dtf.GetvPrePhasorByPtino(v_two.pti_p).get(n));
                            v_two.vp_pos.add( v_dtf.GetvDurPhasorByPtino(v_two.pti_p).get(n));
                            v_two.ip_pre.add( v_dtf.GeteiPrePhasor(index[0]+1).get(n));
                            v_two.ip_pos.add( v_dtf.GeteiDurPhasor(index[0]+1).get(n));
                            v_two.vq_pre.add( v_dtf.GetvPrePhasorByPtino(v_two.pti_q).get(n));
                            v_two.vq_pos.add( v_dtf.GetvDurPhasorByPtino(v_two.pti_q).get(n));
                            v_two.iq_pre.add( v_dtf.GeteiPrePhasor(index[1]+1).get(n));
                            v_two.iq_pos.add( v_dtf.GeteiDurPhasor(index[1]+1).get(n));
                    }
                    v_pti.busno2impe(v_two.pti_p, v_two.pti_q, v_two.ckt);
                    v_two.impe.clear();
                    v_two.impe.add(new Complex(v_pti.rz,v_pti.xz));
                    v_two.impe.add(new Complex(v_pti.r,v_pti.x));
                    v_two.impe.add(new Complex(0,0));
                    v_two.impe.add(new Complex(v_pti.yz,0));
                    v_two.impe.add(new Complex(v_pti.y,0));
                    v_two.line = v_pti.busno2len( v_two.pti_p, v_two.pti_q);
                    break;
            }
        }
}
```

```
/**     judge whether the three end algo is selected*/
void select3(ptifilepro v_pti, DfrToFl v_dtf, DfrInfo v_DI){

        int k;
        isthree = 0;

        for ( k=0; k<v_DI.Getnofbinternal(); k++){

                v_three.ftype=v_DI.Getft(1) ;

                isthree = findTeedline(v_pti, v_dtf, v_DI, v_DI.Getfptino1in().get(k));
                if (isthree == 1) {
                                break;
                }

                isthree = findTeedline(v_pti, v_dtf, v_DI, v_DI.Getfptino2in().get(k));
                if (isthree == 1) {
                                break;
                }
        }
}
/**
*               Judge whether branch PT, QT and ST's currents are monitored, as well as
*                judge whether terminal P, Q and S's voltages are monitored.
*/
int findTeedline(ptifilepro v_pti, DfrToFl v_dtf, DfrInfo v_DI, int busT){

        int m, n, isTeed, found = 0;
        int[] i_index=new int[3],v_index = new int[3];
        boolean a1, a2, a3, a4, a5, a6;
        busP=busQ=busS=0;

        isTeed = isTeedline(v_pti, busT); //is busT the middle no. of a Teedline config. ?
        if (isTeed == 0) {return 0;}

        //judge whether the currents are monitored
        for (m=0; m<v_dtf.encurrent; m++) {
                a1 = ((int)v_dtf.eiptino1.get(m)== busP);
                a2 = ((int)v_dtf.eiptino2.get(m)== busT);
                a3 = ((int)v_dtf.eiptino1.get(m) == busQ);
                a4 = ((int)v_dtf.eiptino2.get(m) == busT);
                a5 = ((int)v_dtf.eiptino1.get(m) == busS);
                a6 = ((int)v_dtf.eiptino2.get(m) == busT);
                if (a1 && a2) i_index[0] = m + 1;
                if (a3 && a4) i_index[1] = m + 1;
                if (a5 && a6) i_index[2] = m + 1;
        }
        //judge whether the voltages are monitored
        for (m=0; m<v_dtf.envoltage; m++) {
                a1 = ((int)v_dtf.evptino.get(m) == busP);
                a2 = ((int)v_dtf.evptino.get(m) == busQ);
                a3 = ((int)v_dtf.evptino.get(m) == busS);
                if (a1) v_index[0] = m + 1;
                if (a2) v_index[1] = m + 1;
                if (a3) v_index[2] = m + 1;
        }
```

```
if ( (i_index[0]!=0) && (i_index[1]!=0) && (i_index[2]!=0) && (v_index[0]!=0) &&
(v_index[1]!=0) && (v_index[2]!=0)) {

        found = 1;
        v_three.pti_p = busP;
        v_three.pti_q = busQ;
        v_three.pti_s = busS;
        v_three.pti_t = busT;
        v_three.ckt1= v_dtf.eiptick.get(i_index[0] - 1);
        v_three.ckt2= v_dtf.eiptick.get(i_index[1] - 1);
        v_three.ckt3= v_dtf.eiptick.get(i_index[2] - 1);

        v_three.vp_pre.clear();
        v_three.vq_pre.clear();
        v_three.vs_pre.clear();
        v_three.vp_pos.clear();
        v_three.vq_pos.clear();
        v_three.vs_pos.clear();

        v_three.ip_pre.clear();
        v_three.iq_pre.clear();
        v_three.is_pre.clear();
        v_three.ip_pos.clear();
        v_three.iq_pos.clear();
        v_three.is_pos.clear();

        for (n=0; n<3; n++){
                v_three.vp_pre.add(v_dtf.GetevPrePhasor(v_index[0]).get(n));
                v_three.vq_pre.add(v_dtf.GetevPrePhasor(v_index[1]).get(n));
                v_three.vs_pre.add(v_dtf.GetevPrePhasor(v_index[2]).get(n));

