STONY BROOK UNIVERSITY

CEAS Technical Report 840

Optimal Phase Balancing using Dynamic Programming with Spatial Consideration

Kai Wang, Steven Skiena and Thomas G. Robertazzi

October 12, 2017
manuscript completed August 15, 2013

Optimal Phase Balancing Using Dynamic Programming With Spatial Consideration

*Kai Wang, **Steven Skiena, *Thomas G. Robertazzi

**Electrical Engineering Department, Stony Brook University*
100 Nicolls Road, Stony Brook, 11790 USA
Corresponding author: Kai Wang. Tel: 1-631-885-5289
Email address: kaiwang.sunysb@gmail.com

***Computer Science Department, Stony Brook University*
100 Nicolls Road, Stony Brook, 11790 USA

Abstract

Unbalanced loads on feeders increase power system investment and operating costs. Three-phase lateral loads phase swapping is one of the popular methods to balance such systems. In this paper, a novel dynamic programming algorithm for phase balancing with spatial consideration is studied. This algorithm produces optimal solutions for an interesting class of instances of this NP-complete problem efficiently.

Keywords: Spatial Load Balancing, Dynamic Programming, NP-complete, Smart Grid, Feeder.

1. Introduction

Over the past 15 years, research has been conducted on three phase feeder balancing. Phase balancing aims to reduce the unbalance of loads on three phases which can bring severe voltage drops in the feeders. The majority of electric power systems utilize, in the electric distribution system, feeders which carry three phases of alternating current/voltage. It is desirable for electric utilities and providers of electric power distribution systems to have approximately equal loads on each phase. This is a problem as even if loads are initially balanced, with time loads increase, decrease, are added or removed from each phase, causing an unbalance of loads. Even during the same day there may be much variation of load on each phase of a feeder. There are two major phase balancing methods: there is feeder reconfiguration at the system level and there is phase swapping at the feeder level [2]. Phase swapping is not as well studied in the electric power literature as feeder reconfiguration. This paper is about phase swapping algorithms.

Why does one wish phases to be in balance? Phase unbalance can limit the amount of power transferred on a feeder as on an unbalanced feeder one phase may reach its maximum carrying capacity measured in amperes (i.e. ampacity) while the other two phases are then underutilized and unable to carry their full or even nearly their full amount of current. This is poor utilization of the existing power distribution network and may result in unnecessary feeder expansion and upgrades which raise utility costs. Because one phase may be near its maximum ampacity, phase unbalance can also lead to preventive breaker/relay tripping and shutdown of a feeder whose restoration also involves a cost to the electric utility.

Periodically crews rebalance feeders. This can be done during periods of maintenance or restoration. One suburban Northeast U.S. utility rebalances feeders if the percentage of unbalance exceeds 15%. Generally it takes 10 to 15 minutes to switch a load so the overall job may take an hour plus travel time to the location. Work by a crew of two employees can cost several hundred dollars. However preparatory work such as scheduling can bring the total cost to several thousand dollars for one tap change. Three factors are considered in making a decision to rebalance a feeder: the monetary cost of making the tap change(s), the expected increase in feeder balance (saved energy) and the temporary interruption of power to the customer. Tap change generally fall into two situations: a new customer is to be connected or the phase balance for existing feeders has become significantly unbalanced. Once a feeder is re-balanced it will initially be in balance but drift into unbalance as time goes on.

Even in more limited electric power systems, the same problems may arise. For instance *Gaffney* [3] reports problems with effective phase balancing in electric power systems in the tactical battlefield environment, largely because of insufficient operator training and experience. *David* [4] [5] proposes automatic phase balancing but does not propose an algorithm for this purpose.

The variables in the phase balancing problem are the phases each load is connected to and the goal is to minimize the degree of unbalance on feeders. Many algorithms have been used to solve phase balancing problem. The original work is *Zhu*, *Chow* and *Zhang*'s mixed-integer programming in 1998 [1], but this algorithm has a drawback that the objective functions can only be linear. In 1999, to expand to nonlinear objective functions, *Zhu*, *Bilbro* and *Chow* introduced simulated annealing [2]. In 2000 and 2004, *Chen* and *Cherng* and *Gandomkar* applied a genetic algorithm to the problem [6] [7]. In 2005, *Lin*, *Chen*, et. al adopted a heuristic greedy algorithm [8]. In 2007, *Huang*, *Chen*, *Lin*, et. al used an immune algorithm to solve the problem [9]. These heuristic algorithms can get near-optimal solution quickly but can not guarantee optimal solutions.

Many combinatorial optimization problems have no known efficient algorithms capable of always producing optimal solutions. For those problems that computer scientists have been shown to be NP-complete, there is convincing evidence that no correct, efficient algorithms can exist. An efficient algorithm for any one of the hundreds of known NP-complete problems would imply efficient algorithms for all of them, implying that all are equally hard to compute.

