Integrated Design of Electrical Distribution Systems: Phase Balancing and Phase Prediction Case Studies

by

Murat Dilek

Dissertation submitted to the Faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Electrical Engineering

2001
Blacksburg, Virginia USA

Keywords: Phase Balancing, Phase Prediction, Integrated Design, Power Distribution Design, Software Framework, Software Architecture

Integrated Design of Electrical Distribution Systems: Phase Balancing and Phase Prediction Case Studies

by

Murat Dilek

Dr. Robert P. Broadwater, Chair

Bradley Department of Electrical Engineering

(ABSTRACT)

Distribution system analysis and design has experienced a gradual development over the past three decades. The once loosely assembled and largely ad hoc procedures have been progressing toward being well-organized. The increasing power of computers now allows for managing the large volumes of data and other obstacles inherent to distribution system studies. A variety of sophisticated optimization methods, which were impossible to conduct in the past, have been developed and successfully applied to distribution systems.

Among the many procedures that deal with making decisions about the state and better operation of a distribution system, two decision support procedures will be addressed in this study: phase balancing and phase prediction. The former recommends re-phasing of single- and double-phase laterals in a radial distribution system in order to improve circuit loss while also maintaining/improving imbalances at various *balance point* locations. Phase balancing calculations are based on circuit loss information and current magnitudes that are calculated from a power flow solution. The phase balancing algorithm is designed to handle time-varying loads when evaluating phase moves that will result in improved circuit losses over all load points.

Applied to radial distribution systems, the phase prediction algorithm attempts to predict the phases of single- and/or double phase laterals that have no phasing information previously recorded by the electric utility. In such an attempt, it uses available customer data and kW/kVar measurements taken at various locations in the system. It is shown that phase balancing is a special case of phase prediction.

Building on the phase balancing and phase prediction design studies, this work introduces the concept of integrated design, an approach for coordinating the effects of various design calculations. Integrated design considers using results of multiple design applications rather than employing a single application for a distribution system in need of improvement relative to some system aspect. Also presented is a software architecture supporting integrated design.

Acknowledgments

Special thanks to Dr. Robert Broadwater for giving generously of his valuable time, intellect and friendship.

Thanks are also due to Dr. Jaime De La Ree, Dr. Yilu Liu, Dr. Hanif Sherali and Dr. Hugh VanLandingham for serving on my advisory committee.

Contents

Figures

Tables

Chapter 1 Introduction

1.1 Prelude

The primary mission of a transmission and distribution system [1.1], [1.2] is to deliver power to electrical customers at their place of consumption and in ready-to-use form. While performing this first and main task, a utility is interested in achieving sufficient levels of the following attributes (objectives):

- reliability
- performance
- efficiency
- peak load
- protection
- costs

When a utility reaches every customer with an electrical path of sufficient strength to satisfy the customer's demand for electric power, that electrical path must be *reliable* so that it provides an uninterrupted flow of stable power to the utility's customer. *Reliable* power means delivering all of the power demanded and doing so all of the time.

Beyond the need to deliver power to the customer, the utility must also deliver it without having under-voltages, over-voltages or over-currents that are not tolerable by any piece of equipment in the system including customer equipment. Further, the voltage at the customer premise is required not to be below the nominal utilization voltage by more than a specified amount (e.g. 5%) because most electrical equipment is designed to operate properly when supplied with acceptable voltages and currents. *Performance* is a measure of how well and how stable voltages and currents fall into allowed ranges.

Having a high power factor is almost as favorable as having in-range voltages and currents with regard to attaining good performance measures. Movement of power through any electrical device, be it a conductor, transformer, regulator, etc., produces a certain amount of electrical loss due to impedance (resistance to the flow of electricity) of the device. These losses which are a result of the inevitable laws of nature create a cost to the utility. Electrical losses can be measured, assessed, and minimized through proper engineering, but never eliminated completely. The lower the loss, the more *efficiently* the system is said to be operating.

A power delivery system exists because consumers demand electric power. The load on a power system, or on any unit of equipment within that power system, represents the accumulated electrical load of all the customers being served by that system or unit of equipment. The load will vary from hour to hour, from day to day, and from season to season. What interests a power system (utility) most is the *peak value* encountered within a particular time span (typically a year). This peak is important because it is the maximum amount of power that must be delivered, and thus defines the capacity requirements for equipment. In a growing power system, not only does the system-wide peak grow from year to year, but also peak loadings on many units of equipment, such as substations and feeders, increase annually as load increases. The system must have sufficient capacity to meet the peak demands of its customers. The units of equipment within the system must be able to handle the excessive conditions seen at the *peak load* interval. System designs or operations that lead to lower peak loads are sought by utilities.

When electrical equipment fails, such as a line being knocked down during a storm, the normal function of the electrical equipment is interrupted. *Protective* equipment is designed to detect such conditions, protect the equipment from excessive currents, and isolate damaged equipment whenever a fault, failure or other unwanted condition occurs on the system. Thus, a good measure in *protection* refers to the ability of protecting not only devices but also people from unwanted conditions.

A transmission and distribution system can be expensive to design, build, and operate. Equipment at every level involves two types of *costs*: *capital costs* and *operating costs*. Capital costs include the equipment and land, labor for site preparation, construction, assembly and installation, and any other costs associated with building and putting the equipment into operation. Operating costs include labor and equipment for operation, maintenance and service, taxes and fees, as well as the value of the power lost due to electrical losses. Usually, capital cost is a one-time cost (once it's built, the money has been spent). Operating costs are continuous or periodic. Obviously, the utility aims to maintain these costs at the minimum level possible [1.3].

1.2 Challenges in Distribution System Analysis and Design

The utility needs to perform analysis and design studies to assure itself that the system provides satisfactory levels of the attributes above. Basically, analysis studies involve evaluating the system (or any piece of system equipment) behavior whereas design studies consider evaluating decisions that can be made about the system behavior.

However, distribution systems exhibit various characteristics that impede modeling and analysis studies associated with the operation, control and planning of distribution systems. These impediments are summarized as follows [1.4]:

- *System:* The distribution system is usually composed of a large variety of components that are of different phases. It can have for example many single- or double-phase laterals branching off the main three-phase feeder.

- *Load Distribution:* Loads are scattered along feeders and laterals. Flows on the distribution system usually are unbalanced.

- *Data:* The challenge in distribution system analysis stems mainly from the fact that distribution systems are designed to operate without much monitoring and control. Therefore, there is very limited actual data about the system for modeling and analysis.

A good design study defines the goals and identifies a performance index. That is, it determines “What are we trying to do and how will we measure our success?”. Then it identifies alternatives, “What are our options?”.

Finally, it selects the best alternative(s) based on the performance index [1.5].

Yet, for a number of reasons pertinent to electric power distribution systems, performing distribution system design studies is usually very challenging compared to many other design studies. In evaluating alternatives, one should measure the value of each alternative using the performance index defined. For this, it is usually required to analyze first the impact of the alternative on the system. Hence, impediments involved in analysis studies penetrate to design studies. The following lists a set of additional reasons that complicate design (or decision making) studies [1.5].

- *Combinatorial aspects of distribution systems*. Distribution equipment is interconnected in such a way that every unit impacts the operation, performance and economics of other components of the system. Thus, the evaluation of a potential solution can become a surpassingly involved process.

- *Size.* Distribution systems involve a great number of components. This means that there may exist too many possible alternatives that may be identified and analyzed. The evaluation of the set of all feasible alternatives may turn out to be a very laborious, and sometimes impossible, process.

- *Uncertainty.* Uncertainty in predicting future events is a major concern in design studies involving long-range planning [1.3]. Even the best forecasts and projections of financial and other planning priorities leave some uncertainty about the future conditions influencing the distribution plan. For example, a new factory might develop as rumored, causing a large, yet unforecasted, increment of load growth. As a result, plans should be developed to address any, at least the most likely, eventualities (multi-scenario approach) [1.3].

- *Involvement with other aspects of society*. Distribution designers must interact with many aspects of the society, including other utilities, regulators, intervenes, community groups and business leaders who often have important, but not quantifiable, concerns about power quality, cost, aesthetics, and equipment siting [1.5].

1.3 Some Design Procedures in Electric Distribution

Distribution system analysis and design has experienced a gradual development over the past three decades. The once loosely assembled and largely ad hoc procedures have been progressing toward being well-organized and quantitative elements of analysis and design studies. The increasing power of computers now allows for managing the large volumes of data and other obstacles inherent to distribution systems. A variety of sophisticated optimization methods, which were impossible to conduct in the past, have been developed and successfully applied to distribution systems.

Among the many procedures that deal with making decisions about the state and better operation of a distribution system, four decision support procedures will be mentioned in the following discussion. The first two procedures will be under consideration in great detail in the proceeding chapters. The last two are presented to provide an understanding on the variety of available design-oriented studies.

1.3.1 Phase Balancing Problem

It is necessary to distribute load evenly across the three phases in order for the system to have better system performance. As part of this work, decisions must be made about how single- and two-phase laterals are connected. For example, a single-phase lateral could be connected to either phase A, B, or C of the main feeder. Making decisions as to how the laterals should be connected is referred to as lateral phase balancing design. Here, lateral phase balancing will also be referred to as just "phase balancing (PB)."

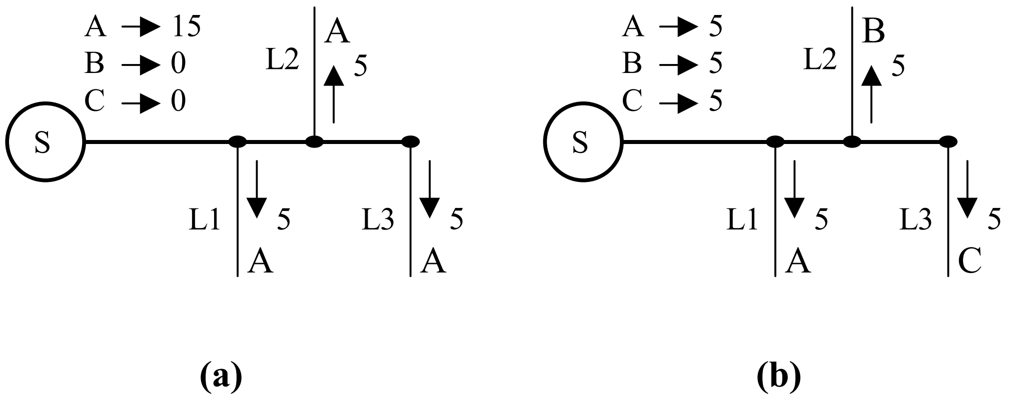


Figure 1.1 Performing phase balancing on a simple unbalanced circuit. (a) kW flows before PB. (b) kW flows after PB. Circuit losses are ignored for simplicity. Thicker lines correspond to the three-phase segments of circuit

Given a distribution system, PB determines phasing modifications that will produce a more efficient system (i.e., system with less losses). The circuit given in Figure 1.1(a), for example, is operating inefficiently due to highly unbalanced phases. This poor balance is due to the fact that all single-phase laterals have been connected to phase A. PB recommendations are as follows:

- No change on L1.
- Connect L2 to phase B of the three-phase feeder.
- Connect L3 to phase C of the three-phase feeder.

The circuit as modified according to these recommendations (see Figure 1.1 (b)) is now more efficient than the original case. In addition to efficiency, PB affects the performance, peak load, and the cost of the power delivery system.

1.3.2 Phase Prediction Problem

It is not uncommon to find electric utilities that did not record the phasing information of the laterals in the system. This is a major deficiency in information because analysis studies, such as the calculation of power flows and voltages on power system components, need to know how and to what phases laterals are connected. Without this information, engineers may arbitrarily assign phases to laterals which have unknown phases. Studies then are carried out based on these assumed phasings.

However, this problem can be tackled. Utilities usually record voltage, current, power factor or real/reactive power measurements taken at some important three-phase locations. Furthermore, information associated with loads connected to the laterals is likely to be available/recorded. Thus one can use these load and measurement data to determine the unknown phases. In such an attempt, phases of laterals are changed in an iterative manner until the resulting flows at the measurement locations come closest to the measured flows. This phase identification process is called phase prediction.

Consider the simple circuit given in Figure 1.2(a), where phases of the laterals are not known. However, some kW measurements are taken at the substation and lateral loads are also known. Note that there are 27 phasing combinations possible such as AAA, AAB, AAC, ABA, ABB, and so on, where letters in a combination represent the phases for L1, L2 and L3 respectively. In this simple case, the answer is ABC (Figure 1.2(b)) because the flows that will result with this combination exactly match the measurements at the substation.

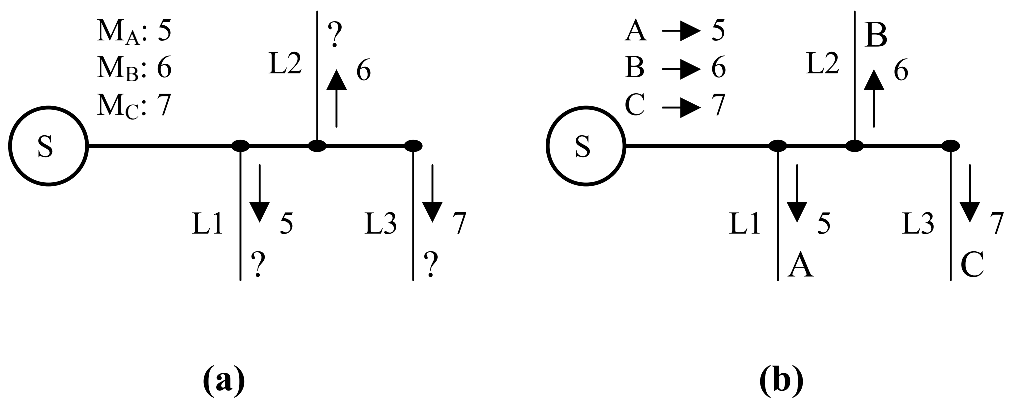


Figure 1.2 Performing phase prediction on a simple circuit with three laterals that have unknown phases. (a) kW measurements at the start of the circuit and calculated (estimated) kW flows on the laterals. (b) Predicted phases of laterals, and kW flows that will be obtained at the start of the circuit based on these phase assignments.

1.3.3 Capacitor Placement Problem

Capacitors are extensively used in distribution systems as one solution to reactive power compensation, power loss reduction and voltage regulation problems. The general capacitor design problem consists of determining the locations, sizes, and types of capacitors. In a typical capacitor placement problem, limiting voltage and power factor values are specified in order to be used for selecting the placement locations. For example, locations where voltage is less then 95% of the nominal voltage and power factor is worse than 85% can be considered as candidate locations for capacitor placement. In addition, an upper power factor limit can be provided so that one keeps adding capacitors to a selected location until the power factor of the selected location reaches this upper limit.

As with phase balancing, the performance, efficiency, peak load and cost of the power delivery system are affected by capacitor design.

1.3.4 Reconfiguration Problem

In an electrical distribution system that operates radially, every segment of the system gets its power from only one substation, with a unique path from it to the associated substation. This path involves feeders, switches, transformers, etc. Almost always the distribution system has more than one substation and a substation in the system has more than one feeder. This multi-substation, multi-feeder design means that for a segment in the system there might be more than one option regarding from what substation or feeder to obtain power.

Switching operations are used to transfer load from one feeder to another while keeping the system radial. In so doing, system losses are reduced and load balancing among the feeders can be achieved. Reconfiguration is the process of altering the topological structures of distribution feeders by changing the open/closed status of the sectionalizing and tie switches.

During emergency situations such as fault conditions and equipment failures, transferring load and or reconfiguring the system is then performed to isolate the problem area while retaining service to as many customers as possible [1.6], [1.7]. This type of action is known as restoration, but is based upon reconfiguration.

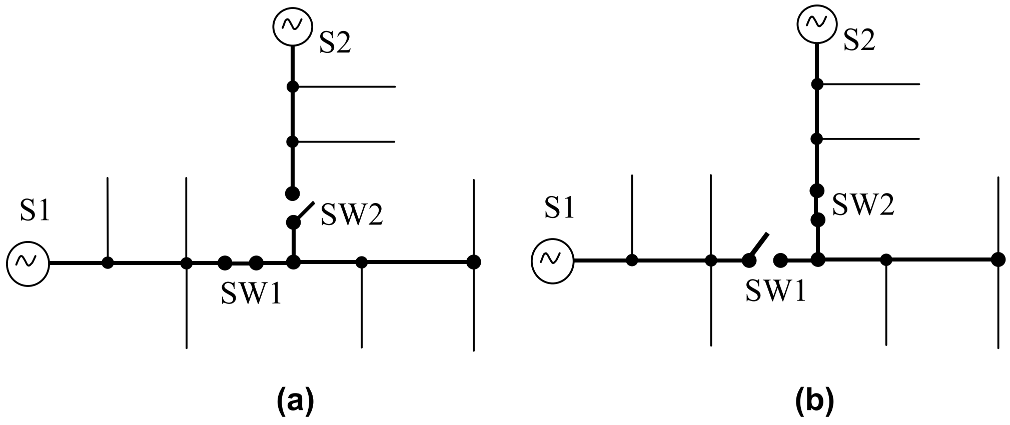


Figure 1.3 Performing reconfiguration on a simple two-substation distribution system. (a) Original system. (b) Reconfigured system.

To illustrate the use of reconfiguration in normal conditions, consider the system given in Figure 1.3(a). There are two substations, S1 and S1, and two switches SW1 (normally closed) and SW2 (normally open). A reconfiguration study on this system suggests that losses can be minimized by changing the system topology as shown in Figure 1.3 (b). In this recommended configuration SW1 is opened and SW2 is closed.

The reconfiguration tool will affect the reliability, efficiency, peak load and costs of the power delivery system.

1.4 Software Aspects

Design procedures are implemented as computer programs. Not only design procedures but also various analysis studies can be implemented as computer programs. These programs are often incorporated in a single software system. The Electric Power Research Institute (EPRI) sponsored Distribution Engineering Workstation (DEWorkstation) [1.8] is such a software package which is designed to meet the present and future needs of distribution engineering. It provides the distribution engineer with an integrated data and application environment.

Over the past few decades software has became an important part of an increasing number of technical systems including power systems. Like most software systems, software systems designed for power systems are complex entities. The reason is that the problem domain (i.e., analysis and design of an electric power distribution system) is complex. It is difficult to characterize the behavior of a continuous complex system and to manage the software development process.

There are certain attributes that “good” software is expected to meet. It is not easy to express what these attributes are. Although perspectives to identify, define and value these attributes may differ from person to person, the following lists a number of software qualities. These qualities have been edited from the discussions given by various authors [1.9], [1.10], [1.11], [1.12], [1.13]. The list below reflects a mix of attributes from both user and developer points of view.

- **Reliability**: The software needs as much as possible to be free of errors, bugs, defects or system failures.

- **Functionality**: The software product satisfies given needs. That is, it assists it users in their tasks.
- **Security**: The software prevents unauthorized access or misuse.
- **Portability**: Transferring software to other hardware or operating systems should be easy.
- **Performance**: The system should perform adequately in term of running-response time, number of events processed per second, and so on.
- **Scalability**: The software should be able to be scaled up to handle greater usage demands (e.g., greater volume of data, greater number of users, greater number of requests)
- **Usability**: Learning and using of the software product should be easy, and this should apply to all categories of users.
- **Reusability**: It should be possible that the whole software system or any implementation/design unit in the software can be reused in other software products.
- **Readability**: The effort required to understand a piece of software should not be too much.
- **Maintainability**: It must be possible to make changes to the software. There is another term called flexibility which is used to reflect a special dimension of maintainability. Flexibility refers to the ability to upgrade or enhance the software as the requirements change.

- **Affordability**: The cost/time to buy, maintain or develop software should be reasonable.

It is hard to assess the overall quality of software. There is no software development technique that will insure that the software product achieves satisfactory degrees of all software quality attributes listed above. However, object-oriented technology [1.10], [1.14], [1.15], [1.16], [1.17] has become very promising over the last decade for solving software quality problems. Although, it is not a guaranteed solution, it is the best among the available techniques. It is a paradigm that has changed the way software is built.

The following sections discuss briefly the basics of object-orientation, and compare it with traditional approaches. The last section gives an introduction to another technology called component technology, which has recently emerged as a derivation from object-orientation.

1.4.1 Object-Orientation at a Glance

Object-oriented technology encourages thinking in terms of objects. Naturally, the most fundamental concept of the object-oriented paradigm is an object. Object-oriented software is built around objects that collaborate with one another to solve a problem. Conceptually, an object is a thing that you can interact with. It can be the representation of a physical thing such as a book or a resistor as well as a conceptual thing such as a compiler or a matrix. One can send an object various *messages* that it will react to. How it *behaves* depends upon the current internal *state* of the object. The current *state* of an object may change as part of the object's reaction to receiving a *message*. An object has an *identity* which distinguishes it from all other objects [1.9].

Hence an object is a thing that has a *behavior*, *state*, and *identity* [1.10]. As a software entity, an object is a collection of data and a set of operations that manipulate the data. The basic properties of an object are considered below.

- *State*. The collection of information, represented by data, which is currently held by the object is the object's state. Data is represented as *attributes*, each of which has a value. A *file* object in a word processor system for example may have attributes such as name, size, date of creation, owner etc. An object's state may change over time. Only the operations performed on the object can change its state.

- *Behavior*. An object understands certain messages and does not support others. It can receive certain messages and act on them. The way in which an object reacts to a message may depend on the current values of its *attributes*. The *file* object, for instance, can support operations such as copy, delete and rename.

- *Identity*. Although this term is not clear, it can be said that an object may be uniquely identified. That is, the values of the object's attributes could change, perhaps in response to a message, but it would still be the same object.

Every object belongs to a specific *class*. A *class* describes a collection of related objects. Objects of the same *class* support the same collection of operations and have common set of possible *states* [1.18].

Object-orientation exhibits some characteristics that distinguish it from the traditional structured (procedural) approach. Although one can claim a number of reasons for its superiority, the following section discusses the fundamental aspects inherent to object-orientation which make it favorable as compared to the procedural approach.

1.4.2 Object-Oriented vs. Procedural

When we analyze a problem domain, we generate models. A model is an abstract representation of a specification, a design or a system from a particular point of view [1.9]. The model represents an aspect of reality and is always much simpler than the reality. Models help us to manage the complexity and understand the problem domain reality [1.19].

In the traditional analysis approach, the problem domain is modeled using functions or behaviors as building blocks. With object-oriented analysis, objects are building blocks for modeling the reality in the problem domain. After object types and the services of these object types are described, the behavior of the problem domain is modeled as a sequence of messages that are passed between various objects [1.19].

For instance, if we view a simple word processor system from a procedural point, we will think of all functions a word processor performs such as editing, printing, spell-checking, etc. However, an object-oriented approach will view the word processor as collection of objects such as document, screen, command, file, word, etc.

The fact that object orientation looks at the world in terms of objects provides one very important benefit. A good software system should meet what the users need. Yet, user needs change over time. It is one of the hardest things for an existing software system to adjust itself in order to follow the emerging requirements of the user. When the system's model of the problem domain and the processes to be performed are similar to the user's model, changes that should be made in the system's model in response to the changes in the user's model will exhibit similar characteristics to those in user's model. Notice that looking at the world in terms of objects is the way a human-being (the user) looks at the world. Hence, in object orientation, the system's model and the user's model are compatible.

In the procedural approach, the way the system is modeled is very different from the way the user views the problem domain. A small change in the user's model (that is in user requirements) may necessitate dramatic changes in the system's model (that is in the software).

There is another factor that makes object-orientation superior to the procedural approach: Domain objects change less frequently and less dramatically than the exact functionality that the user requires. So if a system is structured on the basis of these objects, changes to the system are likely to require less major disruption than if the system were based on the more volatile functional aspects [1.9].

In summary, compared to the function-oriented approach, the object-oriented paradigm provides the software with the ability to [1.9]:

- Capture requirements more easily and accurately
- Follow changes in user requirements better
- Allow for a more naturally interacting system.

1.4.3 Component Technology

Research on object-oriented technology and its extensive use has led to the development of component-oriented programming. Rather than being an alternative to object-orientation, component technology extends the initial object concepts. [1.20]. Despite the fact that there is little agreement on the definition of 'component' and 'component-based software engineering (CBSE)' [1.21], the best comprehensive definition for a component can be given as 'a thing we will be able to reuse or replace' [1.9]. Osterlund in [1.22] defines a component as 'a set of executable modules having a well-defined API and a set of requirements on the environment in which it is to be installed'.

The basic idea behind component software is to enable independent development of software modules that can plug together without a need to recompile. In the idealistic view, component software technology will allow customers to buy components from different suppliers, install them, and obtain a working complex system that provides the joint functionality enabled by all the different components [1.22].

Component-based development (CBD) is the process of building software systems out of prepackaged generic elements (i.e., components). The increasing use of components is stimulated by object-orientation and will drastically change the way systems are built and maintained [1.23]. CBSE will be the mainstream software practice [1.21]. As we continue to move toward object orientation and CBSE, software development will become less creative and more like traditional manufacturing. The emphasis will shift from coding to architectural thinking (designing and integrating) [1.23].

A main focus of this dissertation is to point out the importance of current trends in software engineering and suggest that power system software, which is in general large and complex, should be developed employing the techniques offered by these technologies.

1.5 Problems to be Addressed

One aspect of the problems to be addressed is concerned with design calculations in electric distribution systems. These design calculations need to work together in an integrated software environment. In developing this integrated software environment component-based software architecture will be applied.

It is very important for a distribution system to have balanced currents in the three-phase portion of the system. Usually the substation is the major element of the system that should be kept as balanced as possible. By this we mean the currents at the substation should be balanced. While making phase moves to improve the imbalance of the substation, some internal locations might have imbalances worsened due to these phase move operations. This has significance in situations where certain segments of the system are transferred from one feeder to another. If such a transfer occurs at a location that was previously worsened, then this transferred section will damage the imbalance of the substation that it is being transferred to (Of course we assume the substation has good balance before the transfer is made).

Another aspect of phase balancing is that a utility usually can not afford too many phase moves. If phase balancing comes up with 100 phase move suggestions for example, the time and the labor to implement these suggested operations are not manageable by the utility. Most of the time a utility is interested only in a couple of best moves that will lead to improved system operation.

This work will consider developing a phase balancing algorithm with an emphasis given to the practical aspects above. It enables the user to specify locations where the imbalances do not deteriorate during the course of phase balancing. Also, it allows the user to indicate a limit on the number of phase move operations that will be performed by phase balancing.