                v_three.vp_pos.add(v_dtf.GetevDurPhasor(v_index[0]).get(n));
                v_three.vq_pos.add(v_dtf.GetevDurPhasor(v_index[1]).get(n));
                v_three.vs_pos.add(v_dtf.GetevDurPhasor(v_index[2]).get(n));

                v_three.ip_pre.add(v_dtf.GeteiPrePhasor(i_index[0]).get(n));
                v_three.iq_pre.add(v_dtf.GeteiPrePhasor(i_index[1]).get(n));
                v_three.is_pre.add(v_dtf.GeteiPrePhasor(i_index[2]).get(n));
                v_three.ip_pos.add(v_dtf.GeteiDurPhasor(i_index[0]).get(n));
                v_three.iq_pos.add(v_dtf.GeteiDurPhasor(i_index[1]).get(n));
                v_three.is_pos.add(v_dtf.GeteiDurPhasor(i_index[2]).get(n));
        }
        v_three.impe.clear();
        v_pti.busno2impe(busP, busT, v_three.ckt1);
        v_three.impe.add(new Complex(v_pti.rz,v_pti.xz));
        v_three.impe.add(new Complex(v_pti.r, v_pti.x));
        v_three.impe.add(new Complex(0, 0));
        v_three.impe.add(new Complex(v_pti.yz, 0));
        v_three.impe.add(new Complex(v_pti.y, 0));

        v_pti.busno2impe(busQ, busT, v_three.ckt2);
        v_three.impe.add(new Complex(v_pti.rz, v_pti.xz));
        v_three.impe.add(new Complex(v_pti.r, v_pti.x));
        v_three.impe.add(new Complex(0, 0));
        v_three.impe.add(new Complex(v_pti.yz, 0));
        v_three.impe.add(new Complex(v_pti.y, 0));
```

```
                v_pti.busno2impe(busS, busT, v_three.ckt3);
                v_three.impe.add(new Complex(v_pti.rz, v_pti.xz));
                v_three.impe.add(new Complex(v_pti.r, v_pti.x));
                v_three.impe.add(new Complex(0, 0));
                v_three.impe.add(new Complex(v_pti.yz, 0));
                v_three.impe.add(new Complex(v_pti.y, 0));

                v_three.line.clear();
                v_three.line.add((double)v_pti.busno2len(busP, busT));
                v_three.line.add((double)v_pti.busno2len(busQ, busT));
                v_three.line.add((double)v_pti.busno2len(busS, busT));
                }
        return found;
}

/**
<pre>
                 A Teedline is defined as having the following configuration and
                 P, Q and S are different nodes)

                                P-------T---------Q
                                                  |
                                                  |S
</pre>
*/
int isTeedline(ptifilepro v_pti, int busT){

        //struct connect data_con;
        int isORno = 0;
        boolean a1, a2, a3;

        v_pti.getconnect(busT, -1);
        int len_p = arraylenP(v_pti.bus); //return Positive number (line number)
        int len_t = v_pti.bus.size(); //return total line number

        a1 = ! v_pti.busisdoub( v_pti.bus.get(1), busT);
        a2 = ! v_pti.busisdoub( v_pti.bus.get(2), busT);
        a3 = ! v_pti.busisdoub( v_pti.bus.get(3), busT);

        if ( (len_p == len_t) && (len_p == 4) && a1 && a2 && a3 ){

                isORno = 1;
                busP = v_pti.bus.get(1);
                busQ = v_pti.bus.get(1);
                busS = v_pti.bus.get(1);
        }else{
                isORno = 0;
        }

        return isORno;
}
/**     obtain the length of the array counting only the positive numbers
        array: 1 2 34 0: return 3;
        array: 1 2 -34 0: return 2;
*/
```

```
int arraylenP(List list) {

        int count = 0;
        for(int i=0;i<list.size();i++)
        if((Integer)list.get(i)>0)count++;

        return count;
}
/**
        Wwrite the index of the voltages and currents to file: MatchIndex.
         Indexed to quantities written in file:FtVoCu.
        MatchIndex will be read by fitness function in GA for fitness evaluation
*/
void matchingindex(setpath files, ptifilepro v_pti,DfrToFl v_dtf,  DfrInfo v_DI){

        String s="";
        CopyFile c=new CopyFile();

        int mnvoltage, mncurrent, k, k1, mnv, mnc;
        //mnv, mvc are the true number of matching voltages and currents
        //mnvoltage, mncurrent are the number of matching voltages and currents,
        //specified in .tot file

        mnvoltage = v_DI.Getmatch_bus().busnumber;
        mncurrent = v_DI.Getmatch_nocurrent();

        //only the voltage of the starting bus of the faulty lines are used
        mnv = 0;
        if (mnvoltage == -2) {

                for (k=0; k<v_DI.Getnofbinternal(); k++)
                        for (k1=0; k1<v_dtf.envoltage; k1++){
                                if (v_DI.Getfptino1in().get(k).equals(v_dtf.evptino.get(k1))) {
                                        mnv ++;
                                        break;
                                }
                        }