The phase balancing problem we describe in this paper can readily be shown to be equivalent to integer partitioning, a well-known NP-complete problem. Thus an efficient algorithm for phase balancing which always produced optimal solutions would imply efficient algorithms for all problems in NP, which computer scientists considered extremely unlikely. However heuristic algorithms that produce near optimal solutions with reasonable efficiency are possible, and are often developed for this purpose. [10]

In 2013, we introduced a dynamic programming algorithm to obtain the optimal solution for phase balancing problem in a reasonable running time [11]. However, this algorithm has a shortcoming in that it only balances the whole feeder, but not every section along the feeder. This may lead to a situation where the three phase current is balanced at the beginning of the feeders, but not balanced at other positions of the feeders.

In this paper, a dynamic programming algorithm is applied to solve the phase balancing problem along each part of the feeder. The computation

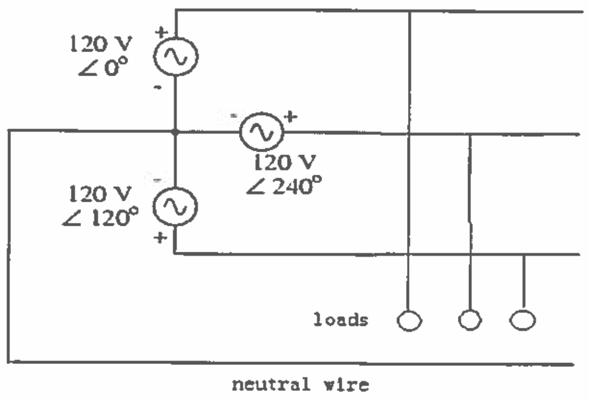


Figure 1: Three phase wiring diagram

complexity of this algorithm is $O(nT^2)$ (T is the sum of all the loads) in the worst case, and all operations we do are in linear time. A mathematical model, algorithm and objective functions are introduced (section 2). An optimal dynamic programming algorithm is discussed in detail (section 3). Simulation results appear in section 4. The conclusion is in section 5.

The key lesson of our paper is that heuristic techniques such as simulated annealing and genetic algorithm (and DE and PSO) can be replaced by the dynamic programming methods we employ, which give optimal results instead of heuristic ones. Our generalized dynamic programming algorithm gives optimal results in reasonable running time.

2. Problem and Algorithm Formulation

2.1. Overview

The algorithm we discuss in this paper is a dynamic programming algorithm. Assume the feeder is linear and the generation input is at the left. In terms of an objective function we seek to minimize a weighted sum of the degree of imbalance of each section along the feeder for a given number of tap changes. Suppose we have N loads on a linear feeder. To do this we create N objective function matrices (one for each section) as well as N cost matrices. For each matrix in both sets the (horizontal) rows correspond to potential load on phase A and the (vertical) columns correspond to the potential load on phase B. Thus the (i,j)th entry of the kth objective function matrix is the objective function value with partial total load i for phase A for the first k sections and with partial total load j for phase B for the first k sections. Implicitly the partial total load on phase C is the total load on the first k sections minus load i and minus load j for the first k sections.

Also, the (i,j)th entry of the kth cost matrix is the minimum number of tap changes one has to use to achieve the corresponding objective function in the (i,j)th entry of the kth objective function matrix. How does one compute the (i,j)th entry in the kth objective function matrix? One knows the load on phase A is i for the first k sections and the load on phase B is j for the first k sections. Implicitly one then knows the load on phase C is the total load for the first k sections minus i and minus j. So the (i, j)th entry is the absolute difference between the maximum of the loads on each phase minus the average load per phase for the first k sections and this difference is divided by the average load per phase for the first k sections. On the other hand, the (i, j)th entry of the kth cost matrix (which is the minimum number of tap changes to achieve the corresponding objective function value) is generated by a recursion that appears below. Note there are different recursions depending on whether loads are connected to one phase or two/three phases.

Once all of the matrices are generated, what is essentially a shortest path algorithm can be run from matrix to matrix where the distances are the objective function entry values. However in generating the possible paths thru the matrices there are some constraints on which entries in the $(k+1)$st matrix an entry in the kth matrix can be connected to. For instance if one is at entry (4,5) with a load of 1 on phase C in the 3rd objective function matrix, a path can connect it to entry (6,5) with a load of 1 on phase C in the 4th matrix if the 4th load is 2 (single phase load) but a path cannot connect it to (8,5) with a load of one on phase C in the 4th matrix.

Thus, unlike the *Dijkstra* shortest path algorithm we generate all possible feasible paths. However these constraints reduce the number of paths to be considered. Actually there is one more set of matrices that is generated as the recursion is run to record the associated paths for future use. This is done in k path matrices, where using similar definitions of i and j as before, the (i, j)th entry of the kth matrix is a pointer to the position of its parent entry along a path in the $k - 1$st objective function matrix.