Distribution systems are large and complex, consisting of a wide variety of components. In order to analyze a distribution system one must gather the information about the type, number, size and location of the elements that constitute the system. This is very hard work as pointed out in Section 1.2. Even when information is available, keeping track of this information may be very difficult. For example, records about a transformer might have been kept in separate databases, and these records might contradict each other because they were not synchronized consistently. Another difficulty may arise from the fact that parts are replaced over time and data is not updated accordingly.

One class of data that is very significant to the utility is that regarding customers. The location, number, type, and energy usage of customers should be recorded and maintained. The significance of the customer data comes from the fact that many essential analysis studies such as power flow and load estimation depend on customer information. Customer information is not complete without knowing the phases that customers are connected to. Hence, phasing information is a prerequisite to analysis studies.

As pointed out earlier, utilities might have no customer phasing information previously recorded. Fortunately, they usually keep other data such as type, energy usage and location of customers as well as some measurements taken at various locations in the system. This work will attempt to predict unknown phases on the system via using the customer data and the measurements. Real and reactive power flow measurements will be used.

A distribution engineering software system is simply a collection of analysis and design applications, each developed to handle a certain task. It is a large system, being the integration of many complex application modules. The complete software is too complex for a single person or even a team of people to manage. Thus, different software teams develop different analysis/design applications. The development teams may work for different companies. People perhaps do not even know one another. Furthermore, the applications they develop are applied to the same distribution system. It is vitally important that the system model capture the essential properties of the system components (that is the abstraction of the reality should be accurate and complete), so that every application has the information that it needs.

A vital aspect of integrated software is that the modules be inserted or deleted without disrupting the remaining parts. At least the degree of disruption should be minimal. In the software industry it has been a dream to achieve modular systems in a way that software modules can be plugged together and work together like electronic chips on a circuit board. No solution so far has solved this modularization problem.

Therefore, the problem of analyzing and designing distribution systems leads to another arena when it comes to implementing the engineering studies as software entities in an integrated software system. While developing phase balancing and phase prediction algorithms, this work will use software principles towards achieving modularity.

In order to achieve modularization, high-level decisions regarding the overall structure of the system should be made prior to developing any individual modules. The architecture of the system, where the modules are to be developed, plugged and then integrated involves such decisions. Our work intends to propose a software architecture that will help understand and manage the complexity.

Studies of design type are employed by engineers in the hope to improve certain system aspects. As pointed out earlier, a capacitor placement design study helps in improving, for example, performance and efficiency. Although the primary concerns in applying different design studies may differ, two or more different design studies might be effecting similar system aspects, without knowing one another. One system aspect might be impacted by the application of these different design studies. Hence, one can ask the question " how about applying these design tools together? If I use them together can I get a better result than the one I would be getting if I applied only one design study?" This work aims to investigate the ways to combine the results coming from different design tools. In our work, considering multiple design tools when trying to improve a given system in some system aspect will be referred to as integrated design.

In the traditional approach, the engineer uses a single application at a time. Only an integrated software environment, in which various application modules can communicate with one another and work together, can provide the engineer with the ability to effectively perform multiple things. He can access the results of applications. He can change, manipulate or store these results. Then, to see the consequences, he can make applications run in whatever order he wishes. Many examples regarding such things that one can achieve with integrated software can be given. One benefit of having integrated software, however, is clear at this point: it makes the engineer powerful and productive.

1.6 Literature Review

A literature search has been conducted mainly in the areas of:

- Distribution system phase balancing
- Phase prediction in distribution systems
- Integrated design of existing design tools
- Software architecture for distribution system analysis and design.

Regarding the lateral phase balancing problem in radial distribution systems, very little work has been reported in the literature. Dolloff [1.6], for example, has proposed one phase balancing algorithm that evaluates phase balance improvements at the substation. He employs a heuristic technique in which the substation is the only balance point, and phase recommendations are evaluated for obtaining the minimum real power or reactive power imbalance at the substation. In his study, he disconnects all phases that could be moved and rebuilds the phasing for the entire circuit from scratch. This rebuilding of the entire circuit phasing sometimes proves to be a problem in practice. In practice the engineer does not always want to re-phase the entire circuit.

Another phase balancing study has been reported by Zhu [1.24]. Zhu uses a mixed-integer programming technique. The objective of this technique is to find the optimal phasing patterns so that unbalanced current flows at the "monitored branches" are minimized. The unbalanced flow for a monitored branch is defined as the maximum difference seen among any pair of three-phase currents. Once the problem is formulated, an optimization software package is used to solve the problem. In a later study [1.25], he defines a non-linear objective function that involves the number of phase moves and flows in the branches. Then, he uses simulated annealing as the optimization method to solve the nonlinear, integer- programming problem of phase balancing.

In regard to the phase prediction problem of distribution systems, no previous study has been reported in the literature. This work will be the first that attempts to solve the phase prediction problem. The solution algorithm to be developed involves tabu search [1.26], [1.27], [1.28] method as the optimization tool to find the unknown phasing combinations. The literature survey indicates that tabu search has been applied to solve various power systems problems such as capacitor placement [1.29], reactive power optimization and planning [1.30], unit commitment [1.31] (this paper uses a hybrid of genetic algorithm and tabu search), allocation of new feeders and routes for distribution system expansion planning [1.32], service restoration at the areas that are outaged due to faults in distribution systems [1.33] and so on.

Several different optimization approaches were initially considered [1.34]. The tabu search (TS) method was selected because of the following reasons:

- Features that help avoid getting stuck in local minimums
- Features that help avoid numerical cycles
- Parameters, such as tabu tenure, whose values may be determined by experimentation for a class of problems
- Speed
- Ease of programming

In Sections 1.3.3 and 1.3.4, we have briefly explained two design studies in addition to the phase balancing and the phase prediction: capacitor placement and reconfiguration. These two are popular topics of optimization in electrical distribution systems. Although it is not the intention of this study to develop a capacitor placement or a reconfiguration algorithm, a literature survey reveals that these topics have been studied by many researchers. To name a few, approaches including genetic algorithms [1.35], fuzzy techniques [1.36], tabu search [1.29], knowledge-based programming [1.37], and traditional optimization [1.38] and heuristic search techniques [1.39] have been developed for solving the capacitor placement problem. Similarly, a variety of techniques and procedures including non-linear programming [1.40], approximate power flow methods [1.41], [1.42], combinatorial optimization techniques of genetic algorithms [1.43], and simulated annealing [1.44], knowledge-based expert systems [1.45], heuristic search strategies [1.46], transshipment formulation [1.47] and artificial neural networks [1.48] have been applied to solve the problem of distribution system reconfiguration for normal/contingency situations.

Even though few phase balancing algorithms and many capacitor design and reconfiguration algorithms are present for electrical distribution, no reference that attempts to integrate any two or more of these algorithms can be found at this point in the literature review. That is, in every algorithm surveyed, a specific problem such as the loss minimization problem is attempted through using a single type of operation such as capacitor placement. No work has been proposed to use more than one type of operation such as using capacitor placement together with phase balancing for solving a particular problem of, for example, loss minimization.

In regard to software aspects, this work strongly encourages object-oriented technology for modeling and analysis of distribution systems. Object-oriented technology has become popular in the last few years and now is being adopted increasingly by many software developers. Over the last decade, a number of power system problems, which were investigated by many researchers in the past, have been re-engineered from the object-oriented point of view.

For instance, Zhou [1.49] discusses an efficient object oriented environment for power systems simulation applications along with object-oriented based load flow problems. Manzoni et al. [1.50] considers applying the object-oriented approach to the problem of power system dynamic simulation. The work in [1.51] discusses an object-oriented implementation of load flow. Similarly, an object-oriented modeling of power system components and sparse matrix operations is presented in [1.52] The paper [1.53] addresses an object-oriented approach to develop a software package that will help to restore the system after a wide system outage. Foley et al. [1.54] suggests a method for on-line network analysis using physically based objects.

A number of architectures for dealing with integrated software for power systems have been addressed in the literature. The views taken to consider the software differ in the proposed approaches. Some consider a particular part of the whole system while others take a comprehensive point of view. Flinn [1.55] addresses a database system capable of supporting a diverse set of power system simulation applications ranging from harmonics analysis to load flow calculations. The article [1.56] presents a system solution approach for electric distribution management systems that meets the requirements imposed by industry trends and the electric utility customer. That work is aimed at developing high levels of flexibility, life-time costs, extending product life cycles, more efficient end-use and the ability to evolve over time.

Another work [1.57] proposes an architecture that provides the basis for a successful evolution of a large-scale distribution management system (DMS) over long periods of time. Built on heavily object-oriented ideas, the modeling technique presented addresses the particular needs of DMS such as automation functions.

Another architecture for energy management system (EMS) applications has been addressed by Ma [1.58]. His work includes technologies such as Internet/Intranet and relational databases to handle deregulation and automation challenges. A similar study that considers the electric energy market model (transmission access), power system model and analysis models has been conducted by [1.59].

Foley et al. in [1.60] addresses an object-oriented GUI (graphical user interface) system that might be used as a front-end for power system operation and control. The GUI system is aimed to be suitable for EMS or operator training simulators, and is portable over a wide range of hardware platforms.

A comprehensive study has been presented by Zhu and Lubkeman in [1.61]. They address object-oriented modeling of power system components along with the description of a object-oriented modeling methodology. Also, they propose a distributed software architecture that consists of application modules, a utility database, a SCADA real-time database, and graphical user interface.

1.7 Dissertation Outline

The remainder of this dissertation is organized into four chapters. Chapter 2 describes a phase balancing algorithm. The algorithm has been developed to handle the practical aspects inherent to the phase balancing problem as discussed in Section 1.3.1. Some examples are presented to illustrate the algorithm.

Chapter 3 covers another algorithm that is developed to assist utilities in dealing with the phase prediction problem. A discussion of how to define an objective function and, hence, to measure and evaluate the merit of a possible solution is presented. Chapter 3 also gives a brief discussion on the mechanics of tabu search. Then it describes how to tailor a tabu search method to solve the phase prediction problem. Examples are presented. It is pointed out that phase balancing is a special case of phase prediction.

Chapter 4 considers software aspects of distribution engineering analysis and design software systems. The discussion begins with describing software architecture for distribution analysis and design. Then follows a description of the integrated design idea including a simple example. Finally, the architecture is reconsidered to incorporate the integrated design concept, and some conclusions are drawn.

Chapter 5 gives conclusions concerning the performance of the algorithms developed, lists the contributions of this research effort, and outlines possible future work.

1.8 References

[1.1] A. C. Monteith and C. F. Wagner (eds), *Electrical Transmission and Distribution Reference Book*, *4th Edition: Seven Printing,* Westinghouse Electric Company, Pittsburgh, PA, 1964.

[1.2] J. K. Dillard (ed.), *Electric Utility Engineering Reference Book: Distribution Systems*, Westinghouse Electric Company, Pittsburgh, PA, 1965.

[1.3] H. L. Willis, *Power Distribution Planning Reference Book*, Marcell Dekker, Inc., New York, 1997.

[1.4] M. Baran, "Challenges in Distribution System Analysis," *Proceedings of 1999 IEEE Power Engineering Society Summer Meeting (Cat. No. 99CH36364)*, Piscataway, NJ, USA, vol. 2, pp.627.

[1.5] M. V. Engel (ed.) et al., *IEEE Distribution Planning Tutorial*, IEEE text 92WHO 361-6 PWR, Institute of Electrical and Electronics Engineers, New York, 1992.

[1.6] P. A. Dolloff, *Optimization in Electrical Distribution Systems: Discrete Ascent Optimal Programming*, Ph.D. Dissertation, Electrical Engineering Department, Virginia Polytechnic Institute and State University, Blacksburg, VA, February 1996.

[1.7] T. E. McDermott, *A Heuristic Nonlinear Constructive Method for Electric Power Distribution System Reconfiguration*, Ph.D. Dissertation, Electrical Engineering Department, Virginia Polytechnic Institute and State University, Blacksburg, VA, April 1998.

[1.8] R. Broadwater, J. Thompson, M. Ellis, H. Ng, N. Singh and D. Loyd, "Application Programmer Interface for the EPRI Distribution Engineering Workstation," *IEEE Trans. Power Systems*, vol. 10, no. 1, February 1995, pp. 499-505.

[1.9] R. Pooley and P. Stevens, *Using UML: Software Engineering with Objects and Components*, Addison-Wesley Longman, 1999.

[1.10] G. Booch, J. Rumbaugh and I. Jacobson, *The Unified Modeling Language User Guide*, Addison-Wesley Longman, 1998.

[1.11] D. D'Souza and A. C. Wills, *Objects, Components, and Frameworks With UML*, Addison-Wesley Longman, 1998.

[1.12] D. Leffingwell, D. Widrig and E. Yourdon, *Managing Software Requirements: A Unified Approach*, Addison-Wesley Object Technology Series, November 1999.

[1.13] W. Pree, *Design Patterns for Object-Oriented Software Development*, Addison-Wesley, 1995.

[1.14] G. Booch, *Object-Oriented Analysis and Design With Applications, 2nd Edition*, Addison-Wesley, February 1994.

[1.15] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W. Lorensen, *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, 1991.

[1.16] P. Coad and E. Yourdon, *Object-Oriented Analysis*, Prentice-Hall, Inc., 1991.

[1.17] P. Coad and E. Yourdon, *Object-Oriented Design*, Prentice-Hall, Inc., 1991.

[1.18] C. S. Horstmann, *Mastering Object-Oriented Design in C++*, John Wiley and Sons, Inc., 1995.

[1.19] R. C. Lee and W. M. Tepfenhart, *UML and C++, A practical Guide to Object-Oriented Development*, Prentice-Hall, Inc., 1997.

[1.20] F. Kon and R. H. Campbell, "Dependence Management in Component-Based Distributed Systems," *IEEE Concurrency*, vol. 8, no. 1, January-March 2000, pp. 26-36.

[1.21] W. Kozaczynski and G. Booch, "Component-Based Software Engineering," *IEEE Software*, vol. 15, no. 5, September/October 1998, pp. 34-36.

[1.22] L. G. Osterlund, "Component Technology," *IEEE Comp. Applications in Power*, vol. 13, no. 1, January 2000, pp. 17-25.

[1.23] J. Voas, "Maintaining Component-Based Systems," *IEEE Software*, vol. 15, no. 4, July/August 1998, pp. 22-27.

[1.24] J. Zhu, M. Y. Chow and F. Zhang, "Phase Balancing Using Mixed-Integer Programming," *IEEE Trans. Power Systems*, vol. 13, no. 4, November 1998, pp. 1487-1492.

[1.25] J. Zhu, G. Bilbro and M. Chow, "Phase Balancing using Simulated Annealing," *IEEE Trans. Power Systems*, vol. 14, no. 4, November 1999, pp. 1508-1513.

[1.26] F. Glover, "Tabu Search – Part I," *ORSA Journal on Computing*, vol. 1, no. 3, Summer 1989, pp. 190-206.

[1.27] F. Glover, "Tabu Search – Part II," *ORSA Journal on Computing*, vol. 2, no. 1, Winter 1990, pp. 4-32.

[1.28] F. Glover, E. Taillard, and D. de Werra, "A User's Guide to Tabu Search," *Annals of Operations Research*, vol. 42, 1993, pp. 3-28.

[1.29] Y. C. Huang, H. T. Yang and C. L. Huang, "Solving the Capacitor Placement Problem in a Radial Distribution System Using Tabu Search Approach," *IEEE Trans. Power Systems*, vol. 11, no. 4, November 1996, pp. 1868-1873

[1.30] D. Gan, Z. Qu and H. Cai, "Large-Scale Var Optimization and Planning by Tabu Search," *Electric Power Systems Research*, vol. 39, no. 3, December 1996, pp. 195-204.

[1.31] A. H. Mantawy, Y. L. Abdel Magid and Z. S. Selim, "New Genetic-Based Tabu Search Algorithm for Unit-Commitment Problem," *Electric Power Systems Research*, vol. 49, no. 2, March 1999, pp. 71-78.

[1.32] K. Nara, Y. hayashi, S. Muto and K. Tuchida, "New Algorithm for Distribution Feeder Expansion Planning in Urban Area," *Electric Power Systems Research*, vol. 46, no. 3, September 1998, pp. 185-193.

[1.33] S. Toune, H. Fudo, T. Genji, Y. Fukuyama and Nakanishi, "Reactive Tabu Search for Service Restoration in Electric Power Distribution Systems," *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, 98TH8360, pp. 736-768.

[1.34] C. R. Reeves (ed.), *Modern Heuristic Techniques for Combinatorial Problems*, J. Wiley, New York, 1993.

[1.35] G. Boono and D. C. Hsiao, "Optimal Capacitor Placement in Distribution Systems by Genetic Algorithm*," International Journal of Electrical Power and Energy Systems*, vol. 15, no. 3, June 1993, pp. 155-62.

[1.36] H. N. Ng, M. M. A. Salama and A. Y. Chikhani, "Capacitor Placement in Distribution Systems Using Fuzzy Technique," *1996 Canadian Conference on Electrical and Computer Engineering Conference Proceedings Theme: Glimpse into the 21st Century*, IEEE, New York, USA, 1996, vol. 2, pp. 790-3.

[1.37] T. Taylor and D. Lubkeman, "Applications of Knowledge-Based Programming to Power Engineering Problems," *IEEE Transactions on Power Systems*, vol. 4, no. 1, Feb. 1989, pp. 345-352.

[1.38] S. Jovanovic, B. Gajic and S. Mijailovic, "Optimal Placement and Sizing of Capacitors in Large Industrial Systems," *Modeling, Simulation and Control*, vol. 36, no.3, 1991, pp. 45-64.

[1.39] M. Chis, M. M. A. Salama and S. Jayaram, "Capacitor Placement in Distribution Systems Using Heuristic Search Strategies," *IEE- Proceedings Generation Transmission and Distribution*, vol. 144, no. 3, May 1997, pp. 225-230.

[1.40] C. C. Liu, S. J. Lee and K. Vu, "Loss Minimization of Distribution Feeders: Optimality and Algorithms," *IEEE Transactions on Power Delivery*, vol.4, no.2, 1989, pp. 1281-89.

[1.41] S. G. Civanlar, J. J. Gringer, H. Yin and S. S. H. Lee, "Distribution System Reconfiguration for Loss Reduction," *IEEE Transactions on Power Delivery*, vol. 3, no. 3, 1988, pp. 1217-1223.

[1.42] M. E. Baran and F. F. Wu, "Network reconfiguration in Distribution Systems for Loss Reduction and Load Balancing," *IEEE Transactions on Power Delivery*, vol. 4, no.2, 1989, pp. 1401-7.

[1.43] K. Nara, A. Shiose, M. Kitagawa, and T. Ishihara, "Implementation of Genetic Algorithm for Distribution Systems Loss Minimum Reconfiguration," *IEEE Trans. Power Systems* vol. 7, no. 3, 1992, pp. 1044-1051.

[1.44] K. Aoki, T. Ichimori and M. Kanezashi, "Normal State Optimal Load Allocation in Distribution Systems," *IEEE Transactions on Power Delivery*, no.1, 1987, pp. 147-155.

[1.45] C. C. Liu, S. C. Lee and .S. S. Venkata, "An Expert System Operational Aid for Restoration and Loss Reduction of Distribution Systems," *IEEE Transactions on Power Delivery*, vol.3, 1988, pp. 619-625.

[1.46] A. L. Morelato and A. Monticelli, "Heuristic Search Approach to Distribution System Restoration," *IEEE Transactions on Power Delivery*, vol. 4, no.4, 1989, pp.2235-41.

[1.47] V. Glaocanin, "Optimal Loss Reduction of Distribution Networks," *IEEE Transactions on Power Systems*, vol. 5, no. 3, Aug. 1990, pp. 774-782.

[1.48] H. Kim, Y. Ko and K. Jung, "Artificial Neural Network Based Feeder Reconfiguration for Loss Reduction in Distribution Systems," *IEEE Transactions on Power Delivery*, vol.8, no.3, 1993, pp. 1356-1366.

[1.49] E. Z. Zhou, "Object -Oriented Programming, C++ and Power System Simulation," *IEEE Trans. Power Systems*, vol. 11, no.1, February 1996, pp. 206-215.

[1.50] A. Manzoni, A. S. Silva and I. C. Dekker, "Power System Dynamic Simulation Using Object-Oriented Programming," *IEEE Trans. Power Systems*, vol. 14, no. 1, February 1996, pp. 249-55.

[1.51] A. Neyer, F. F. Wu, and K. Imhof, "Object-Oriented Programming for Flexible Software: Example of a Load Flow," *IEEE Trans. Power Systems*, vol. 5, no. 3, August 1990, pp. 689-696.

[1.52] B. Hakavik and A. T. Holen, "Power System Modeling and Sparse Matrix Operations Using Object-Oriented Programming," *IEEE Trans. Power Systems*, vol. 9, no. 2, May 1994, pp. 1045- 1051.

[1.53] Z. L. Gaing, C. N. Lu, B. S. Chang and C. L. Cheng, "An object-Oriented Approach for Implementing Power System Restoration Package," *Proceedings of 1995 IEEE Power Industry Computer Applications Conference*, 95CH35798, pp. 467-473.

[1.54] M. Foley and A. Bose, "Object-Oriented Online Network Analysis," *IEEE Trans. Power Systems*, vol. 10, no. 1, February 1995, pp. 125-132.

[1.55] D. G. Flinn and R. G. Dugan, "A Database for Diverse Power System Simulation Applications," *IEEE Trans. Power Systems*, vol. 7, no. 2 May 1992, pp. 784-790.

[1.56] K. I. Geisler, S. A. Neumann, T. D. Nielsen, P. K. Bower and B. A. Hughes, "A Generalized Information Management System Applied to Electrical Distribution," *IEEE Computer Applications in Power*, vol. 4, no. 3, July 1990, pp. 9-13.

[1.57] J. Britton, "An Open, Object-Based Model as the Basis of an Architecture for Distribution Control Centers," *IEEE Trans. Power Systems*, vol. 7, no. 4, November 1992, pp. 1500-1508.

[1.58] J. T. Ma, T. Liu, L. Wu, "New Energy Management System Architectural Design and Intranet/Internet Applications to Power Systems," *Proceedings of 1998 IEEE Energy Management and Power Delivery Conference*, vol. 2, 98EX137, pp. 559-563.

[1.59] E. Handschin, M. Heine, D. Konig, T. Nikodem, T. Seibt and R. Palma, "Object-Oriented Software Engineering for Transmission Planning in Open Access Schemes," *IEEE Trans. Power Systems*, vol. 13, no. 1, February 1998, pp. 94-100.

[1.60] M. Foley, A. Bose, W. Mitchell and A. Faustini, "An Object-Based Graphical User Interface for Power Systems," *IEEE Trans. Power Systems*, vol. 8, no. 1, February 1993, pp. 97-104.

[1.61] J. Zhu and D. L. Lubkeman, "Object-Oriented Development of Software Systems for Power System Simulations," *IEEE Trans. Power Systems*, vol. 12, no. 2, May 1997, pp. 1002-1007.

Chapter 2 Phase Balancing Application

In this chapter we will present an algorithm for phase balancing of large-scale unbalanced radial distribution systems. The algorithm recommends re-phasing of single- and double-phase laterals in order to improve circuit loss while also maintaining/improving imbalances at various *balance point* locations. Phase balancing calculations are based on circuit loss information and current magnitudes which are calculated from a power flow solution. The algorithm is designed to handle time varying loads when evaluating phase moves that will result in improved circuit losses over all load points. The proposed algorithm is heuristic in nature, and thus does not necessarily achieve the global minimum loss point. Example problems illustrating the algorithm are presented including a circuit that has 3894 components with 170 laterals.

2.1 Introduction

Due to single-phase loads, two-phase loads, unbalanced three-phase loads, and unbalanced impedances, the distribution system operates in an unbalanced fashion. Unbalanced phase loading is perhaps the most important cause of unbalanced operations during normal conditions.

Imbalance has an affect on system performance measures as indicated below.

- In an unbalanced system the peak load that may be supplied is reduced. This reduction is primarily due to the most heavily loaded phase reaching maximum allowable ampacity of the serving equipment or conductors, and possible undesirably low voltage on the heavily loaded phase. Relative to equipment utilization, phase balancing can extend the useful life of a substation transformer, which is generally one of the most expensive pieces of equipment in the circuit.

- Due to the quadratic nature of losses, the losses in the heavily loaded phase increase rapidly with imbalance. Also, the losses increase due to the additional current in the return path.

- A severely unbalanced circuit can result in excessive voltage drops on the heavily loaded phase, and thus low voltage complaints.

- System protection is likely to be affected by unbalanced operations. For instance, substation relays sometimes trip on circuit imbalance, such as may occur on a cold morning with most of the residential load connected to one phase which is on and most of the commercial load connected to another phase which is off. This type of problem may be avoided by properly balancing the different load types across the phases.

Phase balancing is one approach to correcting the imbalance problems in distributions systems. Phase balancing in our study is defined as determining the phasing manner in which single- or double-phase laterals should be connected to the three-phase portion of a circuit in order to obtain a better balance. In other words, phase balancing is concerned with re-phasing single- or double-phase laterals in distribution systems.

There are a number of practical aspects that should be considered while performing a phase balancing study on a distribution system. As pointed out in Section 1.3.1, one aspect is that phase moves that are performed in an effort to improve the imbalance of the substation should not worsen the imbalances at some internal locations. Such internal locations are those switch locations where circuit segments are transferred from one feeder to another. If a transfer occurs at a location that was previously worsened, then the transferred section will damage the imbalance of the substation that it is being transferred to.

Another aspect involved in phase balancing is that the number of phase move recommendations should be kept as low as possible because of the time and labor constraints of the utility. Besides, more phase moves being performed result in more customers being interrupted. Most of the time a utility is interested only in a couple of best moves that will lead to improved system operation.

In this chapter, a phase balancing algorithm, with an emphasis given to the practical aspects addressed above, will be developed. The algorithm enables the user to specify locations where the imbalances do not deteriorate during the course of phase balancing. Also, it allows the user to indicate a limit on the number of phase move operations that will be performed by phase balancing. The utility might not want to perform any phase moves on certain laterals for some reason (For example, it may be too hard for the field crew with a bucket truck to access some lateral connections). The algorithm proposed also permits such locations to be marked as ‘not moveable’ so that phase balancing will not attempt to perform any phase moves on these locations.

Furthermore, phase balancing for a single loading condition in winter may not work well for summer conditions, or even for other winter loading conditions. The algorithm developed in this work may be used to design the phase balance over time varying load patterns on very large circuits

Subsequent sections will describe the phase balancing algorithm proposed. Simple examples of using the algorithm will then be discussed. An additional test case run on a model of an existing, large utility circuit is presented.