                        for (k=0; k<v_DI.Getnofbexternal(); k++)
                                for (k1=0; k1<v_dtf.envoltage; k1++){
                                if
                                (v_DI.Getmslexternal().get(k).bus.get(0).equals(v_dtf.evptino.get(k1)){
                                        mnv ++;
                                        break;
                                }
                        }
                 if (mnv > 0) {
                        s+=mnv+"\n";
                        for (k=0; k<v_DI.Getnofbinternal(); k++)
                                for (k1=0; k1<v_dtf.envoltage; k1++){
                                        if (v_DI.Getfptino1in().get(k).equals(v_dtf.evptino.get(k1))) {
                                                s+=k1+"\n";
                                                break;
                                        }
                                }
```

```
            for (k=0; k<v_DI.Getnofbexternal(); k++)
                        for (k1=0; k1<v_dtf.envoltage; k1++){
                                    if(v_DI.Getmslexternal().get(k).bus.get(0).equals(v_dtf.evptin
                                    o.get(k1)))
                                    {
                                                s+=k1+"\n";
                                                break;
                                    }
                        }
            }
            if (mnv ==0 ) mnvoltage = 0;
}

//all the monitored quantities are used
if ( mnvoltage == 0){
            s+=v_dtf.envoltage+"\n";
            for (k = 0; k<v_dtf.envoltage; k++) s+=k+"\n";
}

//none of the voltages are used
if (mnvoltage == -1) { s+=0+"\n"; }

if (mnvoltage > 0) {
            s+=mnvoltage+"\n";
            for (k=0; k<mnvoltage; k++)
                        for (k1=0; k1<v_dtf.envoltage; k1++){

                            if (v_DI.Getmatch_bus().bus.get(k).equals( v_dtf.evptino.get(k1))){

                                    s+=k1+"\n";
                                    break;
                            }
                        }
}
//only the currents of the faulty lines are used
mnc = 0;
if (mncurrent == -2){
            for (k=0; k<v_DI.Getnofbinternal(); k++)
                        for (k1=0; k1<v_dtf.encurrent; k1++){
                                    if (
                                    (v_DI.Getfptino1in().get(k).equals(v_dtf.eiptino1.get(k1)))&&

                                    (v_DI.Getfptino2in().get(k).equals(v_dtf.eiptino2.get(k1)))&&

                                    (v_DI.Getfptickin().get(k).equals(v_dtf.eiptick.get(k1)))){
                                                mnc++;
                                                break;
                                    }
                        }
            for (k=0; k<v_DI.Getnofbexternal(); k++)
                        for (k1=0; k1<v_dtf.encurrent; k1++)
                        {
            if ( (v_DI.Getmslexternal().get(k).bus.get(0).equals(v_dtf.eiptino1.get(k1))) &&
                 (v_DI.Getmslexternal().get(k).bus.get(1).equals(v_dtf.eiptino2.get(k1))) &&
                  (v_DI.Getmslexternal().get(k).ckt.equals(v_dtf.eiptick.get(k1))))
                                    {
```

```
                                                    mnc ++;
                                                    break;
                                            }
                                    }

            if (mnc > 0) {
                            s+=mnc+"\n";

                            for (k=0; k<v_DI.Getnofbinternal(); k++)
                                    for (k1=0; k1<v_dtf.encurrent; k1++){
                                        if ( (v_DI.Getfptino1in().get(k).equals(v_dtf.eiptino1.get(k1))) &&
                                            (v_DI.Getfptino2in().get(k).equals(v_dtf.eiptino2.get(k1)))&&
                                            (v_DI.Getfptickin().get(k).equals(v_dtf.eiptick.get(k1))) ) {
                                                    s+=k1+"\n";
                                                    break;
                                             }
                                    }

                            for (k=0; k<v_DI.Getnofbexternal(); k++)
                                    for (k1=0; k1<v_dtf.encurrent; k1++){
                                        if
                              (v_DI.Getmslexternal().get(k).bus.get(0).equals(v_dtf.eiptino1.get(k1))) &&
                              (v_DI.Getmslexternal().get(k).bus.get(1).equals(v_dtf.eiptino2.get(k1))) &&
                              (v_DI.Getmslexternal().get(k).ckt.equals(v_dtf.eiptick.get(k1)))) {
                                    s+=k1+"\n";
                                    break;
                                        }
                            }
                    }

                    if (mnc ==0 ) mncurrent = 0;
            }

            //all the monitored quantities are used
            if ( mncurrent == 0) {
                    s+=v_dtf.encurrent+"\n";
                    for (k = 0; k<v_dtf.encurrent; k++)s+=k+"\n";
            }
            //none of the currents are used
            if (mncurrent == -1) {s+=0+"\n";}
            if ( mncurrent>0 ) {