To obtain a solution, one fixes as an input parameter to the algorithm the maximum number of tap changes allowed. Call it M. The last cost matrix is used to determine the set of solutions that meet this constraint. From the remaining solutions we can select the solution with the best objective function value. Once the best solution is selected one can retrieve the phase assignment from the corresponding path matrix.

Alternately we can create a table of the best solution for each specific number of tap changes. To create the pth row in the table one can run the steps of the previous paragraph, keeping only the solutions with p tap changes. Note that beyond a certain number of tap changes the objective function value of solutions tend to get worse.

This overview has been phrased in terms of minimizing a weighted sum of the degree of imbalance on each section. However the use of other objective functions using these techniques is certainly possible. A different one is discussed below. Note that the objective function and cost matrices may be viewed as N sets of two dimensional matrices, or each one as a three dimensional matrix. Another note is that an implementation of this dynamic programming algorithm can do without the objective function matrices since the objective function values can be computed from corresponding indexes of the cost matrices.

Another idea to save memory is to use a different notation to describe the statuses. Let L(i) be the load on each phase after the first i loads on the line, then We can change our state space from:

$C(i, j, k)$ – the cost to achieve a total load of i on phase a and total load of j on phase b, after the first k loads, leaving the total load on phase c implicit.

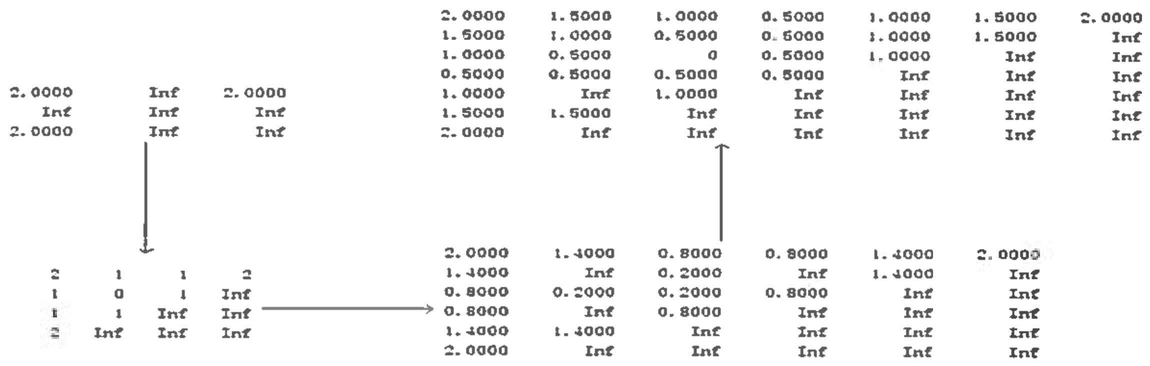
to

$C(da, db, i)$ – the cost to achieve a difference (delta) load on phase a from $L(i)$, a delta load on phase b from $L(i)$ after the first i loads, leaving the delta load on phase c implicit.

The advantage is that this method can reduce the complexity and also we

Table 1: An example feeder with single phase loads

Load size:	2	1	2	1
Phase:	A	B	C	A



```
                                              2.0000   1.5000   1.0000   0.5000   1.0000   1.5000   2.0000
                                              1.5000   1.0000   0.5000   0.5000   1.0000   1.5000      Inf
                                              1.0000   0.5000        0   0.5000   1.0000      Inf      Inf
                                              0.5000   0.5000   0.5000   0.5000      Inf      Inf      Inf
2.0000        Inf    2.0000                   1.0000      Inf   1.0000      Inf      Inf      Inf      Inf
   Inf        Inf       Inf                   1.5000   1.5000      Inf      Inf      Inf      Inf      Inf
2.0000        Inf       Inf                   2.0000      Inf      Inf      Inf      Inf      Inf      Inf

                                              2.0000   1.4000   0.8000   0.8000   1.4000   2.0000
     2     1     1     2                      1.4000      Inf   0.2000      Inf   1.4000      Inf
     1     0     1   Inf                      0.8000   0.2000   0.2000   0.8000      Inf      Inf
     1     1   Inf   Inf                      0.8000      Inf   0.8000      Inf      Inf      Inf
     2   Inf   Inf   Inf                      1.4000   1.4000      Inf      Inf      Inf      Inf
                                              2.0000      Inf      Inf      Inf      Inf      Inf
```

Figure 2: Example of objective function values matrices

can delete the solutions with very large deltas. So one can save a great deal of memory, which otherwise would limit the scalability of this algorithm.

Figure 2 and figure 3 are examples of objective function values matrices and cost function matrices of a feeder with four single phase loads.

The dynamic programming algorithm is now outlined in more detail.