2.2 Definition of Terms

Phase balancing design makes use of the terms "deviation" and "imbalance" which are calculated as follows:

$$I_{ave} = \frac{\sum_i I_i}{sNumPh}$$

$$deviation_i = \frac{I_i}{I_{AVE}} - 1$$

$$imbalance = \max\left\{\left|deviation_i\right|\right\}$$

where:

i	= phase present, $i \in \{A, B, C\}$
I_i	= magnitude of phase i current
I_{AVE}	= average phase current
sNumPh	= number of phases present

Notice that deviation is defined for each phase. It is a measure of how much a phase current is below or above the average phase current. That is, deviation ranges from -1 to 2. A deviation of -0.5 for the ith phase current would indicate that it is running 50% below the average. Likewise, a deviation of 0.5 would indicate 50% above the average. The ideal deviation would be zero. If one phase carries all of the current, then its deviation is 200% above the average.

Imbalance is defined as the absolute value of the worst deviation. Thus, the imbalance ranges from 0 to 2, where 0 is perfectly balanced and 2 is perfectly-out-of-balance. The perfectly balanced case occurs when currents in all phases are equal. Perfectly-out-of-balance results if there is only one phase that carries current while the other two phases have no current.

Note that the way deviation and imbalance is defined in this study differs from that of IEEE. IEEE defines deviation of a phase load as an absolute value of the difference between phase load and average load. In calculating imbalance, maximum deviation is scaled by the average load and then multiplied by 50%. The imbalance value thus obtained ranges from 0% to 100%. IEEE definitions of imbalance and deviation can be found in the Appendix.

There are two types of circuit locations that have special meaning to the phase balancing algorithm:

- *balance points*
- *move points*

Besides the substation, only three-phase switches may be *balance points*. A three-phase switch is identified as a *balance point* if the imbalance at the switch is to be used for evaluation of balance affects due to changes in lateral phasing. That is, a *balance point* is defined as a three-phase switch location the imbalance of which is desired not to deteriorate during the course of phase balancing. For instance, refer to the distribution circuit represented in Figure 2.1. For that circuit, one might consider/designate the switch SW as a *balance point* for phase balancing.

A single- or double-phase component that is 'moveable' and connected to a three-phase component is called a *move point*. A lateral that is considered for moving is made up of a *move point* and all downstream components from that *move point*. Hence, a *move point* can be conceived as the "head of a moveable lateral." A double-phase lateral may contain single-phase components in addition to double-phase components. However, the *move point* of such a lateral is always a double-phase component. L2 in Figure 2.1 is an example of a double-phase lateral which consists of three double-phase and two single-phase components.

Re-phasing a lateral means consistently changing the phase(s) of the *move point* and all downstream components belonging to the lateral. Thus, when a *move point* is re-phased, all subsequent components in the lateral are re-phased consistent with the changes made at the *move point*.

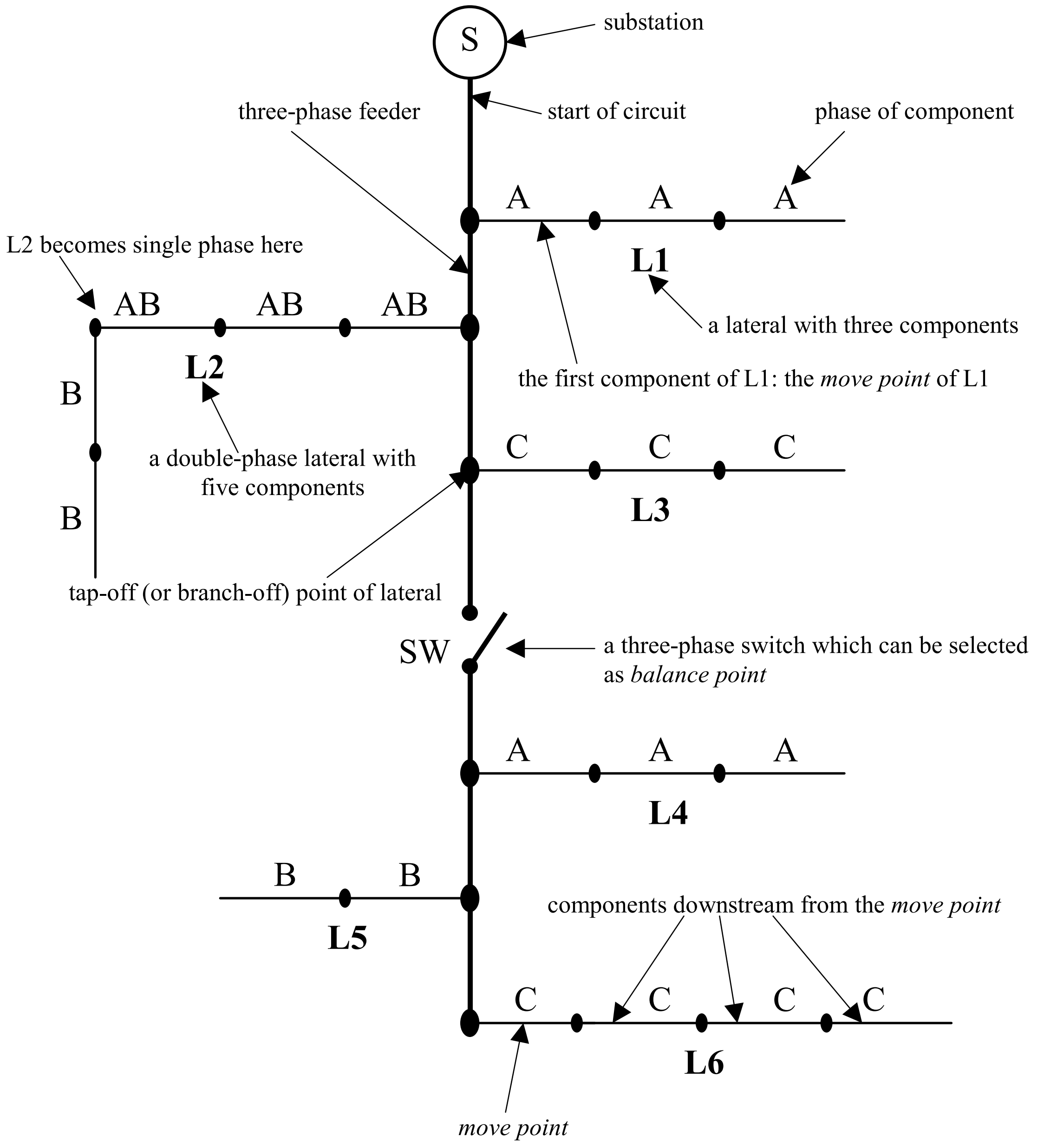

S
substation
three-phase feeder
start of circuit
phase of component
A A A
L1
L2 becomes single phase here
AB AB AB
a lateral with three components
L2
B
the first component of L1: the *move point* of L1
C C C
a double-phase lateral with five components
B
L3
tap-off (or branch-off) point of lateral
SW
a three-phase switch which can be selected as *balance point*
A A A
L4
B B
components downstream from the *move point*
L5
C C C C
L6
move point

Figure 2.1 A distribution circuit for the illustration of laterals, *move points* and *balance points*.

For instance, when phase A of a *move point* is changed to phase B, all downstream phase-A lateral components are changed to phase B as well.

For a single-phase lateral there are two re-phasing alternatives, whereas a two-phase lateral may have as many as five re-phasing alternatives. For example, a *move point* for a single-phase lateral that is originally set to phase A can be re-phased as either phase B or C.

A two-phase *move point* that originally has phase A in position 1 and phase B in position 2, and no three-phase motor loads present, can potentially have any of the following re-phasings:

- AC
- BA
- BC
- CB
- CA

The first letter used in re-phasings denotes the phase of position 1 and the second letter represents the phase of position 2, and position refers to a construction location. Note that if three-phase motor loads are present, changing the phase sequence on the motor could cause harm to the motor.

For each *move point*, re-phasing information is stored in an array of size three referred to as the "movement logic array". In the movement logic array element 0 is associated with phase A, element 1 with phase B, and element 2 with phase C (where 0-based indexing is used as in C++). The value associated with each element shows the re-phasing information for the phase associated with that element. For instance, assume that r[3] represents the movement logic array for a *move point*. Let the values stored in r[3] be given by

r[0] = -1
r[1] = 2
r[2] = -1.

The values above are interpreted as phases A and C not being present on the original lateral, and that phase B should be changed to phase C. Legal values for an element of the movement logic array are -1,0,1 and 2 where:

-1 => phase was not present.
0 => phase should be moved to phase A.
1 => phase should be moved to phase B.
2 => phase should be moved to phase C.

Associated with each *move point* there is another term called "phase movement order index," or *order index* for short. The move that is to be performed first is given an *order index* of 0; the move that is to be performed next is given an *order index* of 1, and so forth. The *order index* is used to track the order in which lateral phase moves are to be performed. If the *order index* for a *move point* is set to –1, then no phase movement is recommended at that *move point*

Having defined basic phase balancing terms, let us proceed to describe the algorithm in the next section.

2.3 The Proposed Algorithm: An Overview

The proposed phase balancing algorithm recommends re-phasing of single- and double-phase laterals to improve circuit loss while also maintaining or improving phase balance at various *balance points* throughout the circuit. The substation is always a *balance point*. In addition to the substation, the user can designate any three-phase switch as a *balance point*.

The phase balancing application selects, from a finite set of phasing patterns, the phasing pattern of single- and two-phase laterals that results in the minimum loss for the set considered. Since circuit losses are closely related to unbalanced currents, improved phase balance results as a natural side effect when laterals are re-phased to reduce losses. Members of the finite set of phasing patterns result from considering one phase move at a time over the set of *move points*. This will be described further below. An exhaustive search can not be performed because the number of possibilities is too large.

The phase balancing algorithm is heuristic in nature, and thus does not necessarily achieve the global minimum loss point. However, the algorithm has been tested on a number of circuits, both small and large, and has always determined lateral movements that improve circuit efficiencies. Example circuits are presented below.

The phase balancing algorithm is incorporated in an integrated environment [2.1] in which it can communicate with other programs such as power flow. That is, it can make power flow run and then it can retrieve the results of the power flow run. Similarly, the ability that the user can mark a lateral as ‘not moveable’ is provided by this integrated environment.

The phase balancing algorithm can take into account time-varying loads when evaluating phase moves. In performing this task, phase balancing runs power flow for each load point and retrieves circuit losses associated with each load point. Phase balancing then sums the losses over all load points to obtain the total loss over the time-varying load curve. Thus, phase balancing can find the phase moves that lead to minimized circuit losses over a time varying load curve of any length. However, practically the integrated environment, in which the phase balancing program runs, restricts the analysis to 8760 load points (or hours) per year for a non-leap year.

Another capability of the phase balancing algorithm is that it enables the engineer to specify the "maximum allowable number of phase moves." If no limit on the number of phase moves is taken into account by the phase balancing application, a large number of phase moves may be recommended, which can not be afforded by the utility. Also, the solution of such a large problem over a time-varying load curve may involve extensive calculational time. Hence, the utility is generally interested in a small number of best moves, which may be specified as the "maximum allowable number of phase moves." The next paragraph describes a curve that may be used to check if the maximum number of allowable phase moves has been set properly.

The phase balancing application provides a plot that shows the decrease in circuit loss versus phase move, where the phase moves are plotted by increasing *order index*. In general, this plot exhibits an increasing curve such that the incremental increase in circuit loss gets smaller with each successive phase move. Thus, the curve may be used to estimate the number of phase moves to be performed to provide the greatest economic benefit. After a certain number of phase moves, succeeding phase moves may result in an insignificant improvement in circuit loss. Ideally, the maximum number of allowable phase moves described in the preceding paragraph should be set such that on the last recommended phase move the improvement in circuit loss is insignificant.

2.4 Algorithm Steps

Inputs to the algorithm consist of

- circuit to be analyzed (where, if any, the 'not moveable' locations have been included)
- the maximum number of allowable phase moves
- *balance point* locations.

The key variables used in the phase balancing algorithm are tabulated in Table 2.1 and are referred to in the following discussion where the algorithm is explained in detail. Also, Figure 2.2 can be referred for the overview of the algorithm steps.

Step 1:

The number of *balance points*, *nBalancePt*, is determined in this step. By default, the start of circuit, located at the substation, is always a *balance point*. Besides, the engineer can specify any three-phase switch as *balance point*. Usually these switches are operated to move load from one circuit to another circuit, and it is desired to balance the load on the circuit segment that is being switched.

Table 2.1 Variables Used by Phase Balancing (first column: variable name, second column: description of variable).

Name	Description
$BalancePt_i$	i^{th} *balance point* , i=1,2,3,..., *nBalancePt*
$BalPtImbal_i$	imbalance of $BalancePt_i$
$CktLoss_i$	total circuit loss when $MovePt_i$ is re-phased
$DevAfter_i$	deviations of phase currents for the three-phase component *i* after phase balancing
$DevBefore_i$	deviations of phase currents for the three-phase component *i* before phase balancing
$DevChange_i$	Changes in deviation of phase currents for three phase component *i*
$MovePt_i$	i^{th} *move point* , i=1,2,3,..., *nMovePt*
nBalancePt	total number of *balance points*
nMoveOrder	order of phase move
nMovePt	total number of *move points*
nPhaseMoves	maximum allowable number of phase moves
$OrderIndex_i$	phase movement order index associated with $MovePt_i$
OrigLoss	total circuit loss before phase balancing
PickWorthy	100*(1 - *PickWorthy*) is the minimum percent decrease in total circuit loss in order for a phase move to be performed
$PhMove_i$	movement logic array associated with $MovePt_i$
TempLoss	total circuit loss that is set at each step to the sum of circuit losses over all load points

```
Step 1: Read input; $nPhaseMoves$, $BalancePt_i$, where $i$=1,2,..., $nBalancePt$.
Step 2: Calculate $nMovePt$. If $nMovePt$=0, STOP.
Step 3: Initialize $OrigLoss$, $TempLoss$ and $BalPtImbal_i$, i=1,2,…,$nBalancePt$.
Step 4: Record $DevBefore_i$ for every three phase component $i$.
Step 5: Initialize:
        $nMoveOrder$ = 0; $OrderIndex_i$ = -1 where $i$=1,2,...,$nMovePt$.
    Step 6: For each $MovePt_i$ where $OrderIndex_i$ =-1:
            • Initialize $CktLoss_i$=TempLoss.
            • Find $PhMove_i$ (and $CktLoss_i$ associated with $PhMove_i$ ) that
              minimizes total circuit loss such that :
                ▪ It gives a circuit loss less than ($PickWorthy*TempLoss$), and
                ▪ None of the balance point imbalances gets worse.
            • Rephase $MovePt_i$ back to its original phasing after evaluating
              all alternatives of $MovePt_i$.
            • If such $PhMove_i$ is found, set $CktLoss_i$.
    Step 7: If no $MovePt_i$ from Step 6 has
            $CktLoss_i$ < ($PickWorthy*TempLoss$), GO TO Step 10.
    Step 8: Select the minimum among $CktLoss_i$ of Step 6. Let it be
            $CktLoss_j$. Then,
            • Permanently rephase $MovePt_j$ according to $PhMove_j$
            • Update $TempLoss = CktLoss_j$ .
            • Set $OrderIndex_j = nMoveOrder$.
            • Set $nMoveOrder = nMoveOrder + 1$ .
    Step 9: If ($OrderIndex_k$ = -1 for any $k$=1,2,...,$nMovePt$ ) AND
            ($nMoveOrder < nPhaseMoves$) THEN,
            • Update $BalPtImbal_l$ for $l$=1,2,...,$nBalancePt$.
            • Go to Step 6.
            ELSE
            • Go to Step 11.
Step 10: If ($nMoveOrder$ = 0 ) ; STOP. (Could not find any improving move.
         Leave the circuit as is.)
Step 11: Record $TempLoss$, $DevAfter_i$ and $DevChange_i$ for every three-phase
         component $i$.
```

Figure 2.2 Phase balancing algorithm steps: Indented Steps 6-9 represent iterative part of solution.

<u>Step 2:</u>

The number of *move points*, *nMovePt*, is determined in this step. In determining the number of *move points*, a reverse trace is performed [2.2]. In the reverse trace, every time a step from a single- or two-phase component to a three-phase component occurs, then the single- or two-phase component is marked as a *move point* (i.e., unless the single- or two-phase component has been marked by the engineer not to be moved). The total number of *move points* marked is stored in *nMovePt*.

If no *move points* are found in the circuit, there are no laterals that may be rephased, and the algorithm exits and reports that there are no lateral movements to be performed.

<u>Step 3:</u>

In this step the total circuit loss over the time-varying load curve, *OrigLoss*, is determined. This represents the circuit loss prior to any phase movements being performed. This also represents the total energy loss if load is assumed to stay constant between discrete load points. At the end of this step this original loss is stored in the variable *TempLoss*.

Also in this step, the imbalance of each *balance point i* is calculated and recorded in the variable, $BalPtImbal_i$.

The variables *TempLoss* and $BalPtImbal_i$ will be used in the next steps for comparison purposes. That is, when investigating a re-phasing alternative for a move point, one should consider that alternative as a possible alternative as long as:

- The alternative gives a better circuit loss than *TempLoss*

- The alternative does not result in an imbalance that is worse than $BalPtImbal_i$, over all i.

Step 4:

In this step the deviations in three-phase currents (i.e., from the average current) prior to any phase moves are calculated. The variable, $DevBefore_i$, is used to store the deviations on each three-phase component in the circuit.

Note that $DevBefore_i$ is an array of size three where index 0, 1 and 2 correspond to phase A, B and C respectively.

Step 5:

In this step the phase movement *order index* of each move point i, $OrderIndex_i$, is set to -1. This indicates that the corresponding move point is available for rephasing. If the phase balancing algorithm encounters an *order index* value other than -1, the algorithm does not attempt to rephase the corresponding move point.

Also, the variable, *nMoveOrder*, which keeps track of the number of phase moves that have been performed, is initialized to 0.

Step 6:

In this step every *move point i* that has an *order index* of -1 is considered in order to find the best single phase move that may be performed for the given set of lateral connections. For every *move point* with an *order index* of –1 the following is performed:

The variable, $CktLoss_i$, which stores the total circuit loss when the *move point i* is rephased, is initialized to *TempLoss*. Then rephasing starts by considering every phasing alternative available at *move point i*. For each phasing alternative, the algorithm re-phases the *move point*, runs power flow and obtains the total circuit loss. If **a)** the new total circuit loss is less than *PickWorthy**TempLoss*, where the variable *PickWorthy* represents the merit in order for a phase move to be performed (e.g., *PickWorthy* = 0.99 means that to make a permanent phase move, the phase move must result in a decrease in total circuit loss at least by 1% as compared to the total circuit loss before the phase move) and **b)** the imbalance of any *balance point i* does not get worse (i.e., the new imbalance at $BalancePt_j$ must be equal to or less than $BalPtImbal_j$ for i=1, 2, ... *nBalancePt*), then the following is performed:

- The new lower value of total circuit loss is stored in $CktLoss_i$
- The rephasing logic that lead to the lower value of total circuit loss is stored in the array $PhMove_i$

After all phasing alternatives have been evaluated at *move point i*, the *move point* is re-phased back to its original phasing. The algorithm proceeds to the next *move point* that has an *order index* of –1 until all such *move point*s have been evaluated.

Step 7:

Following Step 6, if $CktLoss_i$ is still equal to *TempLoss* for every *move point i*, then no phase movement will result in reduced losses. If this is the case, the algorithm stops evaluating phase moves and goes to Step 10 where the results are reported.

Step 8:

In this step, among the *move point*s examined in Step 6, the one that gives the minimum circuit loss is selected. Let this *move point* be $MovePt_j$. It is then re-phased permanently according to $PhMove_j$. The term permanently is used to indicate that $MovePt_j$ will stay at its new phasing for the rest of the algorithm. The resultant circuit loss is now $CktLoss_j$. Because the circuit is modified by a permanent rephasing of $MovePt_j$ (and all downstream components from that *move point*), the circuit loss value, *TempLoss*, should be updated to $CktLoss_j$. Thus, the next permanent move to be found (if any) must give a better circuit loss than *TempLoss*.

The *order index* of $MovePt_j$, $OrderIndex_j$, is set to move order, *nMoveOrder* in order to imply that the (*nMoveOrder* + 1)th phase move must be performed on $MovePt_j$ based on the changes given by $PhMove_j$. Also, a non-negative *order index* implies that the *move point* will stay at this new phasing and will never be considered again for re-phasing in all succeeding steps. The order of the phase move, *nMoveOrder*, is incremented at the end of this step.

Step 9:

In this step the algorithm checks the presence of the following two conditions:

- All *move point*s have been rephased, i.e., $nMoveOrder = nMovePt - 1$
- The limit specified by the maximum number of allowable phase moves, *nPhaseMoves*, is reached

The algorithm goes to Step 11 if either of the above conditions exists. If the neither of the conditions exists, then steps 6–8 are repeated. Because the circuit is modified in Step 8, the new imbalances at the *balance points* (i.e., *BalPtImbal*$_l$ where l=1, 2, ... *nBalancePt*) are evaluated before going to Step 6. Hence, the next permanent re-phasing to be performed must lead to an imbalance at least as good as the corresponding ones found in this step for every *balance point* location.

Step 10:

In this step, the value of *nMoveOrder* is checked. If it is 0, then it means that the algorithm could not find any phase move that would result in improved circuit loss without worsening any of the imbalances at *balance points*. In this case, the algorithm will stop by reporting this situation.

Step 11:

Being in this step implies that some permanent phase move(s) have been performed. Therefore, the total circuit loss, *TempLoss*, is calculated and this new value is recorded as the 'circuit loss after phase balancing'. Also, three-phase current deviations for every three-phase component *i* on the new modified circuit are evaluated and recorded as 'current deviations after phase balancing', *DevAfter*$_i$. Then, the change in current deviations are evaluated for each component *i* and stored in an array of size three, *DevChange*$_i$.

The calculation of the variable, *DevChange*$_i$, is performed based on differences in the distances of the currents to the average. That is, after phase balancing is complete, if a phase current becomes closer to average than it was before the phase balancing then the deviation change for that phase current will be positive implying that the deviation has improved. Otherwise, the deviation will be negative which means the deviation has been worsened.

For example, assume that a three-phase component *i* has the following deviations before and after phase balancing:

$DevBefore_i[0] = 0.40$	$DevAfter_i[0] = -0.16$
$DevBefore_i[1] = -0.25$	$DevAfter_i[1] = 0.0$
$DevBefore_i[2] = -0.15$	$DevAfter_i[2] = 0.16$

Phase A current was 40% above the average before phase balancing, and is 16% below the average after phase balancing. Then the deviation is said to have improved by 24% (24%=40%–16%). In other words, change in phase A deviation is +24%. Similarly, Phase B current was 25% below the average before, and is equal to the average after. Hence, phase B current has improved by 25% (25%=25%–0%). Finally, phase C current was 15% below average before, and after it is 16% above average. For this case, the phase C current balance has become worse by 1% (1%=16%–15%). That is, the change in phase C current deviation is –1%. Thus, the change in deviations, $DevChange_i$, is summarized as:

$DevChange_i[0] = +0.24$

$DevChange_i[1] = +0.25$

$DevChange_i[2] = –0.01$

In essence, the following formula is used to evaluate the change in current deviation at phase *j* of a three-phase component *i*:

$$DevChange_i[j] = \text{abs} (DevBefore_i[j]) – \text{abs} (DevAfter_i[j]).$$

2.5 Simulation Examples

The phase balancing algorithm has been incorporated as a design module in the EPRI Distribution Engineering Workstation, DEWorkstation. The algorithm has been tested on a number of unbalanced circuits and has always recommended design changes that resulted in improved performance. Three examples will be presented in this section to illustrate results from the phase balancing algorithm. The first two examples involve small test circuits while the third example considers a large circuit from the Detroit Edison electric utility company.

2.5.1 Example 2.1: Consideration of Balance Point(s)

Consider the simple radial system depicted in Figure 2.3 where the system consists of two substations, *Sub1* and *Sub2*. Let us call the circuit fed by *Sub1* as *Ckt1* and the one fed by *Sub2* as *Ckt2*. Phasing information (represented in circles) and names of each single-phase lateral are given along with the corresponding line. Such representation for three-phase segments is left out for clarity. Every single-phase lateral has some load connected as shown in the figure.

The objective of this example is to illustrate the reason behind the use of multiple *balance points*. Assume that switch SW1 is closed and switch SW2 is open. Assume also that *Ckt2* has already been balanced.

A phase balancing (PB) study will be performed for *Ckt1*. For this, one can consider two approaches:

i. Phase balance with *Sub1* being the only *balance point*
ii. Phase balance with *Sub1* and SW1 being two *balance points*.

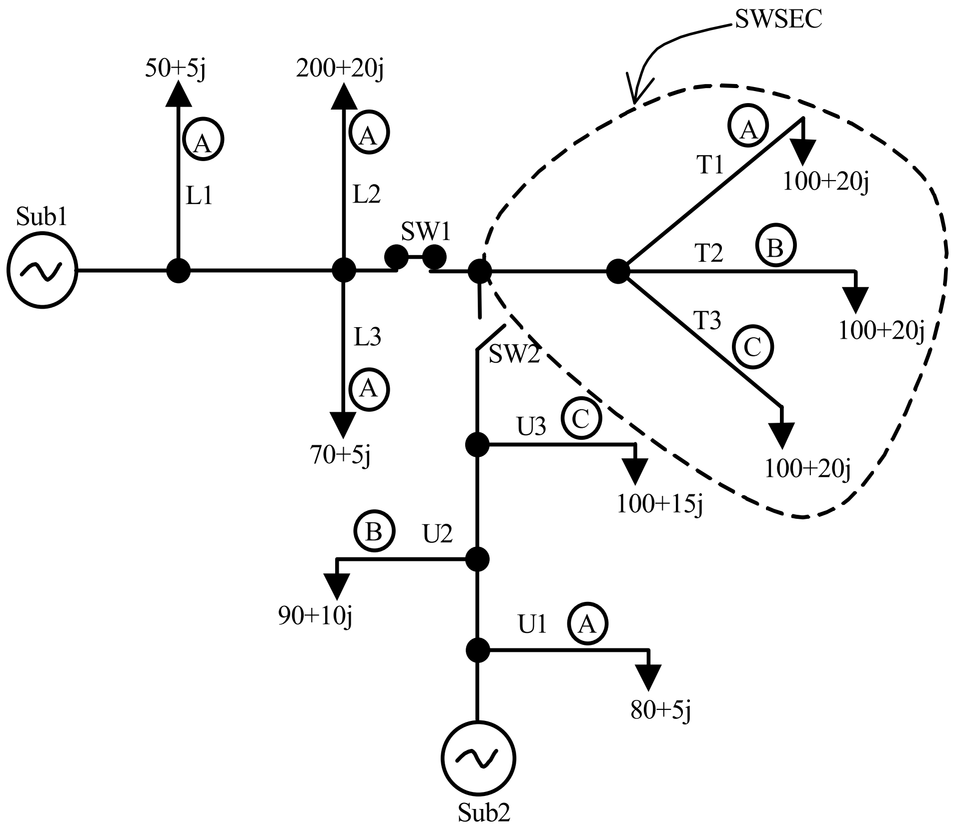


Figure 2.3 System for Example 2.1

Based on the first approach, it will be recommended that L2 be re-phased from A to C and T3 be re-phased from C to B. Note that this approach will damage the imbalance at SW1. Before PB, deviations at SW1 were %0 for A, B and C currents (hence, imbalance was %0). However after PB, deviations will be 0%, 100% and –100% (hence the imbalance will be 1.0) for A, B, and C phase currents receptively.