                s+=mncurrent+"\n";
                 for (k=0; k<v_DI.Getmatch_psse().sectionnumber; k++){
                    for (k1=0; k1<v_dtf.encurrent; k1++){
                            if ( ( v_DI.Getmatch_psse().bus.get(2*k).equals( v_dtf.eiptino1.get(k1))) &&
                            ( v_DI.Getmatch_psse().bus.get(2*k+1).equals( v_dtf.eiptino2.get(k1))) &&
                            ( v_DI.Getmatch_psse().ckt.equals(v_dtf.eiptick .get(k1)) )) {
                                    s+=k1+"\n";
                             }
                    }
            }
          for (k=0; k<v_DI.Getmatch_nocurrent_line(); k++){
                for (k1=0; k1<v_dtf.encurrent; k1++){
                    if ( ( v_DI.Getmatch_line().get(k).bus.get(0).equals(v_dtf.eiptino1.get(k1))) &&
                    v_DI.Getmatch_line().get(k).bus.get(1).equals(v_dtf.eiptino2.get(k1))) &&
                    (v_DI.Getmatch_line().get(k).ckt.equals(v_dtf.eiptick.get(k1))) ) {
                            s+=k1+"\n";
                    }
                }
            }
        }
            s+="0 \n";
            try{c.setContents(new File(files.MatchIndex),s);
            }catch(IOException e){System.out.println(e.getMessage());}
    }
    //select the algorithms and calculate the  the fault location
    void getlocation(setpath files, ptifilepro v_pti, DfrToFl v_dtf,  DfrInfo v_DI, int isone, int istwo, int isthree,
    String algotype, String pathfile) {

            update v_ud = new update(files, v_pti, v_dtf, v_DI);

            if (istwo==1) algotype="two";
            else{
                    if (isthree==1) algotype="three";
                    else{
                            if (isone==1) {
                                    algotype="one";
                            }else{
                                    algotype="ga";
                            }
                    }
            }
            if ( algotype=="one") {
                                    v_one.n = 1;
                                    v_one.method="pl";
                                    v_one.locator();
                                    v_one.getvirf(files);
            }
            if ( algotype== "two"){
                    v_two.impetype= "noneed";
                    v_two.method1= "presyn"; //setting different code for different algs
                    v_two.method2= "cal";
                                    v_two.locator();
            }
            if ( algotype=="three") {
                                    v_three.method="presyn"; //setting different code for different algs
                                    v_three.impetype="noneed";
                                    v_three.locator();
            }
            if ( algotype=="ga") {
                                    v_ud.topoupdate();
                                    v_ud.creatftvocupre();
                                    v_ud.synchronize();
                                    matchingindex( files, v_pti, v_dtf, v_DI);
                                    v_ga.locator(pathfile);
                                    v_ga.setvalue(pathfile);
                                    v_ga.getvirf(files);
            }
    }
    }
```

APPENDIX B

Rule.java

```
package CTC_PsercT32;
import java.io.*;
import java.util.Scanner;
import java.util.*;
 /**
  * Extracting Rule(root of binary tree) information which represents begining node
  * of binary tree.
  * <pre>
  * Rule attributes can be explained through next:
  *    movie_name is name of animation that should be shown in operation view if
  *    CB if this rule was fired and then input operands are compared:
  *       (input 1) (operation) (input 2) Example: (5)(>)(3)
  *       if comparison is satisfied next rule that should be analyzed is left
  *       if comparison is NOT satisfied next rule that should be analyzed is right
  *
  *    input 3 is used as additional parameter in some cases
  * </pre>
  * @author  Maja Knezev
  */
public class Rule implements Serializable{

     /** input1*/
     public String input1="T1";
     /** input2*/
     public String input2="T2";
     /** input3*/
     public String input3="";
     /** operation*/
     public String operation=">";
     /** operation*/
     public String movie_name="";
     /** left and right nodes*/
     public Rule left=null,right=null;

    /** Creates a new instance of Rule */
    public Rule(String input1, String input2,String operation, String movie_name) {
       this.input1=input1;
       this.input2=input2;
       this.operation=operation;
       this.movie_name=movie_name;
    }
    /**
     * Creates a new instance of Rule.
     * @param input1 is input operand used for deciding whether condition is satisfied
     * @param input2 is input operand used for deciding whether condition is satisfied
     * @param input3 is input operand used for deciding whether condition is satisfied
     * @param operation is operation used for deciding whether condition is satisfied
     * @param movie_name is name of animation that should be shown in operation view of
     * CB if this rule was fired and then input operands are compared.
     */
    public Rule(String input1, String input2,String input3,String operation, String movie_name) {
       this.input1=input1;
       this.input2=input2;
       this.input3=input3;
       this.operation=operation;
       this.movie_name=movie_name;
    }
    /**
     * <pre>
     * Creating a new instance of Rule in form of binary tree out of RuleReport in XML format.
     * If flag is true RuleReport is already in BinaryFormat, else if flag is false RuleReport
     * is in form of linked list. Elements in linked list are assumed to be in next order:
     *
     *                 0
     *              /     \
     *            1         2
     *          /  \       | \
     *        3     4       5    6
     *      / \  / \   / \  |\
     *      7   8  9   10  11  12 13 14 ...etc.
     * </pre>
     * @param filename is name of source file from which serilized Rule (root of binary tree) should
     * be retrived
     * @param flag is used to mark if RuleReport is serialized into binary tree or linked list format
     */
    public Rule(String filename,boolean flag) {

       /** Reads Rule from XML file*/
       RulesXML t = new RulesXML();

       if(flag)
       t.ReadFromXML_BinaryTree(filename);
       else
       t.ReadFromXML_List(filename);