2.2. Objectives

In phase balancing problem, there are three main objectives:

1. To avoid overloading.

2. To balance three phase current along the feeders.

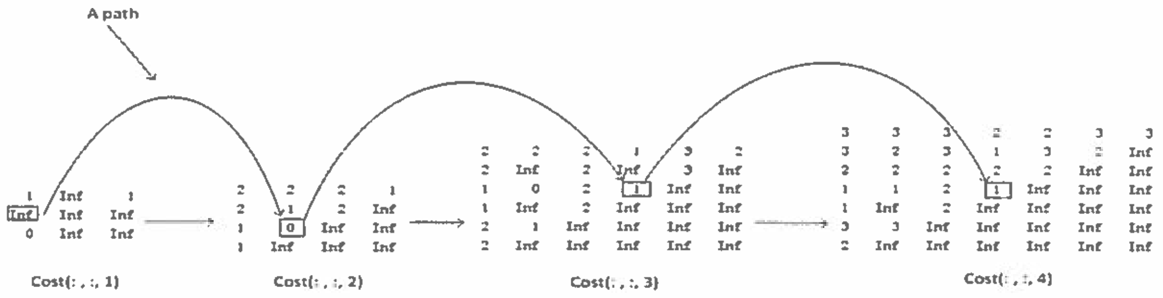


Figure 3: Example of cost matrices

3. Reduce number of phase changes to save labor cost.

2.3. *Overall structure*

Figure 4 illustrates the overall structure for phase balancing of distribution feeders. One abstracts the node connection and hourly load demand for each node from feeder topology information and customer information. Based on this information, one can do a phase balancing analysis and give a phase assignment recommendation.

2.4. *Sample feeder with connecting branches*

Figure 5 shows a radial feeder configuration and the loads ($L_{i,1}$, $L_{i,2}$, $L_{i,3}$) at node i. In our model, the feeder is divided into nodes and sections. Here, $I_{i,j}$ denotes the current on phase j of section i. $L_{i,j}$ is the current (load) demand of node i on phase j. The phase balancing's objective is to find the

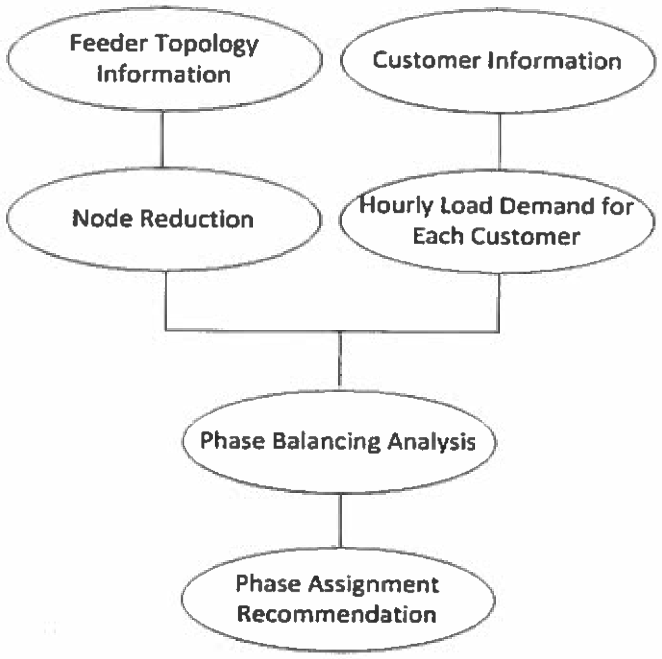

Feeder Topology Information
Customer Information
Node Reduction
Hourly Load Demand for Each Customer
Phase Balancing Analysis
Phase Assignment Recommendation

Figure 4: Overall structure for phase balancing

optimal phase assignment for each load to minimize the unbalanced flows at monitored sections with a certain number of tap changes which is smaller than the given maximum one.

2.5. Objective function

To balance the three phase flows along the whole feeder, one needs to balance the three phase flow in each section. From Kirchhoff's current law, one knows that the current on phase ϕ flowing out of section j equals to the current on phase ϕ from the source minus the total current on phase ϕ of the

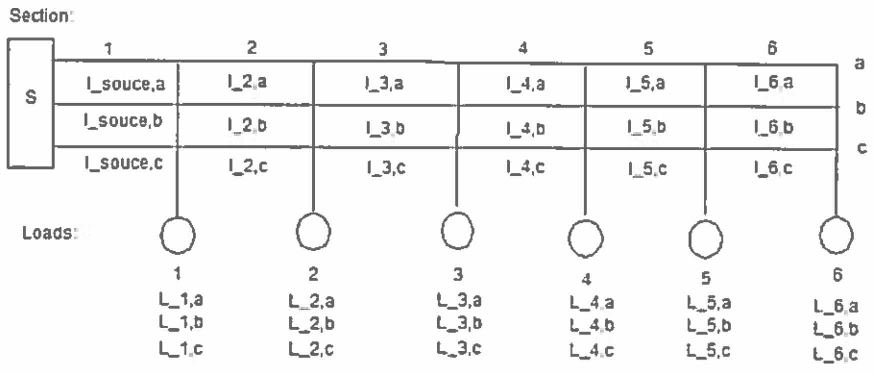


Figure 5: Sample feeder model

first $j-1$ sections.