Based on the second approach, it will be recommended that L2 be re-phased from A to C and L3 be re-phased from A to B. Again note that with this approach, where SW1 is considered to be a balance point, PB does not allow any phase moves beyond SW1 because any such phase move will deteriorate the imbalance of SW1.

Initially, the circuit segment represented by SWSEC is powered by *Sub1*. Now suppose that *Sub1* becomes unable to serve power for some reason. A transfer attempt is performed in order to maintain power at SWSEC. For this, SW1 is opened and SW2 is closed so that SWSEC can obtain power from *Sub2*. One can ask: How does this transfer affect the imbalance at *Sub2*?. Or more specifically:

- Assume *Sub1* has already been balanced based on the first approach. What will be the imbalance at *Sub2* after the transfer of SWSEC?
- Assume *Sub1* has already been balanced based on the second approach. What will be the imbalance at *Sub2* after the transfer of SWSEC?

Table 2.2 gives the answers to these questions. The first column in Table 2.2 represents the substation locations. The second column exhibits deviations and imbalances at these locations before PB is performed. The third column shows the imbalances and deviations that would be observed after the transfer is made and if *Ckt1* were balanced using the first approach given above. The last column displays the deviations and imbalances that would be observed after the transfer is made and if *Ckt1* were balanced using the second approach. A negative deviation indicates a 'below average phase balance,' and a positive deviation indicates an 'above average phase balance'.

Table 2.2 Comparison of PB Results.

Location	**Before PB and Before the Transfer** **SW1:Closed SW2: Open**	**After PB and After the Transfer** **SW1: Open SW2: Closed and SW1 is not a *balance point***	**After PB and After the Transfer** **SW1: Open, SW2: Closed and SW1 is a *balance point***
Sub1	Deviations: A: 102% B: -51% C: -51% imbalance =1.02	Deviations: A: 6.2% B: -2.4% C: -3.8% imbalance =0.062	Deviations: A: -27% B: -17% C: 44% imbalance =0.44
Sub2	Deviations: A: -5.5% B: 0% C: 5.5% imbalance =0.055	Deviations: A: -5.5% B: 53% C: -47.5% imbalance =0.53	Deviations: A: -5.5% B: 0% C: 5.5% imbalance =0.055

Results presented in Table 2.2 reveals that selecting SW1 as a *balance point* will result in the imbalance at *Sub2* being unaffected by the transfer of SWSEC whereas not considering SW1 as balance point will cause the imbalance at *Sub2* to be worsened by the transfer.

2.5.2 Example 2.2: Consideration of Two Loading Scenarios

Consider the distribution circuit depicted in Figure 2.4. As shown in the figure the circuit consists of single-phase laterals branching out from a three-phase feeder. Each single-phase lateral has some number of loads. There are three types of load connected to the circuit:

- Residential with electric heat (RH)

- Residential without electric heat (RNH)
- Commercial type loads (COM).

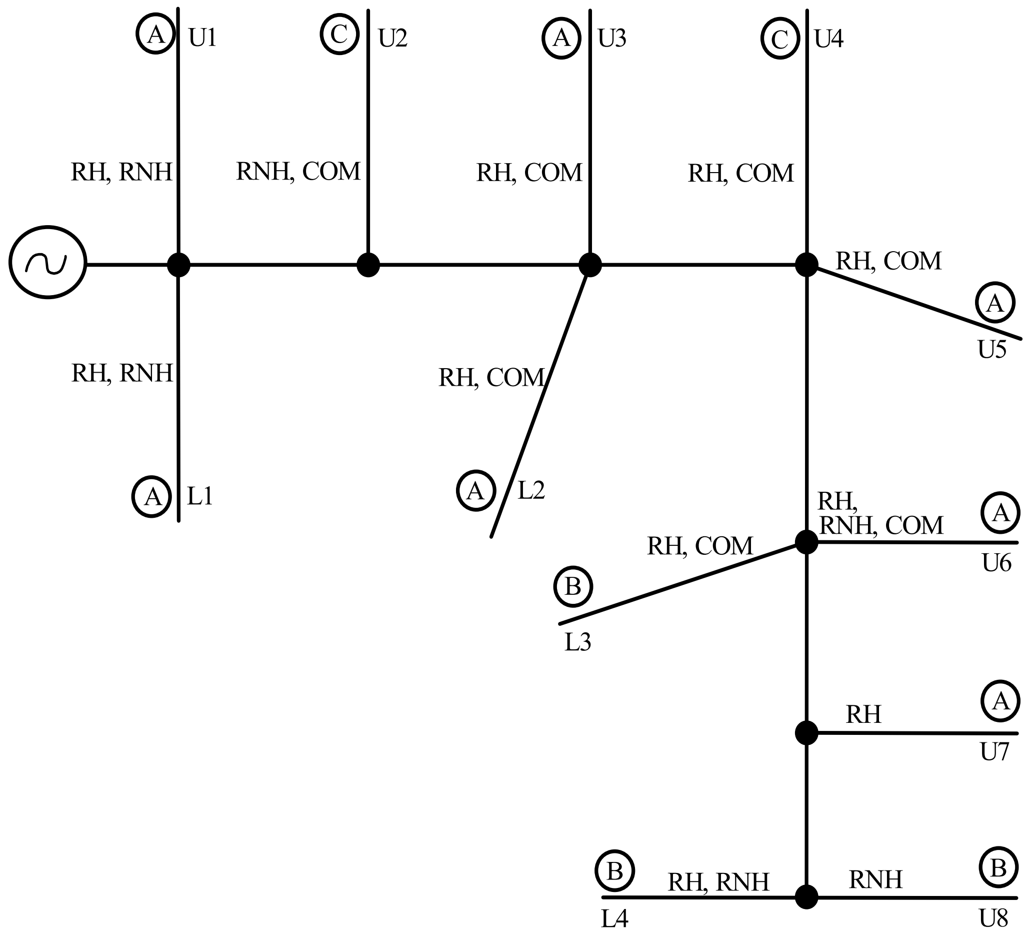


Figure 2.4 Circuit for Example 2.2

The types of loads connected are given beside every lateral in the figure. Each load type is represented by load research statistics [2.3], [2.4].

Assume that the circuit is to be balanced for a January weekday. This example will address two approaches for balancing phases for time-varying loads (i.e., 24 time points over a day for our particular case) as described below

- ***Peak-Load-Point Scenario*****:** For a given period of time, such as a January weekday, the time of the peak load is determined. The phase balancing algorithm is then run just for the time of the peak load. Hence, phases are moved to obtain the most improved circuit loss at this peak load point only. The primary concern here is to reduce the peak load as a result of minimizing the losses that occur at peak Note that the losses contribute to the peak, and reducing the losses will help to reduce the peak. However, the phase balancing calculated for just the peak may not result in the best loss solution over the entire given time period, such as a January weekday.

- ***All-Load-Points Scenario*****:** In this approach, the objective is to find phase moves that will lead to reduced circuit loss over some selected time period, such as a January weekday. Hence, as the name implies, the load for all discrete time points modeled is taken into account. When phase balancing is evaluating a phase move in this scenario, the circuit losses are summed over all load points. Hence, phase moves recommended from this scenario will attempt to reduce the circuit loss over the entire time-varying load curve.

Each load type is represented by time varying load statistics. The estimated 24-hour load curve for a typical weekday in January as observed at the substation is plotted in Figure 2.5. It is obvious from the figure that the loading is highly unbalanced. This is because the majority of the laterals have been deliberately set to phase A for the sake of demonstration purposes.

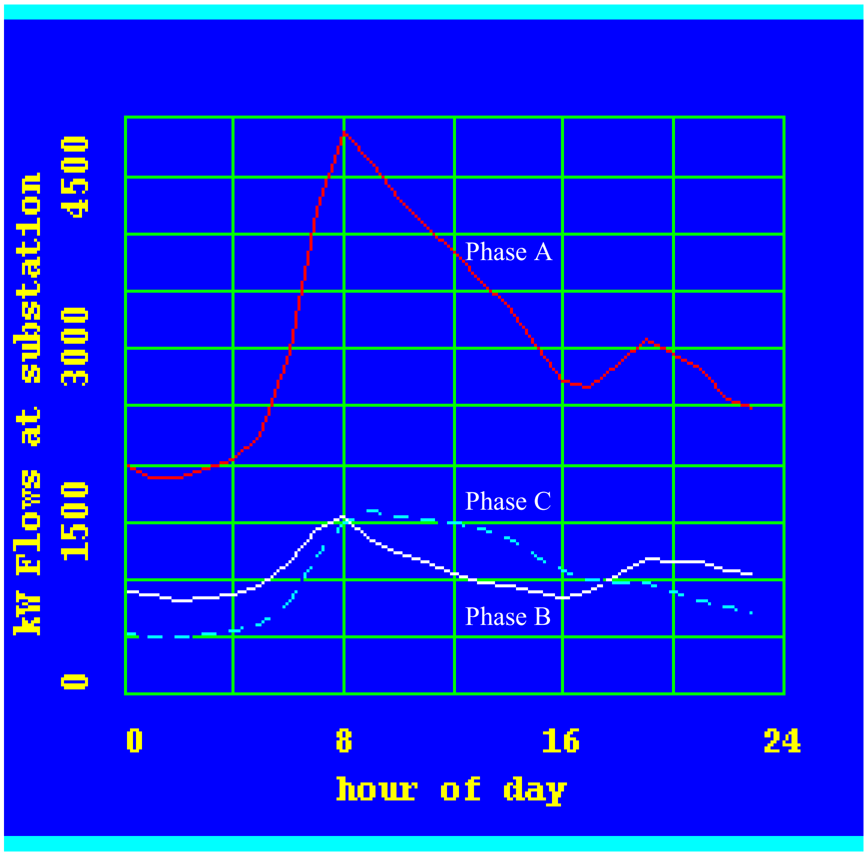


Figure 2.5 Estimated daily load curve before PB for a weekday in January.

The peak load occurs at 8:00 a.m., and the total load flowing into the circuit from the substation at this time of day is found by summing the phase loads. Also, the worst deviations observed at the substation over the 24-hour period, and the total circuit losses are recorded. Hence, before phase balancing:

$$\begin{aligned}\text{Peak load} &= \text{Phase A load} + \text{Phase B load} + \text{Phase C load}\\ &= 4949 + 1528 + 1503\\ &= 7980 \text{ kW (at 8 a.m.)}\end{aligned}$$

$$\text{circuit loss} = 480.5 \text{ kW (at 8 am)}$$

total circuit loss = 4697kWh (over 24 hours)

worst deviations :

Phase A = 91% above the average

Phase B = 46% below the average

Phase C = 45% below the average

For the peak-load-point scenario, phase moves are evaluated at the peak load time point, 8:00 a.m. In the all-load-points scenario, phase moves are evaluated taking all load points into account. Results are summarized in Table 2.3 and Table 2.4. Table 2.3 represents re-phasing recommendations of the two scenarios and Table 2.4 exhibits the results obtained as the circuit is modified according to the recommendations given by the all-load-points scenario. In Table 2.4, the term 'worst deviations' means the deviations that are recorded at the time point where the worst of 24 imbalances occur.

Table 2.3 Phase Movements Recommended by Two Scenarios

Scenario	1st Move	2nd Move	3rd Move	4th Move
Peak-Load	L2: A -› C	U3: A -› B	U2: C -› A	L1: A -› B
All-Load	L2: A -› B	L3: B -› C	L1: A -› B	-

Table 2.4 Phase Balancing Results for Example 2.2

	Worst deviations	**kW flow at peak load**	**Loss at peak load (kW)**	**total circuit loss (kWh)**
before PB	A: 91% above B: 46% below C: 45% below	A: 4949 B: 1528 C: 1503 total : 7980	480.5	4697
after PB by all-load-points scenario	A: 14% above B: 7% below C: 7% below	A: 2850 B: 2382 C: 2509 total : 7741	242.2	2619

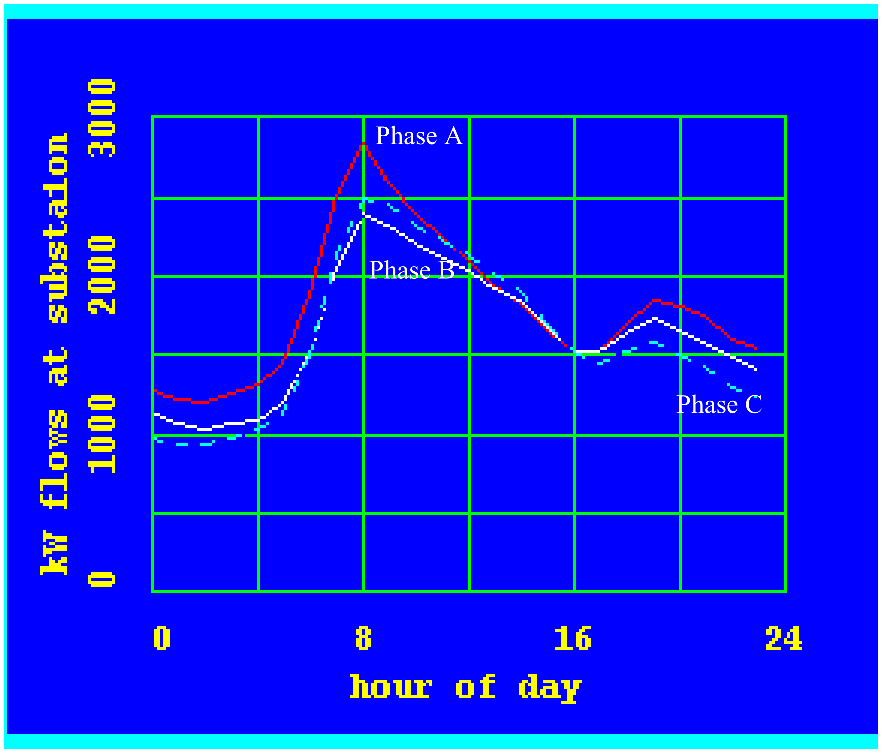


Figure 2.6 Daily load curve after PB with all-load-points scenario.

Figure 2.6 presents the daily load curves that are obtained after PB study using the all-load-points scenario. Comparing Figure 2.5 and Figure 2.6, note the improvement (Figure 2.6) in phase balance over the daily load cycle shown in Figure 2.5.

2.5.3 Example 2.3: A Large Utility Circuit

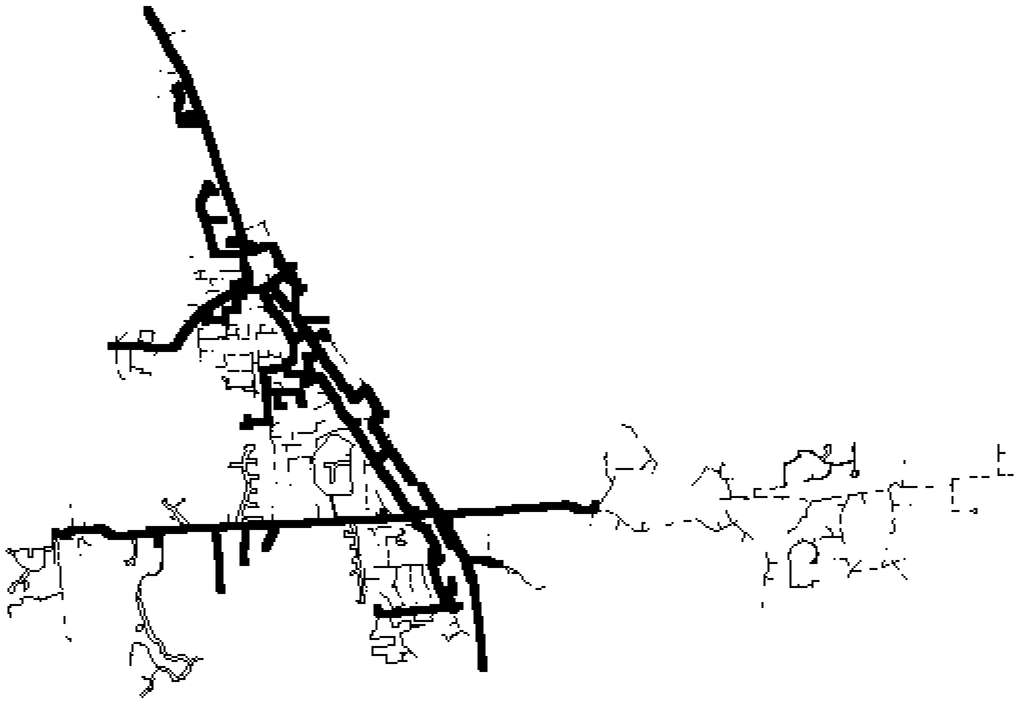


Figure 2.7 DEWorkstation model of a large circuit of Detroit Edison.

This example will consider a large utility circuit which belongs to Detroit Edison. The DEWorkstation model of the circuit is shown in Figure 2.7. The model contains 3894 components, such as line sections, capacitors, switches, etc., and 170 laterals. The circuit is highly unbalanced (phase currents at substation: I_A=428 A, I_B=123 A, I_C=226 A) because the majority of the laterals are hooked to phase A. This initial state of the circuit laterals was chosen for extreme testing of the phase balancing algorithm (the actual circuit is not operated this way).

Considering 12 a.m. for a January weekday as the load point, the imbalance at the substation before the re-phasing is 65% and the circuit has 132 kW loss. The phase balancing routine produced a design with seven phase movements. The circuit loss dropped to 115 kW. Hence, a loss saving of 17 kW is obtained which represents about a 13% loss improvement. Also, after phase balancing, new phase currents at the substation are I_A=273 A, I_B=278 A, I_C=272 A. This implies a substation imbalance of 1.3%.

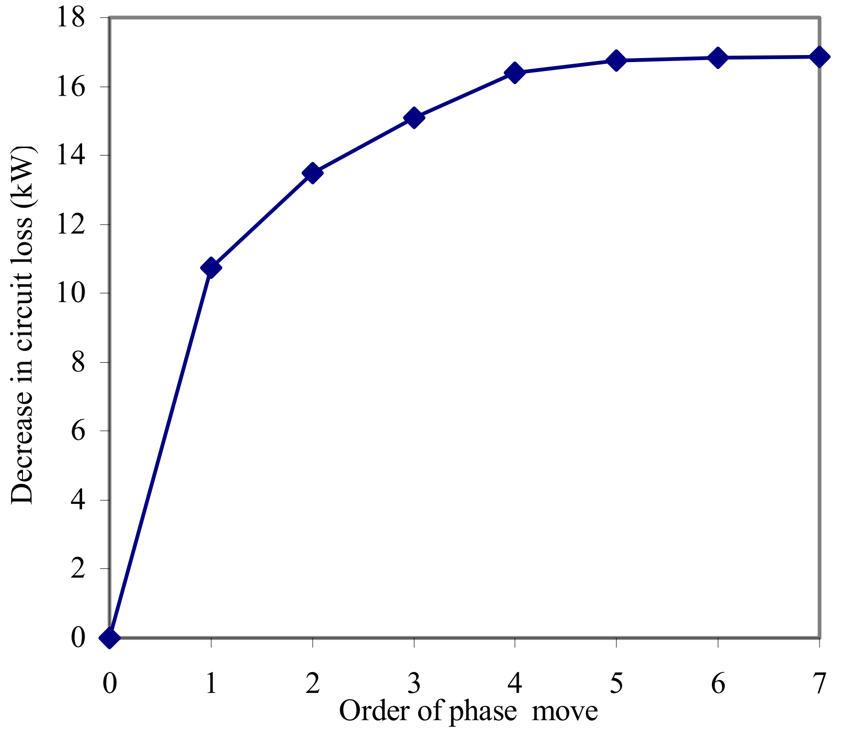


Figure 2.8 Improvement in circuit loss as phases are moved.

As mentioned in Section 2.3, the phase balancing algorithm provides a plot of loss improvement (saving) vs. phase moves. This plot is intended to help the engineer decide on the number of phase moves to perform. Figure 2.8 shows such a plot. It can be seen from Figure 2.8 that no significant loss saving is provided via those phase moves that follow the 4th phase move.

2.6 References

[2.1] R. P. Broadwater, J. Thompson, M. Ellis, H. Ng N. Singh, and D. Lloyd, "Application Programmer Interface for the EPRI Distribution Engineering Workstation," *IEEE Trans. on Power Systems*, vol. 10, no. 1, February 1995, pp. 499-505.

[2.2] R. P. Broadwater, J. Thompson, and T. E. McDermott, "Pointers and Linked List in Electric Power Distribution Circuit Analysis," *Proceedings of 1991 IEEE PICA Conference*, CH2948-8, pp. 16-21.

[2.3] Sargent, R. P. Broadwater, J. Thompson, and J. Nazarko, "Estimation of Diversity and KWHR-to-Peak-KW factors from Load Research Data," *IEEE Trans. Power Systems*, vol. .9, no. 3, , August 1995, pp. 1450-1456.

[2.4] R. P. Broadwater, A. Sergant, A. Yarali, H. Shaalan, and J. Nazarko, "Estimating Substation Peaks from Load Research Data," IEEE Trans. Power Delivery, vol. 12, no. 1, January 1995, pp. 451-456.

Chapter 3
Solving the Phase Prediction Problem

3.1 Introduction

In order to perform multi-phase power flow analysis in distribution systems, the phasing of laterals needs to be known. Laterals can be single- or two-phase, and lateral phasing is one of six values selected from: A, B, C, AB, AC, and BC. Lateral phasing information is also important to other electric utility functions, such as power restoration.

As distribution circuits have been built, many utilities have not recorded/maintained lateral phasing information. Hence, a large utility with over 2000 circuits may have over 200,000 laterals for which phasing is not known.

Using conventional, manpower-intensive techniques to determine lateral phasing requires a long term and expensive effort. This chapter addresses an algorithmic and low-cost approach to estimating the phasing of laterals in distribution circuits, referred to here as phase prediction.

The phase prediction algorithm helps the engineer assign phases to the laterals that have no phase information available. This allows distribution engineering analyses such as power flow studies to be performed on the circuits. Note that in the absence of phasing information it is almost impossible to carry out analysis applications. Later on, when time and labor constraints permit, crews can be sent to attain actual phase connections along the circuit. Predicted phases are then compared with the actual ones obtained by crew investigation and are corrected if any discrepancy occurs.

It may be noted that phase prediction includes the phase balancing problem in its solution space. Due to this, phase prediction also has value in other situations. For instance, old distribution circuits at low voltage levels, such as 4800 volts, are sometimes upgraded to higher voltage levels, such as 13200 volts. In such upgrades, the circuits are completely rebuilt. Such rebuilding provides the opportunity to improve phase balance by re-phasing laterals and loads on the circuit. Phase prediction may be used for such calculations.

The phase prediction program has been written to the Application Programmer Interface of the EPRI Distribution Engineering Workstation (DEWorkstation) [3.1]. In doing this, features of the workstation that have been exploited include automated interfaces to time-varying circuit measurements, small-customer load measurements coupled with load research statistics, and large-customer loads. Also, calculational capabilities of previously existing applications in the workstation have been utilized, including the load estimation and power flow applications.

In the subsequent sections, detailed aspects of the problem to be solved are presented. The phase prediction algorithm makes use of tabu search. A summary on the mechanics of tabu search is given. The solution process for tailoring tabu search to the phase prediction problem is described. The steps used in programming the phase prediction algorithm are listed and discussed. Simple circuit examples are used to cover points that should be understood concerning phase prediction. Results of applying the phase prediction algorithm to a test circuit are presented followed by a simple example illustrating how to use phase prediction as phase balancing.

3.2 Problem Description and Objective

The objective of phase prediction is to match circuit power flow measurements of kW and/or kVar with estimated power flow values of kW and/or kVar by assigning phasing to single- and two-phase laterals with unknown phases. Estimated power flow values are affected as phase assignments are varied. Phases that result in the minimum mismatch are recorded as *predicted* phases.

Phase prediction uses basically two types of information:

- System measurements
- Customer (load) information.

3.2.1 System Measurements

Power flow measurements at other points in the circuit may be used by phase prediction, but phase prediction requires a set of power flow measurements to exist at the start of circuit. A measurement on each of the three phases is needed. If kW and kVar values are not measured directly, then these values must be calculated from other measurements, such as voltage and current magnitude measurements, where assumptions concerning power factor may need to be made.

In order to come up with a set of lateral phasings that are representative of annual load variations, circuit power flow measurements from a number of different time points should be used. For example, time points such as summer peak, summer average load, winter peak, winter average load, and so forth should be considered.

In addition to the start-of-circuit measurements, other sets of measurements at interior points in the circuit, such as capacitor bank installations, may be provided. Such interior measurements help to increase the accuracy of phase prediction.

Consider the circuit given in Figure 3.1. Thicker lines in the figure represent the three-phase portion of the system (the spine). Laterals of single-phase and double-phase are tied to the spine at several locations along the circuit. Laterals are numbered as shown in the figure for easy referral in the discussion to follow. Except for lateral *3*, all laterals are single phase. The majority of the laterals have no phasing information available. Those unknown laterals are tagged with '?'s in the figure. Evidently, only laterals *3*, *5* and *11* have phasing information.