       /**Setting read Rule information to current Rule object*/
       setRuleValue(t.Rules);
    }
    /**
     * Writing (serializing) this object into XML file. This object is assumed
     * to be root of binary tree, so complete tree is serialized.
     * In VID Spreadsheet application absolute path of file used for serializing
     * list is C:\\VID\\Fault\\RuleReport.xml
     *
     * @param filename is absolute path of file into which tree should
     * be serialized
     */
    public void WriteRuleToXMLReport(String filename) {

       /** Reads Rule from XML file*/
       RulesXML t = new RulesXML();
       t.WriteToXML_BinaryTree(filename,this);
    }
    /**
     *  Creating a new instance of Rule(root of binary tree) out of list of
     *  elements.
     *  List elements are assumed to be in next order:
     *  <pre>
     *                 0
     *                 /     \
     *                1       2
     *              /   \   /   \
     *            3    4   5     6...etc.
     *  </pre>
     * @param list is List of Rule objects that should be interpreted as list
     * of nodes of binary tree.
     */
    public Rule(List<Rule> list) {

       for(int i=0;i<list.size();i++){
           try{

         list.get(i).setLRelem(list.get(2*i+1),list.get(2*i+2));
         }catch(IndexOutOfBoundsException e){System.out.println(e.getMessage()); break;}
       }
       this.input1=list.get(0).input1;
       this.input2=list.get(0).input2;
       this.input3=list.get(0).input3;
       this.operation=list.get(0).operation;
       this.movie_name=list.get(0).movie_name;
       this.left=list.get(0).left;
       this.right=list.get(0).right;

       //check print out
      /*Rule temp=this;
       while(temp!=(null)){
             System.out.print("->"+temp.movie_name);
             temp=temp.left;
       }*/
    }
    /** Empty constructor.*/
    public Rule() {}
    /**
     * Sets values of this.Rule according to input argument of type Rule parameters.
     * @param f is object of type Rule used to copy attribute values of this class.
     */
    public void setRuleValue(Rule f){
         this.input1=f.input1;
         this.input2=f.input2;
         this.input3=f.input3;
         this.operation=f.operation;
         this.movie_name=f.movie_name;
         this.left=f.left;
         this.right=f.right;
    }
    /**
     * Assigning next left and right rules that should be executed in case that
     * operation of current rule is satisfied and not respectevly.
     */
    public void setLRelem(Rule l,Rule r){

         this.left=l;
         this.right=r;
    }
```

```
}
```

RulesXML.java

```
package CTC_PsercT32;
import com.thoughtworks.xstream.XStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.ArrayList;
import java.util.Iterator;
/**
 *<pre>
 * This class imports and extract Rule information into and from .XML file
 * From importad rules binary tree is constructed.
 *_________________________________________________________________________
 *  There are three classes dealing with binary tree at this moment:
 *  - RuleXML
 *  - Rule
 *  - RuleOutput.
 *
 *  First run RuleXML class. It will create RuleReport in XML format.
 *  RuleXML class should be run only once.
 *  Than run RuleOutput class. It will read Rule report from RuleReport.xml
 *  extract corresponding binary tree and check corresponding nodes with timings
 *  generated in AllTimings object. As output list of animation will be produced
 *  and executed by 3D CB operational view.
 *  You need next libraries to run this application:
 *  xpp3-1.1.2.3.O.zip and xstream-1.1.3.jar
 *_________________________________________________________________________
 *</pre>
 * @author  Maja Knezev
 *
 **/
public class RulesXML {
    ArrayList<Rule> RulesList = new ArrayList<Rule>();
    Rule Rules=new Rule();

   /** Creates a new instance of RuleXML */
   public RulesXML() {}
   /**
    * Generating rules, connecting them and serializing them into binary tree or
    * list of rules (user decides).
    */
   public static void main(String[] args) {

       System.out.print("Rules are output");
       RulesXML t = new RulesXML();
        Rule Rules1=new Rule("T3","-2.0",">","");

        Rule Rules2=new Rule("T2","T3","<","");

        Rule Rules3=new Rule("T2","T2Ref","T2tol","<>","");
Rule Rules4=new Rule("","","","NotPossible");

Rule Rules5=new Rule("T3","T6","<","A4");

Rule Rules6=new Rule("T3","T6","<","A2");

Rule Rules7=new Rule("T3","T3Ref","T3tol","<>","");

Rule Rules8=new Rule("T3","T3Ref","T3tol","<>","");

Rule Rules9=new Rule("T6","T6Ref","T6tol","<>","A9");

Rule Rules10=new Rule("T6","T6Ref","T6tol","<>","A7");

Rule Rules11=new Rule("T4","T4Ref","T4tol","<>","");

Rule Rules12=new Rule("T6","T6Ref","T6tol","<>","A5");

Rule Rules13=new Rule("T6","T6Ref","T6tol","<>","A7");

Rule Rules14=new Rule("T6","T6Ref","T6tol","<>","A3");

Rule Rules15=new Rule("T2","T2Ref","T2tol","<>","");

Rule Rules16=new Rule("","","<","A8");

Rule Rules17=new Rule("T6","T6Ref","T6tol","<>","A9");

Rule Rules18=new Rule("T4","T4Ref","T4tol","<>","");