$I_{j,\phi} = I_{source,\phi} - \sum_{i=1}^{j-1} L_{i,\phi}$ for all ϕ=a,b,c.

Here, j is the section index and i is the load index. $j \geq 2$.

Then the problem becomes to balance $\sum_{i=1}^{k} L_{i,a}$, $\sum_{i=1}^{k} L_{i,b}$ and $\sum_{i=1}^{k} L_{i,c}$ for $k = 1$ to N.

There are various kinds of objective functions such as cost functions in [2] and the loss function in [6]. In this paper, the objective function is the phasing unbalance index (PUI) which is used in many phase balancing papers [9] [8] [12]:

$$PUI_i = \frac{Max(|I_{a,i} - I_{avg_i}|, |I_{b,i} - I_{avg_i}|, |I_{c,i} - I_{avg_i}|)}{I_{avg_i}} * 100\% \tag{1}$$

Here, $I_{a,i}$, $I_{b,i}$ and $I_{c,i}$ are the total current loads on phase 1, 2 and 3 of

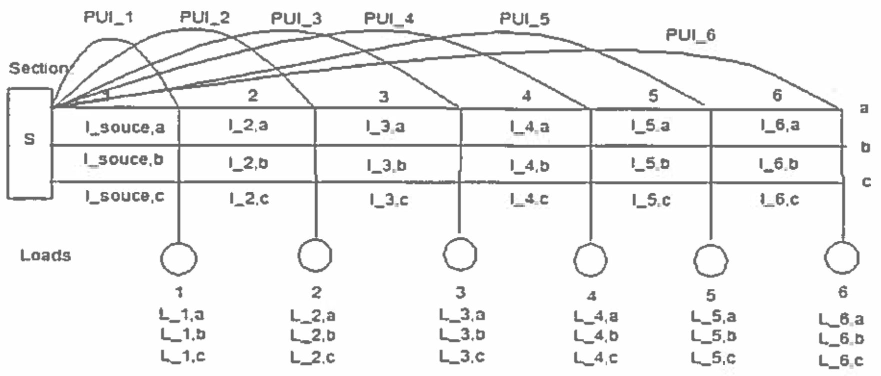


Figure 6: Phasing unbalance index

section i. I_{avg_i} is the mean value of the current load on each single phase of section i. Considering the single phase loads case is a subset of the three phase loads case, we assume that all the loads are connected to three phases. The load range is set as integers between 1 and 100. Larger loads range can be scaled to this range.

Also, to avoid overloading, the current on each phase has to be smaller than the line capacity.

In this paper, two approaches are introduced.

The first approach is to limit all the PUI's of each section under a certain threshold:

$$PUI_i <= threshold \qquad \text{for all i = 1 to N} \tag{2}$$

Here, the threshold can be set by operator or one can use binary search to find the minimum possible threshold.

The second approach is to minimize the weighted sum of phase unbalance indexes for each section of the feeder:

$$\text{Minimize} \sum_{i=1}^{i=N} w_i * PUI_i \tag{3}$$

Subject to:

$$w_i = \sum I_{\phi,i} \tag{4}$$

$$I_{\phi,i} \leq C_i \tag{5}$$

where

ϕ	is one of three phases a, b and c.
i	is the index of section from 1 to N.
C_i	is the phase line capacity of phase j of section i.

2.6. Load type

There are two types of loads on the feeder: one phase loads and two or three phase loads. Loads on one phase feeders can only connect to one of the three phases. Loads on the two and three phase feeders can connect to two or three phases. That is: single phase loads have three tap change possibilities, two and three phase loads have six tap change possibilities.

Table 1 shows valid connection schemes for various types of laterals.

Original phase		Valid rephasing schemes		
3ϕ	abc	abc	acb	
		bca	bac	
		cab	cba	
2ϕ	ab	ab*	ba*	a*b
		b*a	*ab	*ba
	bc	bc*	cb*	b*c
		c*b	*bc	*cb
	ac	ac*	ca*	a*c
		c*a	*ac	*ca
1ϕ	a	a**	*a*	**a
	b	b**	*b*	**b
	c	c**	*c*	**c