In the circuit there exist some three-phase locations where measurements are taken. Locations m_1, m_2 and m_3 represent interior measurement points while m_s is the start-of-circuit point. Start-of-circuit and interior measurements do not need to be captured at the same time points.

The estimated power flow values are calculated from a load estimation algorithm rather than a power flow calculation [3.1]. The power flow calculation cannot be used due to potential convergence problems. With unknown but assumed phasing, the initial loading on the circuit may be such that there is no steady state power flow solution.

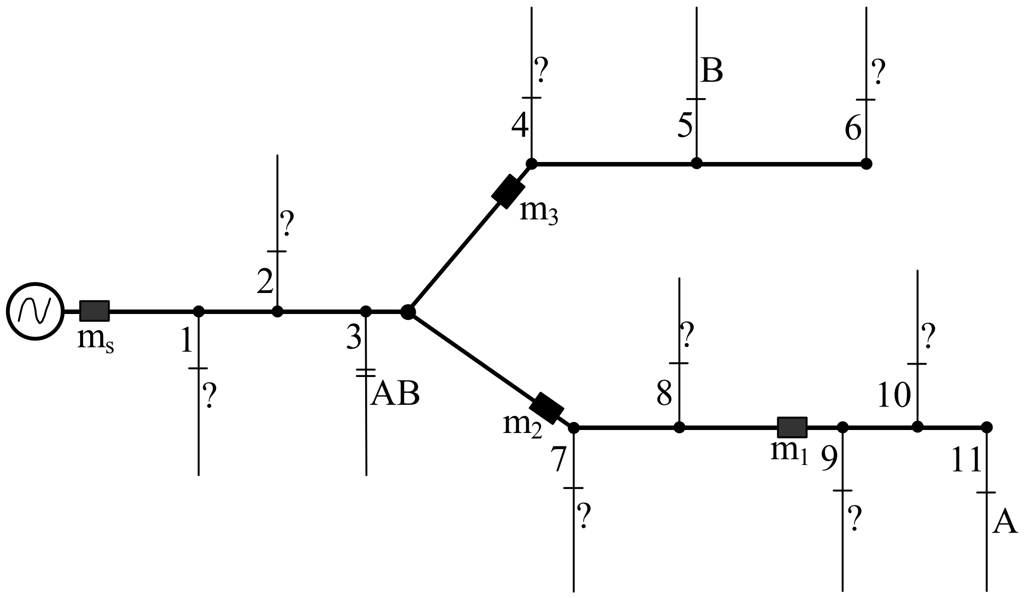


Figure 3.1 A radial distribution circuit with four measurement points and eleven laterals.

3.2.2 Customer Information

Even though the utility does not know the phasing of the lateral or of the customers connected to the lateral, the utility, via information keys between customer information systems and circuit models, often does know which lateral feeds a customer load. An information key that is commonly used is the x–y coordinate location of the distribution transformer that feeds the customer. This is the case in the study performed here.

The load estimation algorithm makes use of large-customer kW/kVar measurements and small-customer kWHr measurements coupled with load research statistics. For large customers, kW, and sometimes kVar, measurements are available on an hourly basis throughout the year. For small customers, only monthly kWHr measurements are available.

Values of customer load need to be determined which correspond in time to the circuit power flow values that are available. For large customers with hourly measurements this is no problem. For small customers, load research statistics are applied to customer kWHr to estimate kW values for a given hour, day type, and month [3.2], [3.3]. Where needed, assumed power factors based upon month and type of customer may be used to obtain kVar values.

Often, and especially for large loads, the phase of a lateral is known. The phase prediction algorithm only adjusts phasing on laterals for which the phasing is specified as being unknown.

In the discussions to follow, the term unknown lateral will be used to indicate the same meaning as a lateral that has unknown phasing.

3.2.3 Constructing an Objective Function

The load estimation algorithm calculates the total load for each phase by summing the individual transformer loads, with an estimate of the losses for each phase included. Measured power flow values may then be compared with the estimated values for the time point of interest. The objective is to find the phasing that minimizes the differences over all time points for which the measurements exist.

A measurement point *A* is referred to as *upstream* relative to a measurement point *B* if *A* is located in the feeder path going from *B* to the start of circuit.

If *A* happens to be the first measurement point that is encountered in the feeder path trace from *B*, then *A* is called the *upstream neighbor* of *B*. Correspondingly, *B* is said to be a *downstream neighbor* of *A*. In a radial system such as shown in Figure 3.1, the start of circuit is *upstream* relative to all measurement points. An interior measurement point can have only one *upstream* neighbor whereas it may have more than one *downstream neighbor*. For example, m_2 is the *upstream neighbor* of m_1, where as m_2 and m_3 are *downstream neighbors* of m_s. m_1 has no *downstream neighbor* while m_2 and m_s are *upstream* relative to m_1. Similarly, since the first *upstream* measurement point that will be encountered on the path going from m_1 to the start of circuit is m_2, m_2 is said to be the *upstream neighbor* of m_1. *Downstream*, *upstream*, *downstream neighbor* and *upstream neighbor* entities corresponding to each measurement point of Figure 3.1 are presented in Table 3.1.

Table 3.1 Relations Among the Measurement Points in Figure 3.1.

Measurement Point	***Downstream* Measurement Points**	***Upstream* Measurement Points**	***Downstream* Neighbors**	***Upstream* Neighbor**
m_s	m_3, m_2, m_1	--	m_3, m_2	--
m_1	--	m_2, m_s	--	m_2
m_2	m_1	m_s	m_1	m_s
m_3	--	m_s	--	m_s

Phase prediction starts prediction at outer measurement points and moves toward the start of circuit. That is, measurement matching is performed in a sorted order in that no measurement point should be processed before any of the measurement points that are *downstream* relative to the measurement point that is being processed.

Based on the requirement that a measurement point not be processed before any of its *downstream* measurement points a processing order such as m_1–m_3–m_2–m_s is a valid sequence. On the other hand, the order m_2–m_3–m_1–m_s will violate the rule because m_1, which is *downstream* relative to m_2, must be processed before m_2. Fortunately, traces that DEWorkstation uses to represent the circuit connectivity enable phase prediction to perform a sorted order [3.4].

Lateral points that are located between a measurement point and any of its *downstream neighbors*, if there are any, are named as *associated* laterals. For instance consider m_s. The set of its *associated* laterals consists of those which are located between it and m_2 or between it and m_3. From the figure it is clear that laterals *1*, *2* and *3* can be associated with m_s. All measurement points and their *associated* laterals are tabulated in Table 3.2.

Table 3.2 Laterals Associated with Measurement Points.

Measurement Point	*Associated* Laterals
m_s	1, 2, 3
m_1	9, 10, 11
m_2	7, 8
m_3	4, 5, 6

Measurements taken at a measurement point are used to predict the phasing of its *associated* unknown laterals

When processing a measurement point, the measured and calculated power flows should be *adjusted* so that they can be used properly. Load estimation power flow values that are calculated at *associated* known laterals and *downstream neighbors* should be subtracted from the both measured and calculated values of the measurement point. The remaining flows are referred to as *adjusted* flows. This adjustment is to justify the fact that flows that remain after the adjustments are due to the contributions made only by unknown laterals *associated* with the measurement point being processed.

Consider that m_s in the figure is to be processed. That is, phases of unknown laterals *associated* with m_s will be predicted via using the measurements available at m_s. In order for this process to be evaluated, it is necessary that all *downstream* neighbors of m_s have already been processed. For the sake of argument, we assume that m_3 and m_2 have already been processed —which means that the phases of laterals *4*, *6*, *7*, *8*, *9* and *10* have been predicted. Note that the processing of m_1 is prerequisite to the processing of m_2.

Now we are ready to process m_s. m_s has some kW measurements recorded at some time point, say t_s. Let these measurement values be M_A, M_B and M_C for phase A, B and C kW flows respectively. The phases on laterals *1* and *2* will be predicted through the processing of m_s. Initially, some random phases are assigned to these laterals (say both have been set to phase A). Then, a load estimation run at the specific time point t_s will result in kW flows as shown in Figure 3.2. Here, we are not interested in what is going on beyond the *downstream neighbors* m_3 and m_2. The flows at m_s obtained by the load estimation are due to the flows at laterals *1*, *2* and *3* and the flows at measurement points m_2 and m_3. In order to be able to evaluate the affect of flows of unknown laterals *1* and *2* on the flows of m_s, the flows at the locations with known phases will be subtracted from the measured and the calculated ('calculated' means 'obtained by the load estimation run') values. That is,

known A flow = A flow at *3* + A flow at m_2 + A flow at m_3
= 4 + 8 + 10
= 22 kW

known B flow = B flow at *3* + B flow at m_2 + B flow at m_3
= 5 + 7 + 9
= 21 kW

known C flow = C flow at *3* + C flow at m_2 + C flow at m_3
= 0 + 6 + 8
= 14 kW

adjusted calculated A flow = calculated A flow – known A flow
= 29 – 22 = 7 kW.

adjusted measured A flow = measured A flow – known A flow
= 22 – 22 = 0

adjusted calculated B flow = calculated B flow – known B flow
= 21 – 21 = 0

adjusted measured B flow = measured B flow – known B flow
= 24 – 21 = 3 kW

adjusted calculated C flow = calculated C flow – known C flow
= 14 – 14 = 0

adjusted measured C flow = measured C flow – known C flow
= 18 – 14 = 4

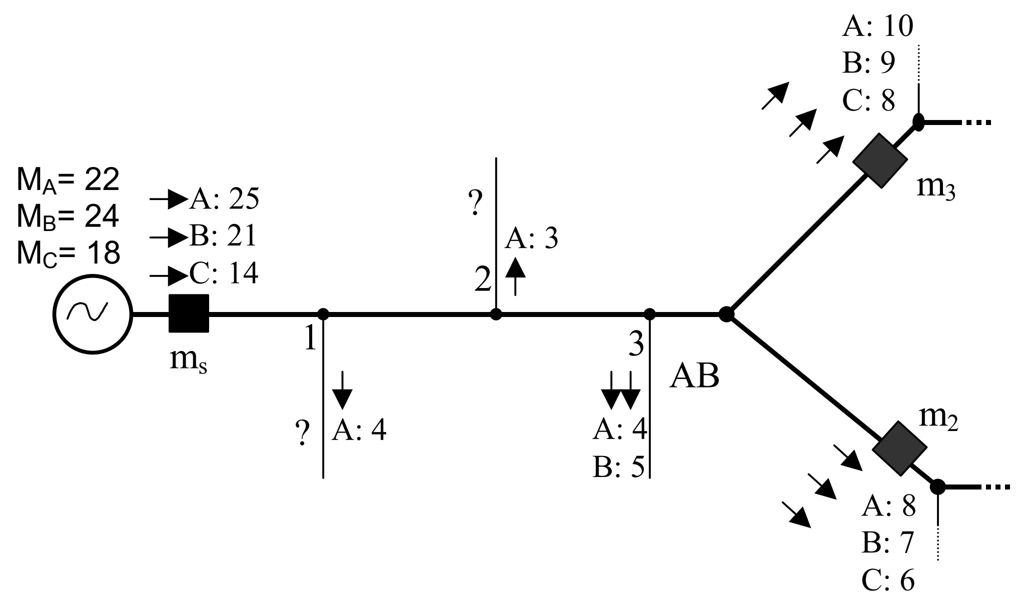


Figure 3.2 Flows (in kW) obtained when processing m_s.

After the adjustments, it is seen that calculated phase A flow is higher than phase A measurement by 7 kWs. However, phase B flow is less than phase B measurement by 3 kWs. Similarly, there is a 4 kW mismatch between the calculated phase C flow and measured phase C flow. Therefore there exists a total mismatch of 14 (7 + 3 + 4) kWs.

It is desired to have calculated flows matching measured flows for each phase. In the particular case here, we are free to change phases on laterals *1* and *2* so that we can obtain improved mismatches due to the new flows which will result from varied phasings of these laterals. Note that lateral *1* and lateral *2* together can take nine different phasing alternatives. Intuitively, in the simple case above selecting the phase of lateral *1* as being phase C and that of lateral *2* as being phase B will give the best mismatch over all other eight alternatives. It is easy to see that changing the phase of lateral *1* from

A to C will reduce *adjusted* calculated A flow at m_s by 4 kWs while increasing *adjusted* calculated C flow at m_s by 4 kWs. Similarly, changing the phase of lateral *2* from A to B will reduce *adjusted* calculated A flow by another 3 kWs while increasing *adjusted* calculated B flow by 4 kWs. Hence,

phase A mismatch	= *adjusted* calculated A flow	–	*adjusted* measured A flow		
	=	(7 – 3 – 4)	–	0	= 0
phase B mismatch	= *adjusted* calculated B flow	–	*adjusted* measured B flow		
	=	(0 + 3)	–	3	= 0
phase C mismatch	= *adjusted* calculated C flow	–	*adjusted* measured C flow		
	=	(0 + 4)	–	4	= 0

total mismatch = phase A mismatch + phase A mismatch + phase A mismatch
= 0

Therefore, the objective, when predicting the phases of unknown laterals *associated* with a given measurement point, is to minimize the mismatches observed at the every phase of that measurement point. Mismatches are defined in absolute terms. That is, having a calculated flow that is higher than the measured flow by say 5 kWs, is regarded the same as having a calculated flow that is less than the measured flow by 5kWs. In case measurements are taken at more than one time point, a separate load estimation run for each time point will be needed. Then, mismatches are evaluated in absolute terms and for each load estimation run. If there are two measurement types to be considered, then mismatches are calculated separately, again in absolute terms, for each flow type of measurement.

Based on the discussion so far, the objective function to be minimized when processing a measurement point M can be formulated as follows:

$$\min \sum_{i\in\{A,\,B,\,C\}} \left(\sum_{j=1}^{nP} \left| mP_{ij} - cP_{ij} \right| + \sum_{j=1}^{nQ} \left| mQ_{ij} - cQ_{ij} \right| \right)$$

where:

nP	: number of three-phase kW measurements available at M
nQ	: number of three-phase kVar measurements available at M
mP_{ij}	: *adjusted* phase i kW flow measurement at time point j
cP_{ij}	: *adjusted* phase i kW flow calculation (estimation) at time point j
mQ_{ij}	: *adjusted* phase i kVar flow measurement at time point j
cQ_{ij}	: *adjusted* phase i kVar flow calculation (estimation) at time point j

Here, a three-phase measurement consists of three single-phase measurements namely phase A, phase B and phase C measurements that are taken at a particular time point at some particular three-phase measurement location (point).

Note that for a single-phase lateral there are three possible values of the phasing which are A, B, or C. For a circuit that has 100 single-phase laterals, an exhaustive search would require 3^{100} evaluations. Thus, an exhaustive search is not practical. After investigating several different optimization approaches [3.8] it was decided to employ the tabu search (TS) method as the optimization procedure to solve the combinatorial optimization problem of phase prediction. The fundamental mechanics of a TS procedure, and the discussion of how phase prediction makes use of TS will be presented in subsequent sections.

3.2.4 Biasing the Measured Values

It is not unusual that measured values may not be in accord with the calculated values at a measurement point of a distribution circuit with known laterals, no matter how well the circuit, including its customers, is modeled. This is normal because the measurements by nature are random variables. In other words, for example, two kW measurements taken at the same location of a given circuit for two time points of the same type may differ from each other.

Second, calculated flows too are evaluated based mainly on the statistical data that have been gathered via performing some research on the customers in regard to the type, number, daily-usage pattern, etc. of their loads. Hence, even if measurements are perfect, it is still highly probable that calculated flows will not match the measured flows.

In the phase prediction problem, where the main job is to determine a phasing pattern that will match the calculated flows to the measured flows, it is quite important to have measured flows not diverging too much from the calculated flows. There might be situations where the phase prediction process cannot work consistently because of the measured flows being distant from the calculated ones.

Consider the circuit given in Figure 3.3. There exists a three-phase kW measurement taken at some time point at the start of circuit. kW flows as calculated by load estimation, considering the particular time point, are shown along the laterals. Evidently, since each phase at the start of circuit has a certain amount of measured flow there must exist at least three single-phase laterals each being connected to different phases from one another.

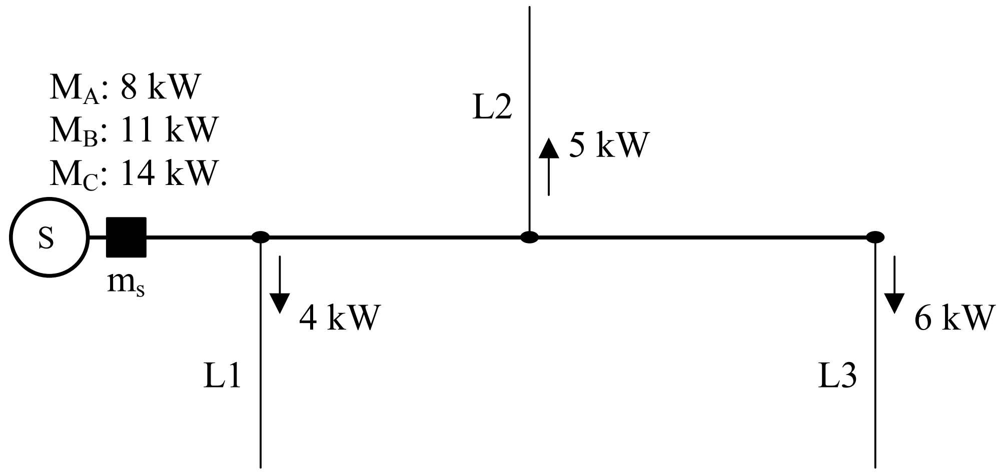


Figure 3.3 A simple circuit with three unknown single-phase laterals.

Let us examine the objective values for the following alternatives. Each alternative is designated by a three-letter set where the first letter represents the phase for L1, the second letter represents the phase for L2, and the last letter indicates the phase for L3.

ABC: objective value = | 8 – 4 | + | 11 – 5 | + | 14 – 6 | = 18

ACB: objective value = | 8 – 4 | + | 11 – 6 | + | 14 – 5 | = 18

BAC: objective value = | 8 – 5 | + | 11 – 4 | + | 14 – 6 | = 18

BCA: objective value = | 8 – 6 | + | 11 – 4 | + | 14 – 5 | = 18

CAB: objective value = | 8 – 6 | + | 11 – 4 | + | 14 – 5 | = 18

CBA: objective value = | 8 – 6 | + | 11 – 5 | + | 14 – 4 | = 18

Apparently, the process is unable to measure the relative attractiveness of the listed alternatives. The big divergence between the measured and the calculated flows has caused the objective function to be indifferent to the six phasing modifications.

In the simple case here, it is easy to make a guess on the phasing values of laterals. Since the phase C measurement is the largest one of all the three measurements, L3 which has the largest flow among the three laterals is expected to be phase C. Likewise, L1 is likely to be phase A because it has the smallest flow similar to the phase A measurement being the smallest of all three measurements. Finally, L2 remains to be phase B. Note that these calculated flows are estimations. Hence, there is still a chance of, for example, L2 having an actual flow of 6 kW and of L3 having an actual flow of 5kW. Nevertheless, we will depend on the results coming from the load estimation, and do not complicate the analysis further by considering the other possibilities.

In general, it may be assumed that the distances among the calculated flows are relatively similar to the distances among the measured flows. Then one might want to bias the measured values so that these biased flows become more comparable to the calculated flows than the unbiased measurements. An easy way to do this is to find an off-set to be subtracted from the measured values. For every type of flow (i.e., kW and kVar) a corresponding off-set value is calculated and the flows that are calculated/measured over all time points are involved in this calculation. Considering the calculated/measured flows at a given measurement point M, the biasing process is formulated as follows.

$$P_{\text{off-set}} = \frac{\sum\limits_{i \in \{A,\,B,\,C\}} \left(\sum\limits_{j=1}^{nP} \left(P_{ij}^{m} - P_{ij}^{c} \right) \right)}{3 * nP}$$

$$Q_{\text{off-set}} = \frac{\sum\limits_{i \in \{A,\,B,\,C\}} \left(\sum\limits_{j=1}^{nQ} \left(Q_{ij}^{m} - Q_{ij}^{c} \right) \right)}{3 * nQ}$$

$$P_{ij}^{m_{\text{biased}}} = P_{ij}^{m} - P_{\text{off-set}}$$

$$Q_{ij}^{m_{\text{biased}}} = Q_{ij}^{m} - Q_{\text{off-set}}$$

where:

nP	: number of three-phase kW measurements available at M
nQ	: number of three-phase kVar measurements available at M
$P_{\text{off-set}}$	: off-set value for biasing kW measurements available at M
$Q_{\text{off-set}}$	: off-set value for biasing kVar measurements available at M
P_{ij}^{m}	: phase i kW flow measured at time point j
P_{ij}^{c}	: phase i kW flow calculated at time point j
Q_{ij}^{m}	: phase i kVar flow measured at time point j
Q_{ij}^{c}	: phase i kVar flow calculated at time point j
$P_{ij}^{m_{\text{biased}}}$	: biased value of P_{ij}^{m}
$Q_{ij}^{m_{\text{biased}}}$	: biased value of Q_{ij}^{m}

With these equations, it is easy to bias (or compensate) the measurements of the simple circuit given in Figure 3.3. Since there are only kW measurements to be considered one will need to evaluate an off-set value for only kW flows. Note that off-set for kW measurements is nothing but the difference between the sum of all measured kW flows and the sum of all calculated kW flows such that this difference is then scaled by the total number of measured kW flows. Because each three-phase kW measurement consists of three single-phase flows, the total number of measured kW flows will be the number of three-phase measurements as multiplied by three. In Figure 3.3 there is only one three-phase measurement. Consider some interim phasing pattern say CBA (i.e., L1 is phase C, L2 is phase B and L3 is phase A), then,

$$P_{\text{off-set}} = \frac{\left(P_A^m + P_B^m + P_C^m\right) - \left(P_A^c + P_B^c + P_C^c\right)}{3*1}$$

$$P_{\text{off-set}} = \frac{(8+11+14)-(6+5+4)}{3*1} = 6$$

$$P_A^{m_{\text{biased}}} = 8 - 6 = 2 \text{ kW}$$

$$P_B^{m_{\text{biased}}} = 11 - 6 = 5 \text{ kW}$$

$$P_C^{m_{\text{biased}}} = 14 - 6 = 8 \text{ kW}$$

One should note that biasing the measured values is a pre-process. That is, the measured flows have to be biased prior to the evaluation of mismatches considered for the objective function. Hence, the measured flows that we have considered when evaluating *adjusted* measured flows in Section 3.3.3 are actually the biased measured flows as calculated based on the discussion given in this section.

3.3 Tabu Search: An Overview

Tabu search (TS), developed by Glover [3.5], is a heuristic algorithm for solving combinatorial optimization problems. After having experienced a very slow growth in popularity at its early stages, it has recently been applied to a variety of problems, and is increasingly finding new application areas [3.6], [3.7].

To describe TS, let us consider a combinatorial optimization problem P defined as:

$$\text{P:} \qquad \text{minimize } c(x): \; x \in X, \; X \subseteq R_n$$

X is defined to be the set of all feasible solutions. Any feasible solution x $\in$X is referred to as a *trial solution*. Associated with each *trial solution* is a numerical value that may be considered the ‘cost’ of the solution and is determined from the ‘cost function’ (i.e., objective function which may be linear or nonlinear). An operation that transforms one *trial solution* into another is referred to as a *move*. In many cases, the evaluation of a *move* is based on the change in the cost function. For each *trial solution* x_i, there exists a subset of X, say X_i, consisting of *trial solutions* that may be reached from x_i in one *move*. X_i is considered to represent the *neighborhood* of x_i.

TS is an iterative procedure that starts from some initial feasible solution x_0 and attempts to evaluate a better solution x_1 in a manner of a greatest-decent algorithm. x_0 is regarded as the current solution at iteration 1. Similarly, x_1 is considered to be the current solution at iteration 2. That is, TS starts from x_0 (iteration 1), selects the best solution x_1 in X_0 then moves from iteration 1 to iteration 2 with x_1 being the current solution at iteration 2. A *neighborhood* X_1 is defined on x_1, and the next *move*, which leads to x_2, is chosen from the set X_1. The process is repeated until one of stopping criteria, such as reaching the maximum allowed iteration, is satisfied. Any solutions that yield lower cost values than those previously encountered are recorded. This enables TS to report the best solution x^* encountered throughout the process (i.e., the solution of P) after the search is complete [3.6], [3.8].

TS provides a mechanism to constrain the method from re-visiting a previous solution except by following a trajectory not traveled before. This is accomplished by introducing *tabu restrictions*. *Tabu restrictions* are conditions that make a *move* forbidden (*tabu*) if the *move* has any of these conditions. The restrictions are recorded in a *tabu list*, where they reside for a specified number of iterations and then are removed, freeing them from their *tabu* status [3.9], [3.10]. The number of iterations for which a restriction will remain *tabu* is referred to as the *tabu tenure* of the restriction [3.8]. *Tabu restrictions* enable the search to overcome cycling and the trap of local optimality which many optimization algorithms fail to manage.

Tabu restrictions may harm the process in the sense that they may also forbid *moves* leading to unvisited solutions, and in particular to unvisited solutions that may be attractive. It is therefore necessary to overrule the *tabu* status of *moves* in certain situations. This is performed by means of *aspiration criteria*, which represent conditions that allow a normally forbidden *move* to be made. For example, a *tabu move* may be attractive because it is leading to a new solution that is better than the best found thus far. In such a case one would not want to lose the *move*. Therefore, the *tabu* status of the *move* is released and the *move* is accepted for consideration [3.11].

We have described basic components of TS in this section. Elaboration of the algorithm can be found in the literature. [3.12] presents a set of papers dealing with technical aspects of TS together with a collection of applications illustrating its principles.

The next section discusses the adaptation of the TS approach to solving the phase prediction problem in radial distribution systems. This adaptation basically includes modeling the combinatorial process along with the characterization of *moves*, defining *tabu restrictions* and *aspiration creiteria*, and determining a *tabu tenure* value to be applied to *tabu restrictions*.

3.4 Phase Prediction Algorithm

3.4.1 Solution String

Consider the circuit shown in Figure 3.4, where there exists a set of measurements to be matched and four laterals. One lateral has known phase value while others have no available phasing information. The task will be to predict phases on these three unknown laterals (L1, L2 and L3) by an iterative process. Laterals L2 and L3 in Figure 3.4 are single phase in contrast to L1 being a double-phase lateral. The presence of a two-phase lateral adds an additional constraint to the problem in that both branches of the two-phase lateral cannot take on the same phase. Here, the term 'branch' is meant to be the part of a lateral that is associated with any individual phase of the lateral.