Rule Rules19=new Rule("","","<","A2");

Rule Rules20=new Rule("T6","T6Ref","T6tol","<>","A3");

Rule Rules21=new Rule("","","","A10");

Rule Rules22=new Rule("","","","");


//START serilizing direct into binary tree
Rules1.left=Rules2;
Rules1.right=Rules15;
Rules2.left=Rules3;
Rules2.right=Rules4;
Rules3.left=Rules5;
Rules3.right=Rules6;
Rules5.left=Rules7;
Rules5.right=Rules4;
Rules6.left=Rules8;
Rules6.right=Rules4;
Rules7.left=Rules9;
Rules7.right=Rules10;
Rules8.left=Rules12;
Rules8.right=Rules11;
Rules9.left=Rules21;
```

```
 Rules9.right=Rules22;
 Rules10.left=Rules21;
 Rules10.right=Rules22;
 Rules11.left=Rules13;
 Rules11.right=Rules14;
 Rules12.left=Rules21;
 Rules12.right=Rules22;
 Rules13.left=Rules21;
 Rules13.right=Rules22;
 Rules14.left=Rules21;
 Rules14.right=Rules22;
 Rules15.left=Rules16;
 Rules15.right=Rules18;
 Rules16.left=Rules17;
 Rules16.right=Rules17;
 Rules17.left=Rules21;
 Rules17.right=Rules22;
 Rules18.left=Rules16;
 Rules18.right=Rules19;
 Rules19.left=Rules20;
 Rules19.right=Rules20;
 Rules20.left=Rules21;
 Rules20.right=Rules22;

 t.Rules=Rules1;
 t.WriteToXML_BinaryTree("C:\\VID\\Fault\\RuleReport.xml",t.Rules);
 t.ReadFromXML_BinaryTree("C:\\VID\\Fault\\RuleReport.xml");
 //*/
 //END serilizing direct into binary tree


/*
 //serilizing direct into binary tree example
 Rules1.left=Rules2;
 Rules1.right=Rules3;
 Rules2.left=Rules4;
 Rules2.right=Rules5;
 t.Rules=Rules1;
 t.WriteToXML_BinaryTree("C:\\VID\\Fault\\RuleReport.xml",t.Rules);
 t.ReadFromXML_BinaryTree("C:\\VID\\Fault\\RuleReport.xml");
 //END serilizing direct into binary tree
 */


 //serilizing into list
/*ArrayList<Rule> p = new ArrayList<Rule>();
 p.add(Rules1);
 p.add(Rules2);
 p.add(Rules3);
 p.add(Rules4);
 p.add(Rules5);
 p.add(Rules6);
 p.add(Rules7);
 p.add(Rules8);
 p.add(Rules9);
 p.add(Rules10);
 p.add(Rules11);
 p.add(Rules12);
 p.add(Rules13);
 p.add(Rules14);
 p.add(Rules15);
 p.add(Rules16);
 p.add(Rules17);
 p.add(Rules18);

 t.RulesList.addAll(p);

 t.WriteToXML_List("C:\\VID\\Fault\\RuleReport.xml",t.RulesList);
 t.ReadFromXML_List("C:\\VID\\Fault\\RuleReport.xml");
 */
 //t.Rules.WriteRuleToXMLReport("C:\\VID\\Fault\\RuleReport1.xml");
 //END serilizing into list
 }

/**
 * Writing (serializing) Rules object into XML file. This object is assumed
 * to be root of binary tree, so complete tree is serialized.
 * In VID Spreadsheet application absolute path of file used for serializing
 * list is C:\\VID\\Fault\\RuleReport.xml
 *
 * @param filename is absolute path of file into which tree should
 * be serialized
 */
public void WriteToXML_BinaryTree(String filename, Rule f){
     XStream xstream = new XStream();
     xstream.alias("ListOfRules",RulesXML.class);

     OutputStream out = null;
     try {
     out = new FileOutputStream(filename);
     } catch (FileNotFoundException ex) { ex.printStackTrace();}

    Rules=f;
    xstream.toXML(this,out);
}
/**
 * Writing (serializing) RulesList object into XML file. This object is list of rules,
 * which represent nodes of binary tree. List elements are assumed to be in next order:
 *  <pre>
 *                   0
 *                 /    \
 *                1      2
 *              /  \  /  \
 *             3    4   5     6...etc.
 *  </pre>
 * In VID Spreadsheet application absolute path of file used for serializing
 * list is C:\\VID\\Fault\\RuleReport.xml
 *
 * @param filename is absolute path of file into which tree should
 * be serialized
 */
public void WriteToXML_List(String filename,ArrayList<Rule> f){
        XStream xstream = new XStream();
        xstream.alias("ListOfRules",RulesXML.class);

        OutputStream out = null;
        try {
        out = new FileOutputStream(filename);
        } catch (FileNotFoundException ex) { ex.printStackTrace();}

       RulesList=f;
       xstream.toXML(this,out);
    }
    /**
     * Reading (deserializing) binary tree from XML file.
     * In VID Spreadsheet application absolute path of file used for
     * deserializing list is C:\\VID\\Fault\\RuleReport.xml
     *
     * @param filename is absolute path of file from which list should
     * be deserialized
     */
     public void ReadFromXML_BinaryTree(String filename){