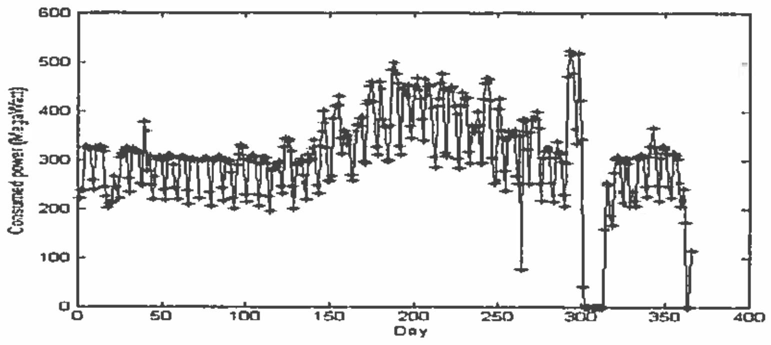


Figure 7: A yearly load profile

2.7. Load pattern

The customer hourly load data can be collected by the AMI (Advanced Metering Infrastructure) meters. However, it would be hard to do the computation if one makes phase balancing recommendation for the next year based on all hourly load data of the last year because of the large amount of data. So the evaluation of the load pattern is considered. One can do phase balancing analysis based on the load in the "peak time" or the "peak day" in the summer. Figure. 7 and 8 shows the load pattern obtained from one of $LIPA$'s substation's data. From the two load profiles, one can see the peak days of the year are in the summer and autumn and the peak hours of the day are in the afternoon.

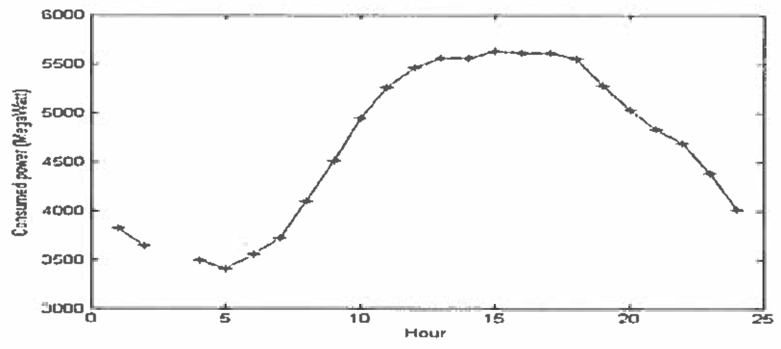


Figure 8: A daily load profile

3. A Dynamic Programming Algorithm to Solve the Phase Balancing Problem

3.1. The optimality and complexity

An "optimal" algorithm for phase balancing is now presented in detail. The phase balancing problem is NP-complete even with two phases and no cost per tap change, because it is equivalent to the integer partition problem and the integer partition problem is NP-complete. The hardness of integer partition depends upon large numbers, because it is not strongly NP-complete. For the phase balancing problem, the loads range between 1 and several hundred amperes. Assume that there are n loads, where the ith load has values (weights) $l_{i,a}$, $l_{i,b}$ and $l_{i,c}$ on three phases and are currently assigned to a feeder. We assume the weights of all loads are integers, and the total load $T = \sum_{i=1}^{n} \sum_{\phi=a}^{c} l_{i,\phi}$. As will be seen the algorithm runs faster with smaller T. Loads can be scaled to bring this about. The solution produced by the dynamic programming algorithm are optimal but it should be noted that the scaling is a source of approximation.

We present an algorithm which runs in $O(nT^2)$ to find the minimum number of changes to reach a particular quality criteria.

Denote the total load on phase i by L_i. Because there are three phases, there are about T^2 sets of possible values for L_1, L_2, and L_3. This is as both L_1 and L_2 are integers between 0 and T, and $L_3 = T - L_1 - L_2$, L_3 would be specified after one has L_1 and L_2.

The algorithm will enumerate all possible partitions of T into L_1, L_2, and L_3, and in particular for each such partition P find way to move from the current state to P using the fewest number of changes. One can evaluate each of these $O(T^2)$ partitions according to the objective function, eliminate all which are not good enough, and then find the minimum cost good-enough transformation.

3.2. Step1: Use recurrence to record number of tap changes

Define $C[x, y, i]$ to be the minimum cost (in terms of number of moves) to realize a balance of $L_1 = x, L_2 = y$ and implicitly $L_3 = T - L_1 - L_2$ after reassignments to the first i loads (from 1 to i).

For the cost matrix for single phase loads we define the following recurrence relation:

$$C[x,y,i] = Min[C[x-l_i,y,i-1]+t(i,1), C[x+l_i,y-l_i,i-1]+t(i,2), C[x+l_i,y,i-1]+t(i,3)] \tag{6}$$

Here in the cost matrix, l_i is the weight of ith load (single phase load), $t(i,\phi)$ is the cost of moving the i_{th} load to phase ϕ. $C[x,y,i]$ is the minimum number of tap changes to move from the initial status to $[x,y,T_i-x-y]$.

$$T_i = \sum_{j=1}^{i} L_j \tag{7}$$

If i_{th} load stays on phase ϕ

$$t(i,\phi) = 0 \tag{8}$$

If i_{th} load leaves phase ϕ

$$t(i,\phi) = 1 \tag{9}$$

Assume the ith load is initially on phase a. Then the optimal solution either leaves load i on phase a (incurring no cost for the move), or moves it to phase b, or moves it to phase c (both of which incur a cost of 1 operation).