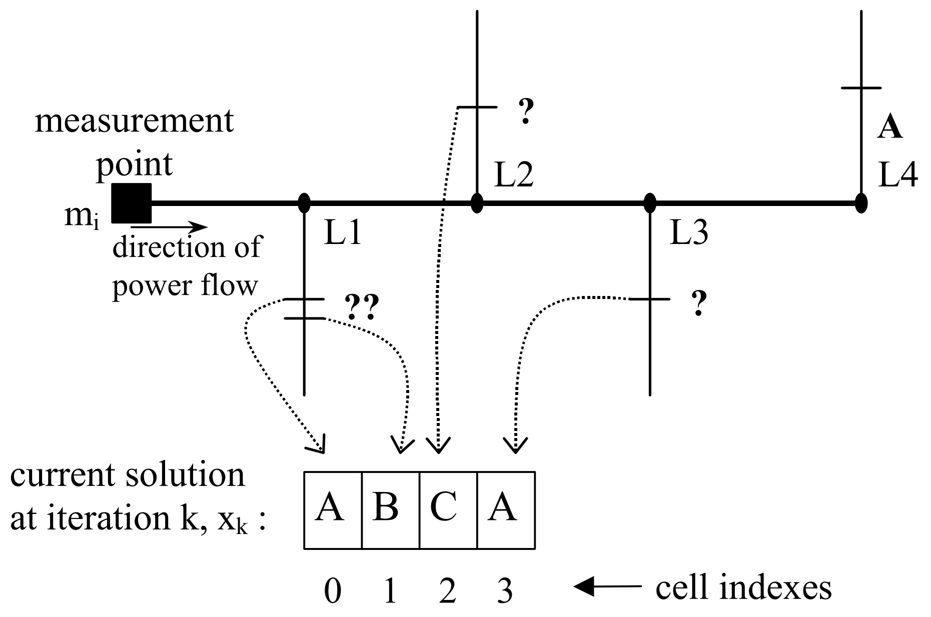


Figure 3.4 A measurement point with four *associated* laterals.

Phases of unknown laterals *associated* with the measurement point are represented in the solution string of the TS algorithm as shown in the figure. With the TS that we consider here the concept of a *cell* is used. Figure 3.4 illustrates four *cells* that may be linked to the branches at L1, L2 and L3. Each *cell* holds the phasing value for its associated branch. *Cells* indexed by 0 and 1 correspond to the branches of L1. Similarly, single-phase (hence single-branch) laterals L2 and L3 are represented by *cells* 2 and 3 respectively. The number of unknown phases determines the dimension of the problem, *n*. If two branches belong to the same lateral, then these branches are called *companions*. Consequently, *cells* representing *companion* branches are also referred to as *companions*. For example, *cells* 0 and 1 in Figure 3.4 are *companions* of each other. Note that *companion cells* are not allowed to take the same phase values.

Also depicted in Figure 3.4 is a solution string of size four obtained at some iteration k, x_k. Here, phase values are indicated in letters to make the illustration easier. In fact, numbers are used to represent the phase values in the programming of the algorithm. It may be noted that x_k does not necessarily have to be the solution as of iteration k. That is, it may or may not provide the best value of the objective function that has been obtained over all iterations performed thus far. The solution as of iteration k is the one that has lead to the best result among the solutions obtained at the iterations until k. The algorithm uses a long-term memory for keeping the track of the best solution encountered during the entire course of the process.

***3.4.2 Description of* Moves**

The TS considered in our study implements two types of *moves* so as to generate alternatives around a current solution:

- *Simple moves*
- *Exchange moves*

Note that solutions that are reached by a single *simple* or *exchange move* from some current solution x_k constitute the *neighborhood* of x_k, X_k.

With *simple moves*, the phasing in a single *cell* is varied. For a *cell* associated with a single-phase lateral, the phasing can take on two other values. For a *cell* associated with a two-phase lateral, the phasing can only take on one additional value.

Exchange moves involve two *cells*. With *exchange moves* the phase values that are contained in two *cells* are swapped.

At a given iteration in TS, all *simple* and *exchange moves* are evaluated, and the *move* that results in the best value of the objective function is chosen as 'the *move* to implement for the iteration'. At the next iteration the process is repeated. A *move* is always made, even if that move does not result in the smallest value of the objective function seen thus far.

A list of all *simple* and *exchange moves* that one can perform around the current solution x_k of Figure 3.4 is presented in Table 3.3. The list consists of 14 *moves* such that the first eight *moves* are of *simple* type while the remaining six are of *exchange* type. In the table, some phase values are underlined to point out the *cell(s)* involved in the *move* that is being considered. For instance, consider the row associated with *Move 10*, where the phase values related to *Cell 0* and *Cell 2* are underlined. This means that *Move 10* is an *exchange move* performed by swapping the phase values contained in *Cell 0* and *Cell 2* of x_k.

A number of observations may be based on Table 3.3. First, two *simple moves* can be performed in regard to each *cell* of the problem at some given iteration. For instance, *Move 5* and *Move 6* are performed by changing the phase of *Cell 2* from C to A and from C to B respectively. Hence there exist *2*n simple moves* at any iteration of a problem of size *n*. However, not every *simple move* can be allowed for consideration if *companion cells* exist. For example, consider *Move 1* which represents a *simple move* performed by changing the phase value contained in *Cell 0* from A to B. In other words, *Move 1* will make the two *companion cells*, *Cell 0* and *Cell 1*, have the same phase. This *move* is not allowed because it will cause both branches of L1 to be connected to phase B of the spine.

Table 3.3 Trial Solutions that can be Obtained by all *Simple* or *Exchange* Moves from x_k (i.e., X_k). Legend for the 'Comment' column is as follows. 'X': not allowed, '√': allowed, '!': nothing will change.

Move No.	String of Trial Solution				Comment	*Move* Type
	Cell 0	*Cell 1*	*Cell 2*	*Cell 3*		
1	B	B	C	A	X	
2	C	B	C	A	√	
3	A	C	C	A	√	
4	A	A	C	A	X	*simple*
5	A	B	A	A	√	
6	A	B	B	A	√	
7	A	B	C	B	√	
8	A	B	C	C	√	
9	B	A	C	A	√	
10	C	B	A	A	√	
11	A	B	C	A	!	*exchange*
12	A	C	B	A	√	
13	A	A	C	B	X	
14	A	B	A	C	√	

In regard to *exchange moves* one can say that every *cell* participates in *exchange moves* with the remaining (n – 1) *cells*. Furthermore, considering an *exchange move* that *Cell i* participates in with *Cell j* is nothing but an *exchange move* that *Cell j* participates in with *Cell i*, one can calculate the total number of *exchange moves* as $n(n – 1)/2$. However, if *companion cells* exist, then not every *exchange move* can be allowed because of the same reasoning explained for the *simple move* case. For example *Move 13* is not allowed for consideration because it will result in *Cell 1* having the same phase as that of *Cell 0*. Also, it may be easily noted that a *move* should be disregarded when the participating *cells* both have the same phase value. Such a *move* is *Move 11* which will provide nothing new for the problem at iteration k.

In this study, a *move* (and the *trial solution* obtained by that *move*) is considered as feasible as long as it does not cause two *companion cells* to take the same phase values, and as long as it does not involve, if it is an exchange move, two cells with the same phase values.

3.4.3 Defining Restriction, Aspiration, and Tabu Tenures

When a *move* is selected to implement for iteration, the *cell(s)* affected by the *move* are placed under a *tabu tenure* limit. That is, the phase value contained in the *cell* cannot be changed back to the value that it previously was for a specified number of iterations, referred to as the *tabu tenure*.

For illustration, assume that a *simple move* has changed the phase value contained in a *cell* from A to B. Also assume that a *tabu tenure* value, say 7, has been defined. Then, phase value A will have a *tabu status* for the duration of the following 7 iterations. This means that any *move* that results in a phase value A on the *cell* is not allowed for consideration over the next 7 iterations. To this end, a *move* is referred to as a *tabu move* if any of the new phase values contained in the affected *cell(s)* has *tabu tenure* value greater than zero.

Therefore, associated with every *cell* is a data structure to implement a short-term memory of recent changes made on the *cell*. A *cell* can take phase values of A, B and C. Each phase value has a *tabu tenure* associated with it.

At the beginning of the TS procedure, all *tabu tenures* are preset to some negative values to indicate no *tabu status* for these phase values. When a *tabu restriction* is marked on a phase, its *tabu tenure* is set to some predetermined value *nTenure*. As the process goes from one iteration to the next the *tabu tenure* associated with the phase value is decremented. Eventually, the *tabu tenure* values become zero (and negative in the subsequent iterations) which means that any *move* that leads to this phase are now allowed. Every time when a *move* selected to implement changes the phase value of a *cell* to something else the *tabu tenure* associated with this particular phase is reset back to *nTenure*.

Nevertheless, one can accept the *tabu move* for consideration without the need for the *tabu tenure* value to become zero (or negative) provided that the *move* satisfies the *aspiration criterion*. The *aspiration criterion* for our study is established as follows: Accept a *tabu move* if it leads to a solution better than the best solution found thus far.

Another aspect of using the TS for solving the phase prediction problem (or any other problem in general) is to identify a robust *tabu tenure* size, *nTenure*. When *nTenure* is too small cycling may occur. Similarly, using too large *nTenure* may lead to deterioration in solution quality which will be caused by forbidding too many moves. Based on our work, the following expression is appropriate for determining sound *tabu tenure* sizes.

$$nTenure = n_1 + 1.2\sqrt{n_2}$$

where:

n_1 : number of *cells* that have total flows less than:

$$10\% \left(\frac{\text{total of } \textit{adjusted} \text{ calculated flows at M}}{n} \right)$$

n_2 : $n - \mathrm{n}_1$

n : dimension of the problem

M : measurement point under consideration

TS does not recognize optimal cost values. That is, it continues the search forever unless a stopping condition is met. Therefore, a stopping condition must be defined. In our study, a limit on the number of iterations, *nMax*, is used as the stopping condition. In the majority of cases studied thus far, optimal solutions are achieved within the first 100 iterations. Nevertheless, since the algorithm is very fast, a value of 500 for *nMax* has been used.

3.4.4 Finding an Initial Solution

An initial solution should be available for TS to be able start. As the process progresses from one iteration to another the current solutions, one of which is the initial solution that corresponds to the very first iteration, get updated with the selected solutions. Any feasible solution can be taken as an initial solution. For instance, one can chose phase A for every single-phase lateral (i.e., every *cell* associated with a single-phase lateral) and phase AB for every double-phase lateral (i.e., every pair of *companion cells* associated with the branches of a double-phase lateral).

Nevertheless, rather than assigning random or pre-specified phasing values to unknown laterals of the problem, we will follow the following procedure for the determination of an initial solution.

Step 1: For every phase of the measurement point under consideration, calculate the total flow by summing up all the *adjusted* measured values associated with the phase. If both kW and kVar flows are to be considered, then the summation should include the *adjusted* measured kW and kVar flows of the corresponding phase of the measurement point.

Step 2: For every *cell*, calculate the total flow by summing up all the flows that are calculated at the branch associated with the *cell*. Again, if both kW and kVar measurements are to be matched, then all kW and kVar flows calculated at the corresponding branch should be included in the summation.

Step 3: Based on total flows calculated in Step 2, list *cells* in descending order. Go to the first *cell* in the list. Let this *cell* be the current *cell*.

Step 4: At the measurement point, specify the phase that has the biggest total flow. Let this be phase i. Initialize the phase of the current *cell* as being phase i. If this current *cell* has a companion *cell* that has already been initialized as being phase i, then select the phase with the second biggest total flow. Let this phase be phase j. Initialize the phase of the current *cell* as being phase j.

Step 5: Update the total flows at the measurement point by subtracting the total flow of the current *cell* from the total flow associated with the selected phase (either phase i or phase j).

Step 6: Consider the next *cell* down in the list. Let this be the current *cell*.

Step 7: Repeat steps 4, 5 and 6 until all the *cells* in the list are initialized.

3.4.5 A Summary of the Algorithm

Based on the descriptions made so far, the following summarizes the TS procedure used in this work.

```
1- Find an initial $x_0$, and let $x^*=x_0$. Set the iteration counter k=0. Set tabu
   tenures to negative values

2- Evaluate (i.e., calculate c(x)) every feasible $x \in X_k$
       i- check tabu status
              if x is tabu goto ii
              else accept x
       ii- check aspiration
              if $c(x)<c(x^*)$ accept x
              else do not accept x

3- Choose the best $x_j$ among accepted $x \in X_k$
       - update $x^*=x_j$ if $c(x_j)<c(x^*)$

4- Update tabu restrictions
       - decrement tabu tenures for all phases of all cells
       - set tabu tenures to nTenure for phase(s) affected by the
         transformation $x_k \rightarrow x_j$

5- k=k+1
       - set the current solution $x_k=x_j$, and goto 2 if k<nMax

6- Stop and report $x^*$ to be the solution of the problem.
```

The phase prediction algorithm actually consists of two stages. In Stage 1 a TS is associated with each circuit location where there exists measured values to be matched. The start of circuit must be one of the measured value locations. In Stage 1 phase prediction is performed to match interior circuit measurements prior to matching the start-of-circuit measurements. The processing of measurement points (hence TS) is executed in a sorted order as defined in Section 3.2.3. Each TS considers only predicting the phases of the *associated* unknown laterals.

In Stage 2 a TS is performed again, but this time only for the start-of-circuit measurement. The solutions that are obtained during the processing of measurement points in Stage 1 are taken as the initial solution to Stage 2. In this stage all unknown laterals are allowed to move to improve the measurement match at the start-of-circuit location. In other words, all unknown laterals are treated as *associated* laterals only with the start of circuit. In Stage 1 this did not happen because laterals were moved only to match their *associated* interior measurement. If there are no interior measurements, then Stage 1 is skipped and only Stage 2 is performed.

3.5 Illustrative Examples

This section presents a number of examples. The examples are chosen to illustrate different aspects of the phase prediction problem. The first example discusses some obstacles inherent in the problem that limit finding a unique solution. The second example involves a 20-lateral circuit as to test the algorithm developed in this study. The last example using a very simple circuit illustrates the idea of how the phase prediction application can be used as a phase balancing application.

3.5.1 Example 3.1: Degeneracies

There may be situations where a phase prediction algorithm fails to find the *true solution* to the problem. In these cases any combinatorial optimization technique, no matter how accurate it is, would be unable to identify the *true solution*. Unfortunately, achievements in capturing consistent measurements that reflect the actual existing flows, constructing circuit models that are the exact representation of the reality, and obtaining statistical data that coincide with the actual load behavior will not help at all to diminish the inability to identify the *true solution*.

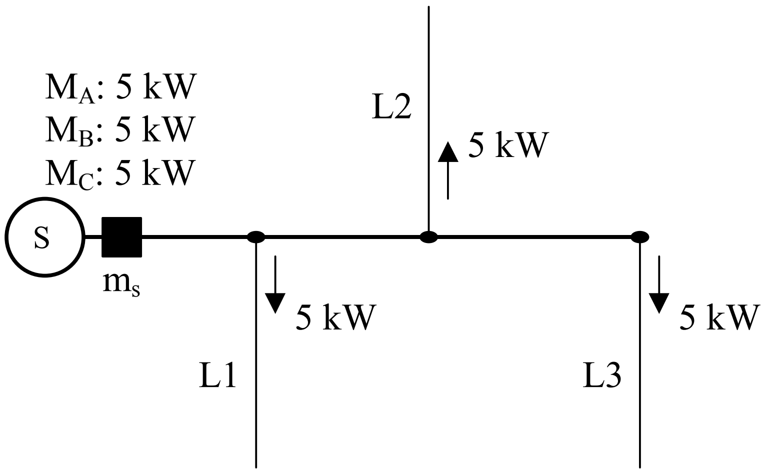


Figure 3.5 A simple circuit with three unknown single-phase laterals of equal flows.

A case with this type of inability is presented in Figure 3.5. In the very simple circuit shown in the figure, the flows on the laterals are equal. Also, measurements on the phases at the start of circuit are the same. This situation corresponds to a degenerate case where there exist multiple solutions. It is easy to see that the phasing patterns ABC, ACB, BAC, BCA, CBA and CAB all lead to an objective function value of 0 —which is the optimal objective function value.

Consequently, the algorithm will identify these solutions as being optimal, based on the definition of the objective function of the problem. That is, all are mathematically the optimal solution to the problem. However, the reality of distribution engineering does not give credit to every optimal phasing pattern. Only one pattern is out there in the circuit, leading to flows as measured at the start-of-circuit location. We call this particular phasing pattern to be the *true solution* to the problem. Therefore, from the standpoint of distribution engineering it is this solution that is being sought, not all optimal solutions. Towards this end, the phase prediction algorithm has nothing to offer.

To recap, it may be said that those laterals that have flows close to one another have a high chance of having phases predicted wrongly with respect to the distribution engineering requirements. Another class of cases that cause the algorithm to fail to predict the *true* phases are those where there exist laterals that have flows that are close or equal to the sum of the flows of some two or more other laterals. In such cases the phases of these 'big-flow' laterals and the phases of the 'small-flow' laterals (i.e., laterals whose flows altogether amount to the flow of some 'big-flow' lateral) are very likely to be predicted wrongly.

Consider the circuit in Figure 3.6. Here, L1 has a flow that is equivalent to the sum of the flows of L2 and L3. As far as the objective value is concerned, the phasing patterns ABBC and BAAC are no different from each other —because both lead to the same objective value. Also, these two patterns are the optimal solutions to the problem. Yet, one of them is the *true solution*. However, the one which has been encountered first during the iterative process of TS will be reported to be the solution. Suppose that ABBC is the *true solution* but the algorithm reports BAAC as the solution to the problem. Then the phases of L1 ('big-flow' lateral), L2 and L3 ('small-flow' laterals) would be predicted wrongly.

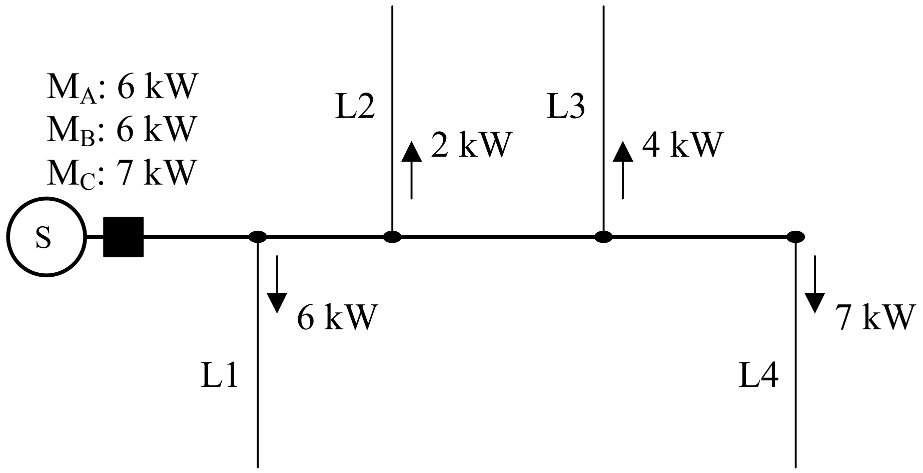


Figure 3.6 A simple circuit involving a lateral (L1) that has a flow equivalent to the sum of the flows of two other laterals (L2 and L3).

It is worth mentioning that inaccuracies in measurements and system (i.e., circuit and load) modeling become more prevailing on the outcome of the phase prediction when lateral flows as well as measured flows are in a small neighborhood of one another. Figure 3.7 illustrates such a situation. Given a particular time point for consideration, the actual kW flows (i.e., the flows that occur at the real circuit) are represented in Figure 3.7(a). The *true* phases for L1, L2 and L3 are A, B and C, respectively. Notice how close to one another the lateral flows are. For this three-lateral system measured flows also are very close to one another.

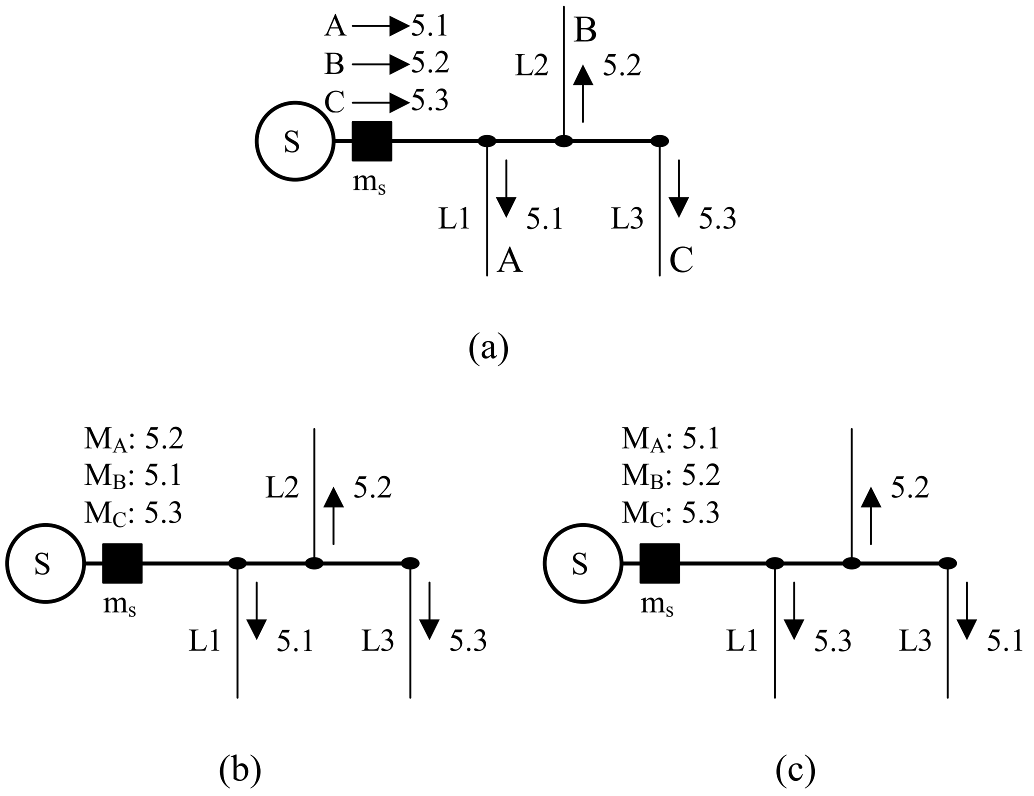


Figure 3.7 A simple circuit subjected to inaccuracies in system modeling and measurements. (a) Circuit with actual flows. (b) Circuit with accurate system modeling and inaccurate measurements. (c) Circuit with inaccurate system modeling and accurate measurements.

Let us assume that the circuit and loads are modeled accurately. Therefore the same lateral flows as the actual lateral flows are obtained by the load estimation calculation. Let us also assume that there are some errors in the measurements. Figure 3.7(b) shows an exact model of the circuit, involving the lateral flows and the erroneous measurements. Both lateral flows and measurements are evaluated considering the same particular time point of Figure 3.7(a). Phase C kW flow is measured correctly whereas phase A and phase B measurements involve errors of 0.1 kW. Based on these flows, the phase prediction application will come up with phase B predicted for L1, a phase A for L2 and a phase C for L3. Obviously, this phasing pattern will be not equivalent to the *true* phasings that are given in Figure 3.7(a).

Now, let us assume that measurements exactly reflect the actual flows but the circuit and loads are not modeled accurately. Then, lateral flows such as the ones depicted in Figure 3.7(c) will be obtained by the load estimation calculation. With these flows and measurements, the phase prediction application evaluates a phase C for L1, a phase B for L2 and a phase A for L3, which is again not the true phasing pattern.

3.5.2 Example 3.2: Test on VA Tech Demo Circuit

This example will consider a test-case circuit involving 20 laterals. The circuit is not shown here due to its large size. Nonetheless, the lateral kW flows are given as shown in Table 3.4. Most laterals in the test circuit consist of more than one component, and the flows in the table are those that occur on the first component of each lateral. Some laterals are of double-phase type, and for such laterals flows are represented by a set of two kW flow values. The start-of-circuit three-phase kW measurement that will be used in phase prediction is as follows: 3876 kW for phase A, 835 kW for phase B and 335 kW for phase C.

Results of the phase prediction study performed on this circuit are given in Table 3.5. As can be seen under the 'comment' column, the majority of the phases are predicted correctly. Phase prediction has failed to find the *true* phases only for the laterals represented by the components *2_Ph_B*, *4_Ph_A* and *8_Ph_B*. Recall that this is due to the fact that *4_Ph_A* (the 'big-flow' lateral) has a flow which is equivalent to the sum of the flows of *2_Ph_B* and *8_Ph_B* (the 'small-flow' laterals). Here, it is apparent that the *true* phase of the 'big-flow' lateral has become the *predicted* phase for 'small-flow' laterals, and the *true* phase of 'small-flow' laterals has predicted in place of the 'big-flow' lateral phase.

Table 3.4 True Phases and kW Flows of the Laterals of VA Tech Demo Circuit.

Component Name	*True* Phasing	kW Flows
McCoy_Combo_Cutout_1	A	356
McCoy_Line_1	AB	696.4, 115.4
Lick_Run_Recloser	A	212
Whitethorn_Fuse	A	101
Vicker_Switch_Fuse	A	410
Fairlawn_Line_2	AB	432, 205
Walton_Mnt._Line_1	AB	679, 256
Fire_Tower_Line_1	AB	183, 56
C_Creek_Combo_Cutout	A	241
Newport_Combo_Cutout	A	179
1_Ph_A	A	31.56
2_Ph_B	B	63.11
3_Ph_C	C	94.67
4_Ph_A	A	126.2
5_Ph_B	B	75.73
6_Ph_C	C	50.49
7_Ph_A	A	56.8
8_Ph_B	B	63.11
9_Ph_C	C	189.3
10_Ph_A	A	82.04

A *tabu tenure* value of 4 has been used in the above study. Changes in the objective value as the algorithm progressed over the iterations can be seen in Figure 3.8. The solution has been obtained at iteration 18.

Table 3.5 Phase Prediction Results for VA Tech Demo Circuit. +: correct prediction, –: wrong prediction.