        XStream xstream = new XStream();
        xstream.alias("ListOfRules",RulesXML.class);

        InputStream in = null;
        try {
            in = new FileInputStream(filename);
        } catch (FileNotFoundException ex) { ex.printStackTrace(); }
        Rules = ((RulesXML) xstream.fromXML(in)).Rules;
        }
    /**
     * Reading (deserializing) list of rules from XML file.
     * In VID Spreadsheet application absolute path of file used for
     * deserializing list is C:\\VID\\Fault\\RuleReport.xml
     *
     * @param filename is absolute path of file from which list should
     * be deserialized
     */
     public void ReadFromXML_List(String filename){

        XStream xstream = new XStream();
        xstream.alias("ListOfRules",RulesXML.class);

        InputStream in = null;
        try {
            in = new FileInputStream(filename);
        } catch (FileNotFoundException ex) { ex.printStackTrace(); }
        RulesList=((RulesXML) xstream.fromXML(in)).RulesList;
        //converting complete list into binary list object
        Rules=new Rule(RulesList);
    }
}
```

RuleOutput.java

```
package CTC_PsercT32;
import java.util.ArrayList;
import java.util.List;

/**
 * Processing binary tree in order to retrieve list of animations that should be shown in 3D operational view.
 * Implemented engine starts from root of binary tree and evaluates according to marked operation and
input
 * arguments whether left or right node of rule should be further checked. This is repeated until last node
 * (leaf). For each node that was accessed during processing, specific animation should be executed on 3D
 * operational view of CB.
 * Input arguments used as operands are retrievd from CB report and CB settings file. CB report containes
 * times of CB event that should be evaluated. CB Settings time containes common operational times and
 * tolerances for specific type of breaker involved in event.
 *
 * @author Maja Knezev
 */
public class RuleOutput {
    static public String CB_Report="C:\\VID\\Fault\\CBReport.xml";
    static public String CB_Settings="C:\\VID\\Fault\\CBSettings.xml";
    static public String Rule_Report="C:\\VID\\Fault\\RuleReport.xml";
    static public String CB_Signals="C:\\VID\\Fault\\CBSignals.dat";
    public AllTimes CB=new AllTimes();
    public List <String> s=new ArrayList();
    public Signals CB_signals=new Signals();

    /**
     *<pre>
     * Retrieving and processing binary tree. Implemented engine starts from root of binary tree and
evaluates
     * according to marked operation and input arguments whether left or right node of rule should be further
     * checked. This is repeated until last node (leaf). For each node that was accessed during processing,
     * specific animation should be executed on 3D operational view of CB and name of animation is added
to
     * list  of strings. Processing of one rule (node of binary tree):
     *_____________________________________________________________
     *   movie_name is name of animation that should be shown in operation view if
     *   CB if this rule was fired and then input operands are compared:
     *      (input 1) (operation) (input 2) Example: (5)(>)(3)
     *      if comparison is satisfied next rule that should be analyzed is left
     *      if comparison is NOT satisfied next rule that should be analyzed is right
     *
     *   input 3 is used as additional parameter in some cases
     *_____________________________________________________________
     *
     * Input arguments used as operands are retrievd from CB report and CB settings file. CB report
containes
     * times of CB event that should be evaluated. CB Settings time containes common operational times and
     * tolerances for specific type of breaker involved in event.
     * In VID Spreadsheet application corresponding reports used during processing are assumed to be as:
     *     CB_Report="C:\\VID\\Fault\\CBReport.xml";
     *     CB_Settings="C:\\VID\\Fault\\CBSettings.xml";
     *     Rule_Report="C:\\VID\\Fault\\RuleReport.xml";
     *     CB_Signals="C:\\VID\\Fault\\CBSignals.dat";
     * </pre>
     * All above CB reports contain list values that belong to different CB events. First element of list is used
    * for retrieving CB input arguments, when using this (default) constructor.
    */
   public RuleOutput() {

      //retriving CB timings
      AllTimesXML t=new AllTimesXML();
      t.ReadFromXML(CB_Report);
      CB=t.CBs.get(0);

      //retriving settings of corrensponding CB
      CB.cb_set=new AllSettings(CB_Settings,0);

      //retriving all CB signals
      SignalsSerillization sig_object=new SignalsSerillization();
      sig_object.ReadObject(CB_Signals);
      CB_signals=sig_object.CBs.get(0);

      CB.print(null);
      CB.cb_set.print(null);

      s.clear();
      Rule tree=new Rule(Rule_Report,true);
      boolean done=false;
      try{
      while((tree!=null)&&(!done))
      {
       //s+="\n"+tree.movie_name;
       if (!tree.movie_name.trim().equals("")) s.add(tree.movie_name);