We need similar recurrences for the cases where load i is on phase b or phase c. The basis of this recurrence is that $C[L_1, L_2, 0] = 0$, $C[x_0, y_0, 0] = \infty$ for all $x_0 \neq L_1$ and $y_0 \neq L_2$ (meaning no other states are achievable with zero moves).

For two and three phase loads, suppose the ith load has three single phase loads $l_{i,a}, l_{i,b}, l_{i,c}$ and they are initially on phase a, b and c. We define the following recurrence relation:

$$\begin{aligned} C[x,y,i] = Min[&c[x - l_{i,a}, y - l_{i,b}, i-1], c[x - l_{i,a}, y - l_{i,c}, i-1] + 1, \\ &c[x - l_{i,b}, y - l_{i,a}, i-1] + 1, c[x - l_{i,b}, y - l_{i,c}, i-1] + 1, \\ &c[x - l_{i,c}, y - l_{i,a}, i-1] + 1, c[x - l_{i,c}, y - l_{i,b}, i-1] + 1] \end{aligned} \tag{10}$$

3.3. Step 2: Record the "Path"

In last subsection, when calculating the number of tap changes for each $[x, y, i]$, one needs to create a three dimensional path matrix (since C is three dimensional) to record what is the "parent" of a status $[x, y, i]$. That is, to record the parent's position of $[x, y, i]$ as a cell. With all the record of these relationships, one can know the paths.

3.4. Step 3: Calculate objective values

After using the recurrence to record the path, one calculates objective values for all $(T_i + 1)^2$ possible $[x, y, i]$ using objective function for all $i \in [1, n-1]$. Then, to take the imbalance of each section into consideration, one can calculate the weighted sum objective values for each path.

3.5. Step 4: Avoid the overload

One needs to delete the solutions that cause overload on the feeders by setting the positions that have indexes larger than the line capacity to infinity. The deletion simply removes the incoming or outgoing edges to these nodes. That is to make sure that all three phase currents in each section is smaller or equal to the line capacity.

3.6. Step 5: Make phase assignment recommendation

For the first approach, if we have a threshold of "what is balanced e-nough", then we can delete any partial solution that is not "balanced e-nough". If there is a solution remaining, it would be found by any path from an end state to a state that passes through "balanced enough" vertices. If we do not have a threshold but want to find the path with the minimum balance, do a binary search on the possible thresholds. Repeatedly pick a possible threshold in the middle of the range of possible thresholds. Delete all vertices more unbalanced than this. Look for a path in the remaining graph. If we find one, try a smaller threshold. If not, try a larger one.

For the second approach, consider now that one has three matrixes: the number of tap changes (cost) matrix $C[x, y, N]$, the path matrix and objective values matrix $Objv[x, y, N]$. Then one can make a table with N rows and three columns. The first column is the maximum number of tap changes allowed to make. The second column is the corresponding best objective value one can get, this can be obtained by searching all the x and y in $Objv[x, y, N]$ that satisfies $C[x, y, N] =$ maximum number of tap changes allowed. The third is the corresponding phase assignment for each load which can be obtained by retrieving the path. From this table, one can make phase assignment recommendation provided the desired number of tap changes or objective values.

3.7. An iterative method to balance tree network feeders

For tree network feeders, one can use the dynamic programming algorithm above to balance each subtree feeder and take every subtree feeders as equivalent nodes in the upper level of the tree. One can balance the whole system using this bottom-up method.

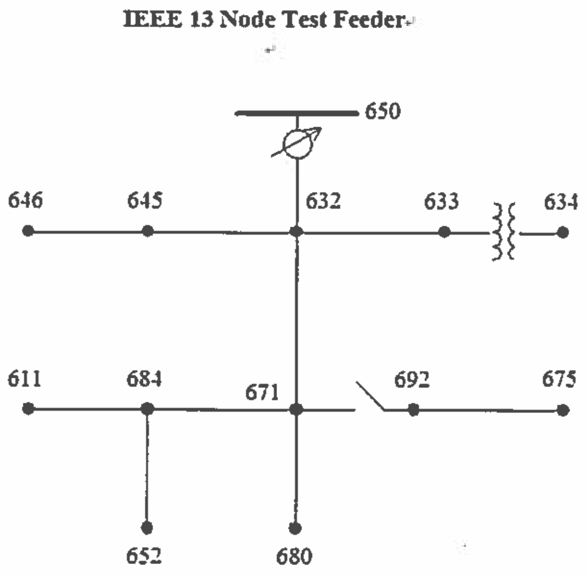


Figure 9: IEEE sample feeder with 13 nodes

For example, one can use the algorithm to balance three phase current of IEEE sample feeder (figure. 9). This feeder contains several branch feeders and they form a tree network. One can divide it into five groups: node 632, 645 and 646 as group A, node 633 and node 634 as group B, node 692 and node 675 as group C, node 611,684 and 652 as group D and node 671 and 680 as group E. One can first balance group A and B and take them as one nodes. Then balance group C, D and E and take them as the second node. At last, balance those two equivalent nodes.