Component Name	*True* Phasing	*Predicted* Phasing	Comment
McCoy_Combo_Cutout_1	A	A	+
McCoy_Line_1	AB	AB	++
Lick_Run_Recloser	A	A	+
Whitethorn_Fuse	A	A	+
Vicker_Switch_Fuse	A	A	+
Fairlawn_Line_2	AB	AB	++
Walton_Mnt._Line_1	AB	AB	++
Fire_Tower_Line_1	AB	AB	++
C_Creek_Combo_Cutout	A	A	+
Newport_Combo_Cutout	A	A	+
1_Ph_A	A	A	+
2_Ph_B	B	A	–
3_Ph_C	C	C	+
4_Ph_A	A	B	–
5_Ph_B	B	B	+
6_Ph_C	C	C	+
7_Ph_A	A	A	+
8_Ph_B	B	A	–
9_Ph_C	C	C	+
10_Ph_A	A	A	+

3.5.3 Example 3.3: Using as Phase Balancing Application

As mentioned in Section 3.1, it is possible to make the phase prediction algorithm act as phase balancing algorithm. This can be achieved in such cases where no limitation is imposed on the number of phase moves to be performed. The phase prediction will then be free to re-phase (i.e., predict the phase of) any lateral in the circuit. Recall that it is not practical to re-phase a 'live' circuit unboundedly since every phase move will interrupt the electric service to a certain number of customers. However, one can re-phase the circuit freely in situations where electricity is out anyway such as a power blackout due to a major storm or a power outage due to failure in the transformer(s) serving the area or due to a re-building process converting the circuit from one voltage level to another.

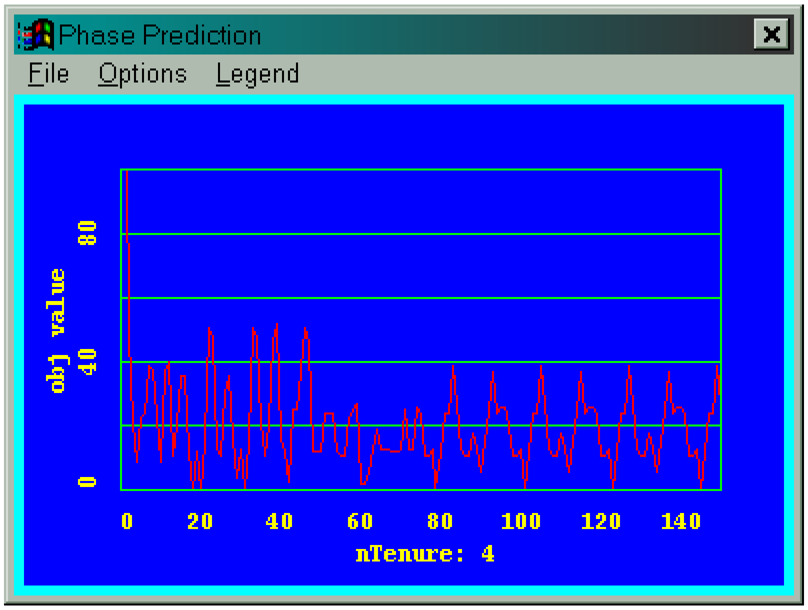


Figure 3.8 Objective function value vs. iterations for phase prediction on VA Tech Demo Circuit.

The task of using the phase prediction as phase balancing consists of re-arranging the flows of the component to be balanced and denoting the laterals *associated* with this component as unknown. To illustrate, consider the very simple circuit given in Figure 3.9(a). The kW flows that occur at a particular time point are shown. Definitely, the circuit is in need of phase balancing because of the extremely unbalanced operation caused by the all laterals being connected to phase A.

To this end, all laterals are to be treated as unknown and the phase prediction is run considering a scenario whereby the average of the flows at m_s is used in the place of every single-phase kW measurement at m_s. This procedure is depicted in Figure 3.9(b). The average flow at m_s for the time point considered amounts to 5 kW (5=(15 + 0 + 0)/3). Then, this average value is treated as the measured kW flow for phase A, phase B and phase C at m_s. There exist six solutions to this problem: ABC, ACB, BAC, BCA, CAB, CBA. The three letters in a solution denote the phases for L1, L2 and L3 respectively. Any one among the six solutions can be accepted as the solution as far as the phase balancing problem is concerned. Recall that only one of them was the *true solution* to the phase prediction problem discussed in Section 3.5.1.

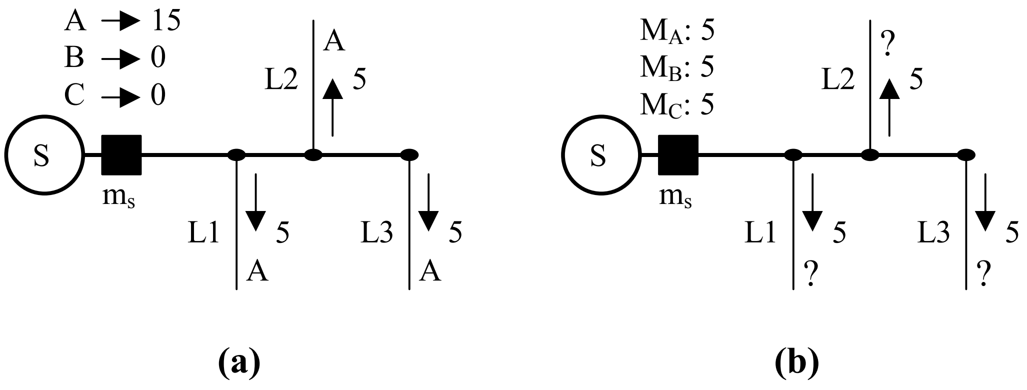


Figure 3.9 Using the phase prediction for phase balancing. (a) An unbalanced circuit with known laterals and kW flows. (b) Circuit as used by the phase prediction for performing phase balancing, where laterals are treated as unknown and the measurements at m_s are obtained by the manipulation of the kW flows at m_s of (a).

Assume the solution reported by the phase prediction is ABC. This phasing pattern will result in kW flows of 5 kW for phase A, 5 kW for phase B and 5kW for phase C. Based on this report, L2 should be re-phased to phase B and L3 should be re-phased to phase C. Therefore, performing phase prediction with using equal single-phase measurements is equivalent to finding the best phasing pattern that will lead to as equal single-phase flows at the measurement location as possible. This process is nothing but phase balancing.

3.6 Conclusion

This chapter describes a new algorithm for the prediction of phases associated with single- and double-phase laterals on radial electrical distribution circuits. The algorithm uses basic principles of TS. Implemented as a design module in the EPRI DEWorkstation software package, the routine has been tested on a number of circuits. Considering the detrimental fact that the phase prediction problem by nature is susceptible to multiple solutions among which only one is the *true solution,* and that inaccuracies in measurements and system modeling can easily cause wrong predictions, the results of these tests have been found successful.

3.7 References

[3.1] R. Broadwater, J. Thompson, M. Ellis, H. Ng, N. Singh and D. Lloyd, "Application Programmer Interface For The EPRI Distribution Engineering Workstation," *IEEE Transactions on Power Systems*, vol. 10, no. 1, February 1995, pp. 499-505.

[3.2] Robert Broadwater, Al Sargent, Abdul Yarali, H. Shaalan and Jo Nazarko, "Estimating Substation Peaks From Load Research Data," *IEEE Transactions on Power Delivery*, vol. 12, no. 1, January 1997, pp. 451-456.

[3.3] A. Sargent, R. P. Broadwater, J. Thompson and J. Nazarko, "Estimation of Diversity and KWHR-to-Peak-KW Factors from Load Research Data," *IEEE Transactions on Power Systems*, vol. 9, no. 3, August 1994, pp. 1450-1456.

[3.4] R. P. Broadwater, J. Thompson, and T. E. McDermott, "Pointers and Linked Lists in Electric Power Distribution Circuit Analysis," *Proceedings of 1991 IEEE PICA Conference*, Baltimore, Maryland, May 1991, pp. 16-21.

[3.5] F. Glover, "Heuristics for Integer Programming Using Surrogate Constraints," *Decision Science*, vol. 8, 1977, pp. 156-166.

[3.6] J. A. Bland and G. P. Dawson, "Tabu Search and Design Optimization," *Computer-aided Design*, vol. 23, no. 3, April 1991, pp. 195-201.

[3.7] F. Glover, "Tabu Search: A Tutorial," *Interfaces*, vol. 20, no. 4, July-August 1990, pp. 74-94.

[3.8] C. R. Reeves (ed.), *Modern Heuristic Techniques for Combinatorial Problems*, J. Wiley, New York, 1993.

[3.9] F. Glover, "Tabu Search – Part I," *ORSA Journal on Computing*, vol. 1, no. 3, Summer 1989, pp. 190-206.

[3.10] F. Glover, "Tabu Search – Part II," *ORSA Journal on Computing*, vol. 2, no. 1, Winter 1990, pp. 4-32.

[3.11] F. Glover, E. Taillard and D. de Werra, "A User's Guide to Tabu Search," *Annals of Operations Research*, vol. 42, 1993, pp. 3-28.

[3.12] F. Glover, E. Taillard, and D. de Werra (eds), *Annals of Operations Research*, vol. 42, 1993.

Chapter 4 Integrated Design in Electrical Distribution

This chapter will introduce the concept of integrated design. When a distribution system is in need of improvement relative to some system aspect, integrated design considers using results of multiple design applications rather than employing a single application. To illustrate the concept, an example involving capacitor placement and phase balancing design is given. A software architecture for distribution engineering analysis and design is presented, with an emphasis on open architecture and object–oriented technology.

4.1 Introduction

Many computer-aided engineering tools have been developed for electrical distribution design problems. A number of the tools have been reported in the literature. For example, different tools have been developed for addressing how capacitor banks should be located and sized on the distribution system [4.1], [4.2]. Some capacitor design tools address improving power factor and decreasing losses while others address the improvement of the voltage profile. Examples of other tools include reconfiguration for minimum loss, reconfiguration for improved reliability, phase balancing, re-conductoring, and the placement and operation of voltage control devices.

What a utility really faces in this arena is a multi-objective design problem. Just for the single design problem of placing a capacitor, a utility may be interested in controlling both voltage and losses. However, a utility is faced with selecting among various design alternatives and the effects that each alternative has upon the utility's objectives. At the highest level, a utility is interested in reliability, performance, efficiency, peak values, costs, and protection.

The picture is further complicated by the fact that competing design alternatives may affect one another if considered together. For instance, assume an existing circuit is in need of design modifications. Also assume that the circuit is operating inefficiently with voltage on the low side. It could be possible to improve this circuit's performance by placing new capacitor banks or by balancing phases or by both. For instance, assume that a capacitor placement study is performed on the circuit and a set of design modifications are arrived at, and let us call this set *Design Modification Set 1*. Now assume that a phase balancing study is performed (in this study the capacitors from *Design Modification Set 1* are not included) which produces *Design Modification Set 2*. Now, if *Design Modification Set 2* is applied to the circuit and then a capacitor placement study is performed, the results of the study will in all likelihood be much different than those found in *Design Modification Set 1*. That is, if the results of one design are applied, the results of another design may be modified.

This work introduces the concept of integrated design, an approach for coordinating the effects of various design calculations. After a brief consideration of computer-aided tools, a discussion on the architecture of distribution engineering software is presented. The integrated design approach is then introduced and incorporated into the software architecture.

4.2 Categorization of Computer-aided Tools

In a broad sense, computer-aided tools that are developed to meet the needs of distribution engineers can be categorized into two groups:

- Analysis tools
- Design tools

Analysis tools basically calculate numbers about a system's state and performance. They are intended to predict or evaluate the behavior of a distribution system as a whole or any particular piece of distribution system equipment under certain conditions. Such conditions as opening a switch, a fault on a circuit part, or the starting of a large motor can be given as examples.

The most obvious analysis tool is the load flow program that calculates voltages, currents and real and reactive power flows. Other analysis tools include load estimation, fault analysis, impedance calculations and flicker analysis.

Design tools, on the other hand, make decisions about how to build or modify the distribution system, where decisions are usually based upon numbers obtained from analysis tools. A phase balancing program recommending how to connect single- and double-phase laterals to the three-phase spine of a given circuit can be given as an example of a design tool.

The utility employs design tools in order to identify alternatives, evaluate their attractiveness based upon some given criteria, and select the most attractive one(s) from among those alternatives. That is, design tools are used to help the power engineer determine which one, out of many possible design options, should be selected. In consequence, the outcome of each tool to some extent effects one or more important system objectives.

Since distribution systems are very large in size, involving 1000's of components, identifying and evaluating alternatives can become a tedious and lengthy process if done manually. Design tools accomplish this process in very small fractions of time. In addition, design tools can apply optimization methods to find the best options [4.3].

In essence, analysis tools predict what will happen under given conditions whereas design tools evaluate what should be done relative to some defined objectives. Classifications of applications different from the one given here can be found in the literature [4.4], [4.5]. For instance [4.4], divides applications into four groups as analysis, design, planning and operation applications. Nonetheless, in the sense of the way analysis and design tools are designated in our study, applications belonging to any of these four groups can be categorized into one of our two classes.

4.3 Subsystem Diagrams

A software system developed for distribution engineering is in general complex. The complexity increases the burden to manage the system in terms of the following software qualities: performance, functionality, maintainability, portability, understandability, testability, scalability and flexibility.

Although each of the software qualities has its own merit, among these qualities maintainability and flexibility bear special importance. All changes made to software after it has been released are traditionally called maintenance. Users' needs change over time, even while software is being developed [4.6]. A software system should be able to evolve (or be upgraded/enhanced) as the system needs change. It is not uncommon that it becomes inevitable to replace/redesign an entire system because complexity is such that the system cannot be adopted to new requirements (flexibility). Object-oriented technology [4.7] is one solution to reducing maintenance and flexibility problems in dealing with complexities.

The architecture of a system has a direct impact on the software qualities mentioned above. The architecture of a system can be defined as the set of decisions and rules that keeps its implementers and maintainers from exercising needless creativity [4.8].

There are many architectural views that help in understanding a system. Different architectural views consider different kinds of elements such as hardware, networks, packages, objects, relationships, concurrent processes, tables and so on. Architecture helps you manage the complexity via making it easier to understand how the parts of a system are defined, what the external characteristics of each part are, how the relationships between parts are defined, and how they interact [4.8].

In our study here we will view the architecture of the system in terms of subsystems, interfaces, dependencies, and the navigability of associations between subsystems. Moreover, we will use the Unified Modeling Language (UML), an emerging industry standard for object-oriented modeling. The UML is "a language for specifying, visualizing and constructing the artifacts of software systems" [4.9].

The UML provides the package mechanism for combining groups of elements. Thus, a package is a general-purpose mechanism used for organizing elements into groups. A subsystem is a special type of a package. It represents a grouping of elements of which some constitute a specification of the behavior offered by the other contained elements [4.9]. A subsystem consists of two parts, a specification part and a realization part. For example, a use case diagram may be found in the specification part and a class diagram in the realization part.

Subsystems are used to decompose a complex system as much as possible into nearly independent parts. In other words, subsystems that make up a system provide a complete and non-overlapping partitioning of the system as a whole [4.9]. A subsystem is represented by a tabbed-folder marked with *<<subsystem>>*.

A subsystem may have one or more interfaces. An interface of a subsystem (or any element in general) specifies the set of operations that are visible outside the subsystem [4.6]. That is, it describes the set of requests to which an element can respond [4.10]. It is graphically represented as a small circle attached to the icon of the element it belongs to. In our study we consider two kinds of relationships between subsystems:

- Association with navigability
- Dependency

Note that either of the above relationships implies the other.

An *association* exists between two subsystems if there exists a meaningful connection between a part of one subsystem (let this part be *A*) and some part of the other subsystem (let this part be *B*). This *association* is said to have navigability when it is possible to navigate across the association, say from *A* (source) to *B* (target). In other words, we say that the source subsystem knows about the target subsystem [4.11].

In general, an *association* with navigability between two subsystems is defined when an object of type *A* knows about an object of type *B* and can send a message to the object of type *B*. This can occur under the following circumstances:

- *A* contains an attribute that refers to the object of type *B*.
- *A* creates an instance of type *B*
- *A* receives the object of type *B* as an argument of a message

A subsystem (the dependent) may be dependent on another subsystem (the target). The dependency relationship indicates that the implementation or functioning of one or more elements in the dependent subsystem requires the presence of one or more other elements in the target subsystem [4.12].

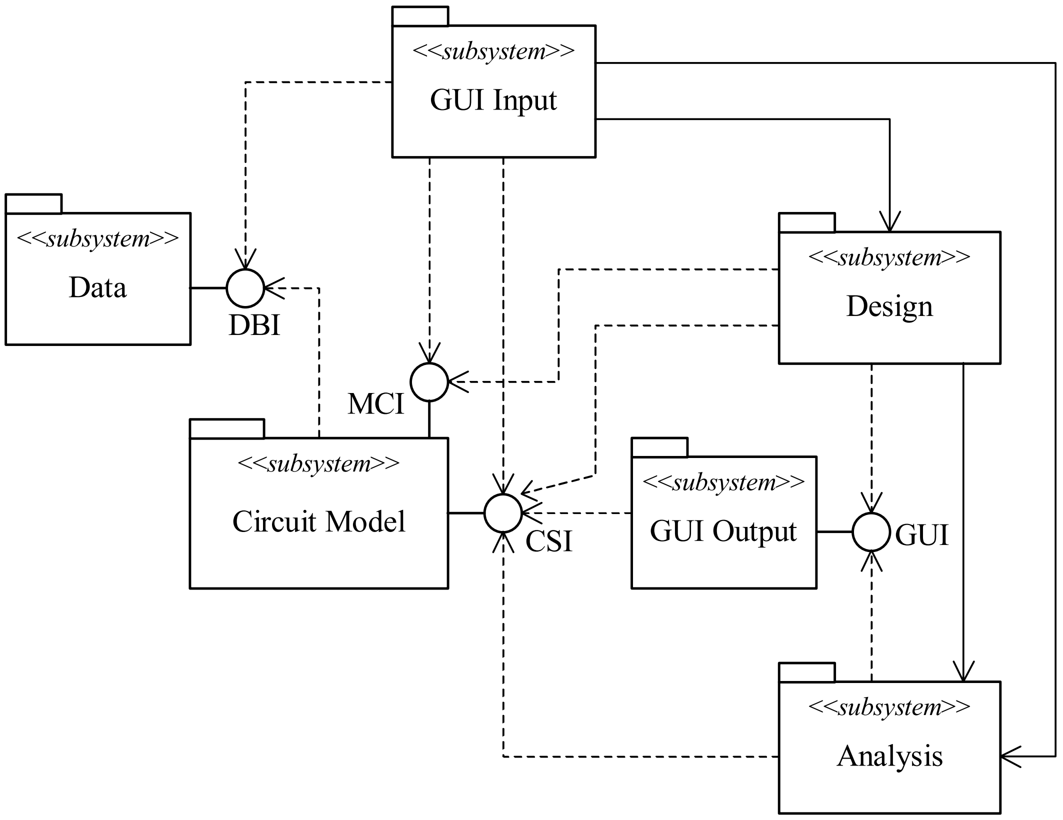
In UML the *association* with a navigability relationship is denoted by an arrowed line from the source to the target whereas a dashed arrow drawn from the dependent to the target represents the dependency relationships. In both cases, the head of arrow points to the target elements.

A UML diagram that depicts the architecture for a distribution engineering system is shown in Figure 4.1. The system consists of six subsystems. The navigability of all relationships presented in the figure is unidirectional. This helps eliminate circular dependencies between subsystems. A circular dependency between a pair of subsystems means that each has the other as a target. Such a dependency is not desirable because a change made to either subsystem will force an examination of the other subsystem for needed changes —with a circular dependency, this is theoretically an infinite loop. Circular dependencies result in increased maintenance and can eventually lead to unmaintainable systems.

The *Data Subsystem* in Figure 4.1 has the responsibility for the persistent storage and retrieval of data associated with the circuit models and analysis/design applications [4.13]. Commonly used data such as corporate, Supervisory Control and Data Acquisition (SCADA), engineering, financial, operations, geographical, and statistical can be included in the *Data Subsystem*. The *Data Subsystem* provides an interface, *DBI*, which is used by the *GUI Input* and *Circuit Model* subsystems. The *Circuit Model Subsystem* manages the majority of data access for applications.

In order to avoid circular dependencies in the software architecture, the graphical user interface elements have been divided into two subsystems, one for input, *GUI Input*, and another for output, *GUI Output*. The *GUI Input Subsystem* is responsible for gathering all graphical user input. The

GUI Input Subsystem sends messages to one of three subsystems: *Circuit Model, Analysis, or Design*.



DBI	: Database Interface	MCI	: Model Change Interface
CSI	: Circuit System Interface	GUI	: Graphical User Interface

Figure 4.1 Subsystem diagrams for the architecture of a typical distribution engineering analysis and design software.

The *GUI Input Subsystem* uses the *MCI* interface of the *Circuit Model Subsystem. MCI* stands for Model Change Interface. *MCI* provides capability for creating and changing circuit models. For instance, via *MCI* such modifications as the following may be performed: circuits may be created or deleted; circuit components may be added, inserted, or deleted; phasing may be changed and loads may be modified.

The *GUI Input Subsystem* sends messages that cause applications in the *Design* or *Analysis* subsystems to run. The messages contain information that is gathered from the user and/or *Data Subsystem* and is needed by the applications. The information passed includes the circuits that are to be analyzed. In order to retrieve information about the circuits to be analyzed, the *Design* and *Analysis* subsystems use the *CSI* interface of the *Circuit Model* Subsystem.

When output is desired from analysis or design applications, the *Design* or *Analysis* subsystems use the *GUI* interface of the *GUI Output Subsystem* to display results.

The *Analysis Subsystem* contains all analysis applications. As described in the previous section, these applications perform such engineering calculations on circuit models as line impedance calculations, load estimation, power flow, fault, and reliability analysis.

The *Design Subsystem* contains all applications that are defined as design tools in the previous section. This subsystem is responsible for calculating new additions and/or modifications to the circuit models that will result in improved performance, reduced cost, reduced peaks, improved reliability, improved power quality and adequate protection. As part of its decision making process, the *Design Subsystem* obtains analysis results from the *Analysis Subsystem*. The *Design Subsystem* uses the *MCI* interface to affect additions/changes to the model.

Notice that many of the subsystems in the figure are dependent on the *Circuit Model Subsystem*. This implies the importance of having robust circuit models. The majority of existing software systems have been developed using function-oriented methodologies, where functions are the building blocks for the software. With this approach, modeling of the distribution system becomes application dependent. That is, the way the system has been modeled largely depends on the way applications have been developed. Functions change as the requirements change. With this approach, the maintenance of the software may become so difficult that the entire system must be redesigned [4.14].

A circuit model developed with object-oriented technology that has identified a "stable set of domain classes" can handle frequent changes without having to redesign the fundamental objects. Object-oriented technology concentrates on identifying and organizing problem domain objects, which are more fundamental and stable than the procedural requirements.

Another important aspect that the software should have is open architecture. An open architecture enables utilities or third party vendors to develop and add new applications, and to customize an existing application in the software. Furthermore, it allows external data (e.g., voltage, current and temperature measurements, customer information system data and so on) to be imported, displayed and made available to any application within the software.

As can be seen between the *analysis* and the *design* subsystems in Figure 4.1, the open architecture provides an environment in which applications can request data or results from other applications. Furthermore, users can easily add, delete, or replace applications. For example, many applications within the distribution engineering software illustrated in Figure 4.1 may depend on results from a specific application, such as power flow. In an open-architecture environment, the existing power flow may be replaced with a new, enhanced version and all applications will continue to work without modification [4.15], [4.16], [4.17].

4.4 Integrated Design Concept

Each design tool works independently toward improving some system attributes. The phase balancing design tool, for example, affects efficiency, performance, peak load and the cost of the power delivery system. Similarly, the capacitor design tool may impact the same attributes even though the extent of the impact might not be the same as in phase balancing. A reconfiguration design tool, on the other hand, can significantly affect reliability while a capacitor design or a phase balancing tool has no such considerable impact on reliability.

Therefore, attributes affected by the use of one design tool could partly/fully overlap those of some other tool(s). Here, given a system attribute (objective), and the design tools affecting that objective, a question can be raised as to whether these tools can be brought together to work more effectively toward the common objective.

To illustrate, let us assume that there exist two design tools: *Tool i* and *Tool j*. One can use these tools to improve some system objectives. The manner of impact does not matter. That is, the use of a particular tool could have a direct effect on some objective while some other objective is affected indirectly. Let *Tool i* affect objectives *x* and *y*, and *Tool j* affect objectives *x* and *z*. Suppose that one is interested in obtaining improved measures in regard to objective *x*. Having the fact that both tools are affecting the desired objective *x*, how can these design tools be combined to achieve a better system design from the standpoint of objective *x*?

The most basic way to interpret the term ‘combining’ is ‘ordering’. Ordering means applying the tools in a temporal sequence, first one tool, then the next, and so forth. For a two-tool case such as the one we have above, two ordering possibilities exist: Either *Tool i* can be applied prior to *Tool j* or *Tool j* can be applied prior to *Tool i*. Of course, the number of combinations can be increased by including other orderings such as *Tool i–Tool j–Tool i* or *Tool j–Tool i–Tool j–Tool i,* where tools are being applied more than once. However, we will consider applying a given tool only once to keep the discussion simple.

In Figure 4.2(1), the first tool applied to some distribution system *DistSystem* that is in need of improvement with respect to the objective *x* is *Tool i*. *Tool i* recommends some changes to *DistSystem*. Let us name these changes as *Modification Set i*, or *Mod_i* for short. *DistSystem* is modified based on *Mod_i*, and the system is transformed into *$DistSystem_i$*. Then, *Tool j* is applied to *$DistSystem_i$*. *Tool j* recommends some changes referred to as *Modification Set ij*, or *Mod_{jj}*. Modifying *$DistSystem_i$* according to *Mod_{ij}* results in *$DistSystem_{ij}$*, the final form of the system.

In the second ordering Figure 4.2 (2), *Tool j* is applied first to the original system. *Modification Set j* (or *Mod_j*) is given by *Tool j*. The system is modified according to the changes in *Mod_j*, and *$DistSystem_j$* is obtained. Then, *Tool i* is applied to *$DistSystem_j$*. This operation produces another modification set, *Mod_{ji}*. The final form of the system, *$DistSystem_{ji}$* is reached after modifying *$DistSystem_j$* based on the changes given by *Mod_{ji}*.