        System.out.println("\r\n"+tree.input1+tree.operation+tree.input2+"+-"+tree.input3+"--
>"+tree.movie_name);

        if(tree.operation.equals("<")){
          if( CB.getValue(tree.input1)<CB.getValue(tree.input2))
          tree=tree.left;
          else tree=tree.right;
        }else if(tree.operation.equals(">")){
          if( CB.getValue(tree.input1)>CB.getValue(tree.input2))
             tree=tree.left;
          else tree=tree.right;
        }else if(tree.operation.equals("<>")){
           /**Here we are checking the range.
            * Assumption is that
            * input1 is real measured value,
            * input2 is refernce value and
            * input3 three is reference tolerance
            */
           float temp=CB.getValue(tree.input1);
           float temp1=CB.getValue(tree.input2)-CB.getValue(tree.input3);
           float temp2=CB.getValue(tree.input2)+CB.getValue(tree.input3);

           if((temp<temp1)||( temp>temp2))
              tree=tree.left;
           else tree=tree.right;
      }else {done=true;}
     }
     }catch(NullPointerException e){}


     System.out.println(s);


  }
  /**
   *<pre>
   * Retrieving and processing binary tree. Implemented engine starts from root of binary tree and
evaluates
   * according to marked operation and input arguments whether left or right node of rule should be further
   * checked. This is repeated until last node (leaf). For each node that was accessed during processing,
   * specific animation should be executed on 3D operational view of CB and name of animation is added
to
   * list  of strings. Processing of one rule (node of binary tree):
   *________________________________________________________
   *   movie_name is name of animation that should be shown in operation view if
   *   CB if this rule was fired and then input operands are compared:
   *      (input 1) (operation) (input 2) Example: (5)(>)(3)
   *      if comparison is satisfied next rule that should be analyzed is left
   *      if comparison is NOT satisfied next rule that should be analyzed is right
   *
   *   input 3 is used as additional parameter in some cases
   *________________________________________________________
   *
   * Input arguments used as operands are retrievd from CB report and CB settings file. CB report
containes
   * times of CB event that should be evaluated. CB Settings time containes common operational times and
   * tolerances for specific type of breaker involved in event.
   * In VID Spreadsheet application corresponding reports used during processing are assumed to be as:
   *     CB_Report="C:\\VID\\Fault\\CBReport.xml";
   *     CB_Settings="C:\\VID\\Fault\\CBSettings.xml";
   *     Rule_Report="C:\\VID\\Fault\\RuleReport.xml";
   *     CB_Signals="C:\\VID\\Fault\\CBSignals.dat";
   *
   * </pre>
   * All above CB reports contain list of values that belong to different CB events. Element of list whose
name
   * matches with a provided name as input argument of this constructor is retrived and used for
comparisons of
   * timings and settings for specific CB and event.
   * @param CB_name is name of CB whose event values should be retrived. Event values are times of the
latest
   * operation, default settings for specific type of CB and recorded signals of processed event.
   */
  public RuleOutput(String CB_name) {


     AllTimesXML t=new AllTimesXML();
     t.ReadFromXML(CB_Report);

     SignalsSerillization sig_object=new SignalsSerillization();
     sig_object.ReadObject(CB_Signals);
```

```
        for (int h=0;h<t.CBs.size();h++){

          if ((CB_name.toLowerCase().equals(t.CBs.get(h).Name.toLowerCase()))||(t.CBs.size()==1) ){
              //retriving CB timings
              CB=t.CBs.get(h);
              //retriving settings of corrensponding CB
              CB.cb_set=new AllSettings(CB_Settings,h);
              //retriving all CB signals
              CB_signals=sig_object.CBs.get(h);
              break;
          }
        }
        if(CB.Name.equals("")) return;

        CB.print(null);
        CB.cb_set.print(null);

        s.clear();
        Rule tree=new Rule(Rule_Report,true);
        boolean done=false;
        try{
        while((tree!=null)&&(!done))
        {
         if (!tree.movie_name.trim().equals("")) s.add(tree.movie_name);

         System.out.println("\r\n"+tree.input1+tree.operation+tree.input2+"+-"+tree.input3+"--
>"+tree.movie_name);

         if(tree.operation.equals("<")){
           if( CB.getValue(tree.input1)<CB.getValue(tree.input2))
           tree=tree.left;
           else tree=tree.right;
         }else if(tree.operation.equals(">")){
           if( CB.getValue(tree.input1)>CB.getValue(tree.input2))
              tree=tree.left;
           else tree=tree.right;
         }else if(tree.operation.equals("<>")){
            /**
             * Here we are checking the range.
             * Assumption is that
             * input1 is real measured value,
             * input2 is refernce value and
             * input3 three is reference tolerance
             */
            float temp=CB.getValue(tree.input1);
            float temp1=CB.getValue(tree.input2)-CB.getValue(tree.input3);
            float temp2=CB.getValue(tree.input2)+CB.getValue(tree.input3);

            if((temp<temp1)||( temp>temp2))
              tree=tree.left;
            else tree=tree.right;

         }else {done=true;}
        }
        }catch(NullPointerException e){}


        System.out.println(s);


    }
    /** Testing engine for processing binary tree*/
    public static void main(String[] args) {

      //RuleOutput process=new RuleOutput();
      RuleOutput process1=new RuleOutput("CB2");

    }

}
```

VITA

Name:	Maja Knezev
Address:	Department of Electrical Engineering, College Station, TX, 77843-3128
Email Address:	maja_knezev@yahoo.com
Education:	Dipl. Ing., Electrical Engineering, The University of Novi Sad, Serbia, 2004 M.S., Electrical Engineering, Texas A&M University, US, 2007