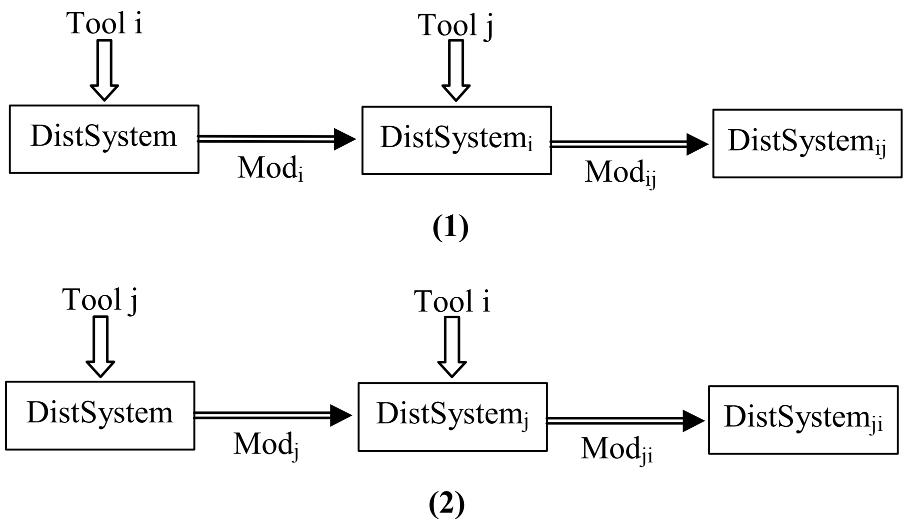


Figure 4.2 Two possible orderings for tools *i* and *j*.

The two final systems, *$DistSystem_{ij}$* and *$DistSystem_{ji}$*, are in general not the same. Hence, two different systems could be derived from a given system by using two distinct design tools, both affecting a particular system attribute. This provides the ability to have multiple alternatives, each of which might result from the combining of multiple design tools. Having this ability makes the decision more effective as compared to the conventional approach where a single alternative is derived from a single design tool.

It might be possible to identify a generic rule ("rule of thumb") in regard to application order by looking at which final system is better in terms of a particular objective. For example, if *$DistSystem_{ij}$* is more efficient than *$DistSystem_{ji}$* then a rule such as "When efficiency is the main concern, performing *Tool j* followed by performing *Tool i* is better than performing *Tool i* followed by performing *Tool j*" can be stated. This rule identifying process is experimental. It is not necessarily a rule that is always valid. If a rule happens to be working for the majority of occasions, e.g. 80%, then it will be sufficient for that rule to be a "a rule of thumb."

A sequence of modifications that will be performed on a given distribution system in an orderly fashion can be regarded as a *project* associated with the system. That is, a *project* is a collection of changes that are arranged so as to be consecutively applied to a given system. The sets Mod_i and Mod_{ij} in Figure 4.2(1) can be aggregated to form a *project* while the sets Mod_j and Mod_{ji} can be gathered to form another *project*. However, the collection of modifications Mod_i and Mod_j is not a *project* because they are not consecutive.

A concrete example will be presented next which illustrates how results from two different tools can be brought together and how the concept of a *project* can be used for this merging process.

4.4.1 Example 4.1: Illustration of Project

Consider the distribution system given in Figure 4.3. It is desired to improve the efficiency (decrease the losses). Suppose that we have two tools that we know can impact losses on the system: a capacitor design program (CD) and a phase balancing program (PB).

CD has the ability to determine locations, sizes and types of capacitors for reactive power compensation, power loss reduction and voltage regulation problems. Besides, it can rank the candidate locations based on the impacts they are having, for example, on power losses.

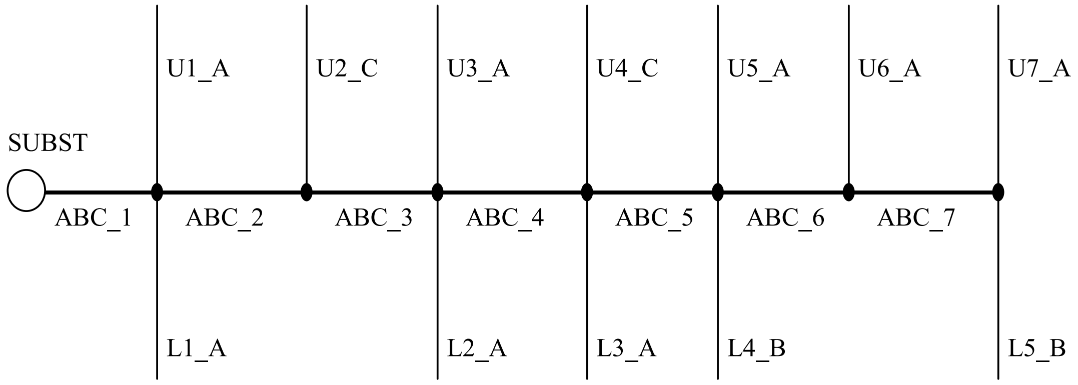


Figure 4.3 Circuit for Example 4.1 where SUBST: Substation, thick lines: three-phase feeder, thin lines: single-phase laterals, last letter of single-phase label: phase of lateral (e.g., U1_A: a single-phase lateral of phase A).

Similarly, PB suggests phase move operations regarding how single- or double-phase laterals should be connected to the three-phase portion of the system in order to obtain a new layout with less loss. For instance, PB may recommend a lateral, which is initially attached to phase A, be connected to phase B.

Assume that there are limits to be considered about the type and number of modifications: It is given that only one capacitor placement operation can be afforded and at most two phase move operations can be accepted. Hence, out of locations suggested by CD, only the one that leads to the best power loss improvement is to be considered. Similarly, the best two phase move recommendations should be taken into consideration.

Two projects will be formed: the *PB–CD Project* and the *CD–PB Project.* In the former case, PB is run on the system and sends the two best modifications to the *PB–CD Project.* After modifications based on these changes are applied to the system, the system undergoes a CD run. The recommendation given by CD is then appended to *PB–CD Project.*

In the latter case, CD is performed first. The recommendations are written into *CD–PB Project.* After the system is modified according to the CD recommendation, PB is then run. PB gives two modifications, and they are appended to *CD–PB Project.*

Table 4.1 Modifications Obtained by Two Projects.

	PB–CD Project	***CD–PB Project***
1st Mod.	Re-phase L2_A from A to C	Place 400 kVar at ABC_2
2nd Mod.	Re-phase U3_A from A to B	Re-phase L2_A from A to C
3rd Mod.	Place 400 kVar at ABC_3	Re-phase U6_A from A to C
Loss	150 kW	133 kW

Thus we have two projects each consisting of three modifications. The results are shown in Table 4.1. The circuit losses obtained at the end of the two projects are recorded in the last row of the table. It is clear that the *CD–PB Project* has superior performance. Thus, one can make the decision in favor of the *CD–PB Project.*

Not only modifications given by design tools but also modifications performed by the user can be included in a *project*. For instance, a user may add a switch operation (e.g., opening a closed switch or closing an open switch) to an existing *project* which already contains some design tool modifications.

A variety of operations can be regarded as a modification. Included in these are phase moves, switch operations, disconnecting a group of elements and reconnecting it at some other point in the system, modifying an existing load or adding a new load, changing the type/size of conductors over some circuit segment, deleting an existing component, and inserting new components.

In the next section we will visit the software structure discussed earlier, and then adopt it to the concepts mentioned above. To this end two new subsystems will be introduced:

- *Integrated Design Subsystem*
- *Evaluation Subsystem*

4.5 Modified System Architecture

The foregoing discussion introduced the idea of a *project*. A *project* is attached to a distribution system and is composed of modifications of various types. In UML terms, a distribution system can be *associated* with a *project* and a *project* is *associated* with a modification. This is represented pictorially as shown in Figure 4.4.

In Figure 4.4, a *project* is a *part of* a distribution system. This is denoted by the diamond figure at the distribution system end of the *association* between distribution system and *project*. The *multiplicity* of the *association* is given by 1 and *. The "*" says that the number of *projects* owned by a distribution system can be any finite number including zero. Similarly, the fact that a *project* is composed of an arbitrary number of modifications is reflected by a diamond at the project end of the association between *project* and modification, along with 1 and * at the ends of the *association*.

A modification is a *generalization* of such operations as re-phasing, open/closing a switch, adding a component and so forth. This is because, for example, every re-phasing operation conceptually is a modification. Conversely, re-phasing is a *specialization* of the modification concept. The *generalization* relationship has been represented by a small hollow triangle as shown in Figure 4.4.

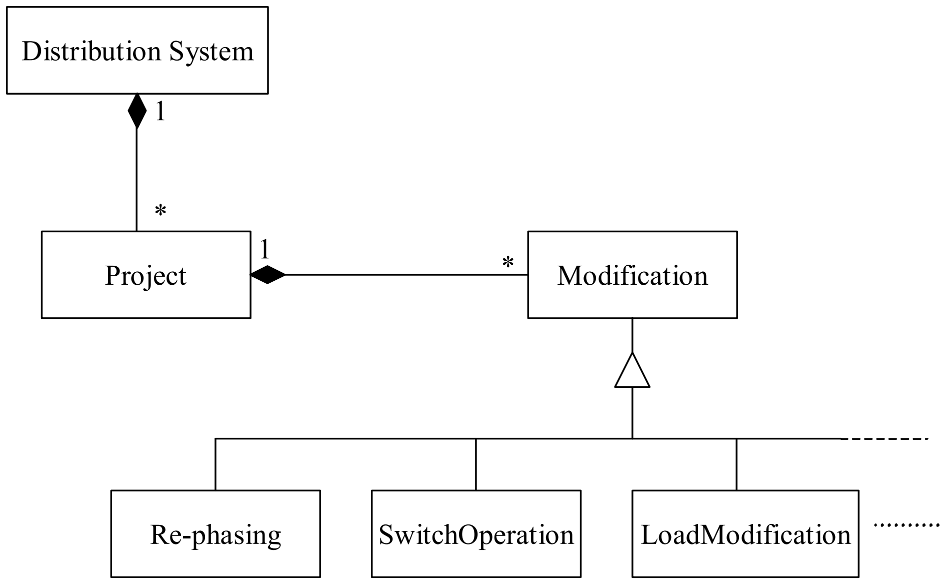


Figure 4.4 Class diagrams illustrating the relationship among a distribution system, a project and a modification.

4.5.1 Integrated Design Subsystem

A *project* provides storage for results from design tools. Despite the fact that results from different design calculations may interact with one another, the architecture shown in Figure 4.1 does not support the idea given by a *project*. That is, in Figure 4.1 there is no planned way for the design applications to cooperate.

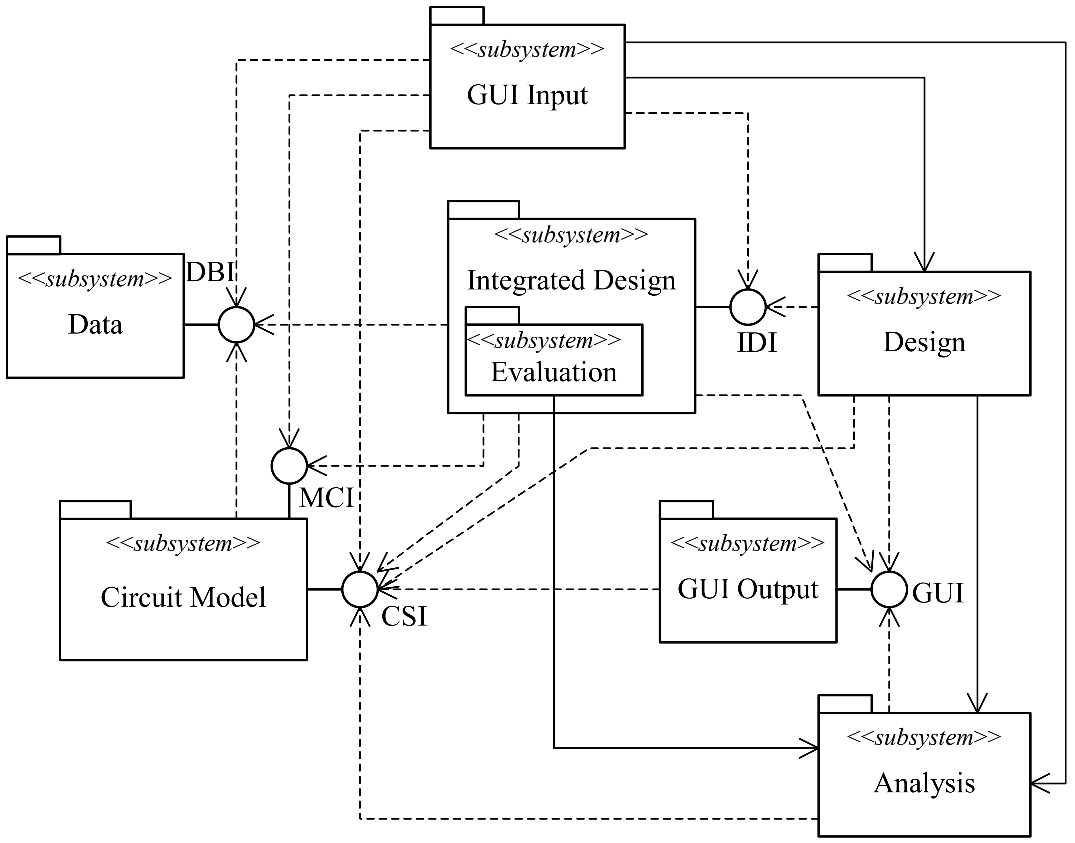


Figure 4.5 Subsystem diagrams after considering integrated design concept (IDI: Integrated Design Interface).

Now consider Figure 4.5 which represents a new approach to design. Figure 4.5 is similar to Figure 4.1, except that Figure 4.5 introduces an additional subsystem, the *Integrated Design Subsystem*. The main responsibility of the *Integrated Design Subsystem* is to combine circuit modifications from any set of design applications that use the *IDI* interface. Hence, the outputs from different design programs that do not know one another may be concatenated together into one set of modifications to be performed on the circuit models.

In Figure 4.1 each individual design application talked directly to the *MCI* interface of the *Circuit Model*. In Figure 4.5 design applications are not allowed to talk directly to the circuit model whenever they need to modify the circuit. That is, in order to effect the circuit model, the design applications must use the *Integrated Design Subsystem* and its *IDI* interface. Also note that the *GUI Input Subsystem* may use the *IDI* interface so that direct user modifications may be mixed with the outputs of various design programs

Notice that a *project* is not a distribution system. Rather, it is the set of changes that are planned for the distribution system. That is why only the changes are to be stored in the *Data Subsystem*. The *Integrated Design Subsystem* can access the *Data Subsystem* to read any *project* associated with a given distribution system, and can make the changes accordingly on the system. Likewise, via operations such as adding some new modification or deleting some existing modifications, an existing *project* can be manipulated by the *Integrated Design Subsystem* and then stored back to the *Data Subsystem*.

Another responsibility of the *Integrated Design Subsystem* is to measure projects against a given system aspect (e.g., efficiency, reliability and so forth). Actually, the evaluation of a system aspect is provided as a service from analysis tools. For instance, the circuit loss calculation can be handled by a power flow program. The *Integrated Design Subsystem* delegates the project measurement job to the *Evaluation Subsystem*, a subsystem within the *Integrated Design Subsystem*. The *Evaluation Subsystem* knows what system aspect is handled by what analysis tool, and can make calls to analysis tools.

Let us assume that there exist four *projects* obtained from the following strategies:

- Strategy 1: Place capacitors
- Strategy 2: Perform phase balancing
- Strategy 3: Place capacitors followed by phase balancing
- Strategy 4: Phase balance followed by placing capacitors.

The *Integrated Design Subsystem* is asked to find the best one based on the minimum circuit loss aspect. Then the *Integrated Design Subsystem* will use the *Evaluation Subsystem* to manage four different strategies. The *Evaluation Subsystem* compares the alternative designs. In performing evaluations, the *Evaluation Subsystem* requests each *project* to be implemented in turn. For each such *project*, the *Evaluation Subsystem* calls the *Analysis Subsystem* (the power flow program to be specific) and gathers circuit loss results. The results concerning the best *project* can then be reported.

4.6 References

[4.1] J. C. Carlisle, A. A. El Keib, D. Boyd and K. Nolan, "A Review of Capacitor Placement Techniques on Distribution Feeders," *Proceedings of the Twenty-Ninth Southeastern Symposium on System Theory*, 97TB100097, pp. 359-365.

[4.2] G. A. Bortignon, and M. E. El Hawary, "A Review of Capacitor Placement Techniques for Loss Reduction in Primary Feeders on Distribution Systems," *Proceedings of the 1995 Canadian Conference on Electrical and Computer Engineering*, 95TH8103, vol. 2, pp. 684-687

[4.3] H. L. Willis, *Power Distribution Planning Reference Book*, New York: Marcell Dekker, Inc., 1997.

[4.4] *Distribution Workstation: Specifications*, RP 3079-1 Electric Power Research Institute, Palo Alto, CA, April 1991

[4.5] W. H. Esselman and G. Z. Ben-Yaacov, "EPRI-Developed Computer Programs for Electric Utilities," *IEEE Comp. Applications in Power*, vol. 1, no. 2, April 1988, pp. 18-24.

[4.6] R. Pooley and P. Stevens, *Using UML: Software Engineering with Objects and Components*, Addison-Wesley Longman, 1999.

[4.7] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W. Lorensen, *Object-Oriented Modeling and Design*, Englewood Cliffs, USA: Prentice Hall, 1991.

[4.8] D. D'Souza and A. C. Wills, *Objects, Components, and Frameworks With UML*, Addison-Wesley Longman, 1998.

[4.9] G. Booch, J. Rumbaugh and I. Jacobson, *The Unified Modeling Language User Guide*, Addison Wesley Longman, 1998.

[4.10] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Reading, Massachusetts: Addison-Wesley, 1995.

[4.11] C. Larman, *Applying UML and Patterns*, Upper Saddle River, NJ: Prentice Hall, 1998.

[4.12] *OMG Unified Modeling Language Specification*, Version 1.3, Object Management Group, Inc., June 1999.

[4.13] D. G. Flinn and R. C. Dugan, "A Database for Diverse Power System Simulation Applications," *IEEE Trans. Power Systems*, vol. 7, no. 2, May 1992, pp. 784-790.

[4.14] J. Zhu and D. Lubkeman, “Object-oriented development of Software systems for Power System Simulations,” *IEEE Trans. Power Systems*, vol. 12, no. 2, May 1997, pp. 1002-7.

[4.15] R. Broadwater, J. Thompson, M. Ellis, H. Ng, N. Singh and D. Loyd, “Application Programmer Interface for the EPRI Distribution Engineering Workstation,” *IEEE Trans. Power Systems*, vol. 10, no. 1, February 1995, pp. 499-505.

[4.16] J. Britton, “An Open, Object-Based Model as the Basis of an Architecture for Distribution Control Centers,” *IEEE Trans. Power Systems*, vol. 7, no. 4, November 1992, pp. 1500-8.

[4.17] L. G. Osterlund, “Component Technology,” *IEEE Comp. Applications in Power*, vol. 13, no. 1, January 2000, pp. 17-25.

Chapter 5 Conclusions

5.1 Concluding Remarks

This dissertation presents the development of four inter-related works:

- A phase balancing algorithm
- A phase prediction algorithm
- The concept of integrated design
- A software architecture for supporting integrated design in distribution engineering software systems

Phase balancing and phase prediction are related design calculations, and these design calculations need to cooperate with other design calculations. The concept of integrated design, and the software architecture for supporting integrated design addresses the need for cooperation.

The phase balancing algorithm developed in Chapter 2 has been tested successfully on large utility circuit models. For circuit models with 100's of laterals it is used to determine the most cost-effective laterals to be moved. Being able to work on both grounded and ungrounded circuit sections, the algorithm has execution times that have proven to be reasonable when tested on large circuit models. For instance, on a 400 MHz Pentium II machine, it takes about 28 minutes to evaluate phase moves on a 4000-component utility circuit for a 24-hour time-varying load pattern. Of course, the execution time decreases as we consider less load points. For example, it takes less than two minutes to solve the same problem above when a single time point (e.g. 7 p.m. on a weekday in January) is selected for analysis.

The phase balancing algorithm takes important practical considerations into account. The phase balance for both summer and winter time-varying load patterns may be designed simultaneously. The balancing at interior circuit switches may also be performed simultaneously with the balance at the substation. This allows for the independent balancing of sections of load that are switched between circuits.

The phase prediction algorithm developed in Chapter 3 has been tested on both small and large circuits. Intrinsic to the phase prediction problem is that more than one solution may lead to the same result when determining lateral phase values that will minimize the mismatch between the measured and calculated values. Also, inaccuracies in measurements or system modeling can easily lead to wrong predictions on lateral phases. Considering these facts, the phase prediction algorithm as tested on a number of actual utility circuit models has proven to be quite successful. The algorithm used is considerably fast due to the optimization method used, tabu search.

It is also noted that the phase prediction algorithm can be used as a phase balancing algorithm. Instead of using the actual three-phase measurements, one can take the average of the three-phase measurements and then use this average measurement in place of every single-phase measurement. Because the phase prediction algorithm is much faster than the phase balancing algorithm, one may prefer using phase prediction for phase balancing when there is no limit on the number of phase moves to be performed. (Remember that the phase prediction does not have a maximum number of phase moves constraint.)

In Chapter 5 a software architecture is proposed. The architecture suggests a framework where different design applications can work together and any application can be added or replaced with minimal effects on the others. This will be achieved through using open architecture.

Central to the survivability of a software system is its degree of maintainability. Maintainability has been defined as the ability to make changes to the software. Using object-oriented technology for the development of the software, and avoiding circular dependencies between software elements as much as possible, are strongly suggested for improving maintainability.

The idea of integrated design has been introduced in Chapter 5. In addition to user modifications, results from different design applications can be collected into *projects*. *Projects* provide the opportunity to have more alternatives when considering improving a given distribution system. *Projects* containing modifications that are obtained from different types of applications may give superior results to the modifications that are obtained via running a single application.

5.2 Contributions

The contributions can be summarized as follows:

- ***A new algorithm for phase balancing of large-scale, unbalanced radial distribution systems.*** The algorithm recommends re-phasing of single- and double-phase laterals in order to improve circuit loss while also maintaining/improving imbalances at various balance point locations. The algorithm is capable of handling time-varying loads when evaluating phase moves that will result in improved circuit losses over all load points. The algorithm allows the user to specify the maximum number of moves that can be afforded.

- ***A new algorithm for predicting unknown phases on large-scale radial distribution systems.*** The algorithm evaluates the best phase combinations that will lead to the minimum mismatch with measured values at measurement locations. The program allows any three-phase location to be treated as a measurement location. Real and reactive power measurements may be considered. This dissertation is the first report of attempting to determine unknown phases on a distribution system.

- ***The idea of integrated design along with the notion of a project.*** Different design applications can send their results to projects. A project may be a mixture of design application and user modifications. Projects attached to a given distribution system are evaluated from the standpoint of some system aspect. The use of projects provides more effective decisions compared to the traditional approaches. This dissertation is the first report to tackle the integration of multiple design applications.

- ***An architecture for a distribution engineering software system.*** A generic architecture is presented for managing large software systems, with an emphasis on open system architecture and object-oriented technology. The integrated design idea is supported within the architecture.

5.3 Future work

Examining the phase balancing algorithm in Chapter 2 reveals that the algorithm is heuristic in nature and does not necessarily find the global optimum. In fact, it is quite likely that the algorithm stops at a local minimum. In our case, this is not that important because most electric utilities are interested in only the best few phase moves. The algorithm can provide these phase moves even if it ends up in a local minimum.

Nevertheless, further work can be done on the phase balancing algorithm. This work would involve applying better search techniques than the one used in our study. The tabu search method, for example, may be a good choice because it is fast and it is very robust against local minimum traps.

Future work can also be recommended for the *Integrated Design Subsystem* presented in Chapter 4. This subsystem is passive in the sense that it is allocating and evaluating the modifications initiated by the user or design applications. The *Integrated Design Subsystem* can be made active by giving it the authority to call design applications as well as analysis applications.

In an active role, the *Integrated Design Subsystem* knows what types of modifications can be performed for achieving improved measures of a particular system aspect. It also knows the design applications responsible for handling a given type of modification. Moreover, it can request services from analysis applications in order to obtain results used in evaluations.

Thus, when given a system aspect to be improved (i.e., an objective), the active *Integrated Subsystem* can call the appropriate design applications and collect the results from them. In the active role the *Integrated Design Subsystem* manages decisions. It can also manage more strict cases where the user place some limitations on the types and the number of modifications to be considered such as “get me the most efficient circuit by using at most two capacitor placement operations and at most one phase move operation”.

Determining the way the *Integrated Design Subsystem* communicates with design and analysis applications and the optimization mechanism that it uses to make decisions represents challenging, but potentially very rewarding, future work.

APPENDIX: IEEE Definition of Imbalance[1]

Imbalance is a measure of the load distribution across all phases present for a given component. The imbalance is calculated using the formula below:

$$\text{imbalance} = \left(\frac{\max\{\text{deviation}\}}{\text{average}} \right) * 100/2 \quad \text{(A.1)}$$

where:

$$\begin{aligned}
&\text{average} &&= \frac{\text{phase A load} + \text{phase B load} + \text{phase C load}}{\text{sNumPh}} \\
&\text{sNumPh} &&= \text{number of phases present} \\
&\text{deviation} &&= \left| \text{ phase i load} - \text{average } \right|, \;\; \text{i} \in \{\text{A, B, C}\}
\end{aligned}$$

The imbalance ranges from 0 to 2 where 0 indicates perfect balance, and 2 represents perfectly out-of-balance cases. Because of this range, the imbalance is normally adjusted to yield a 0%-to-100% range as shown in (A.1).

[1] Adapted from *IEEE Power System Engineering, IEEE Recommended Practice for Electric Power Distribution for Industrial Plants (IEEE Red Book)*, ANSI/IEEE Std 141-1993.

Vita

Murat Dilek was born on May 26, 1971 in Aydin, Turkey. After graduating from high school in 1988, he attended Middle East Technical University (METU) in Ankara, Turkey. He received the degree of Bachelor of Science in Electrical Engineering from METU in July 1993. Following graduation, he continued to pursue his graduate study at the same university where he also worked as a teaching assistant for 15 months. In January 1995, he came to U.S. starting over his graduate study at Virginia Tech, Blacksburg, Virginia. After receiving the degree of Masters of Science in Electrical Engineering from Virginia Tech in December 1996, he was admitted to the Ph.D. program at the same department. During his Ph.D. study, he worked under the supervision of Dr. Robert Broadwater in the areas of electric power distribution system planning, design, and simulation.