Table 2: Feeder before phase balancing

Load index	1	2	3	4	5	6	7	8	9	10
a:	0	5	1	2	7	6	10	3	9	0
b:	0	2	7	0	0	0	0	6	0	2
c:	5	0	10	0	0	7	3	0	3	6
Total on a:	0	5	6	8	15	21	31	34	43	43
Total on b:	0	2	9	9	9	9	9	15	15	17
Total on c:	5	5	15	15	15	22	25	25	28	34

4. Simulation

4.1. Implementing a 20 node feeder

Here, a feeder with 10 randomly generated loads and phases is tested. Table 2 and Table 3 show the phase assignment for each load before and after phase balancing. Figure 10 and 11 show the three phase current for each section before and after phase balancing respectively. Figure 12 shows the corresponding objective values. Note that in the objective values at the end of the curves in figure 12 are worse because there is less flexibility in making tap changes at that point. In fact, no tap changes were made for the last two loads in the example.

Table 3: Feeder after phase balancing

Load index	1	2	3	4	5	6	7	8	9	10
a:	0	5	1	2	7	6	0	3	9	0
b:	0	2	10	0	0	0	10	6	0	2
c:	5	0	7	0	0	7	3	0	3	6
Total on a:	0	5	6	8	15	21	21	24	33	33
Total on b:	0	2	12	12	12	12	22	28	28	30
Total on c:	5	5	12	12	12	19	22	22	25	31

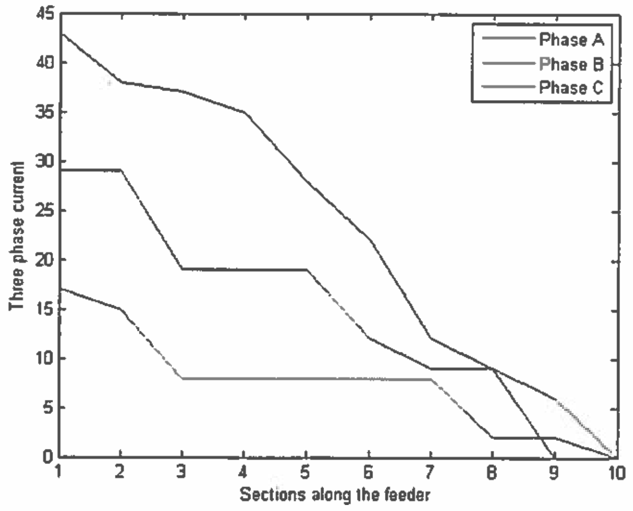


Figure 10: Three phase current along the feeder before phase balancing

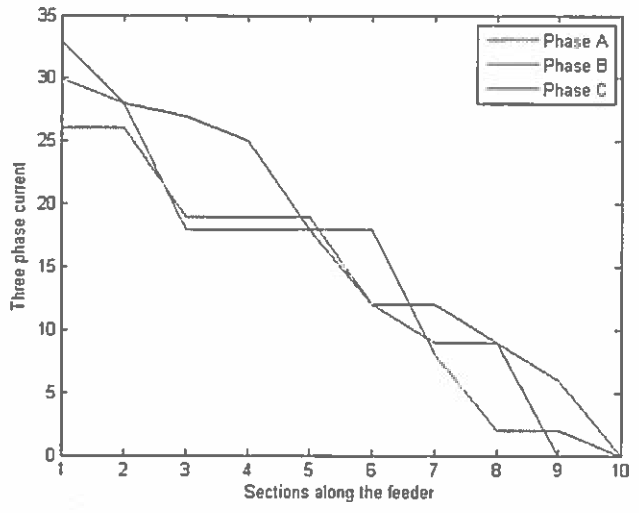


Figure 11: Three phase current along the feeder after phase balancing

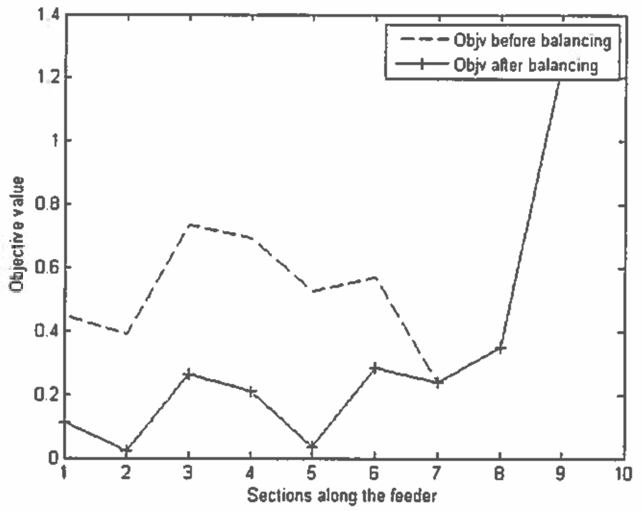


Figure 12: Objective values before and after phase balancing

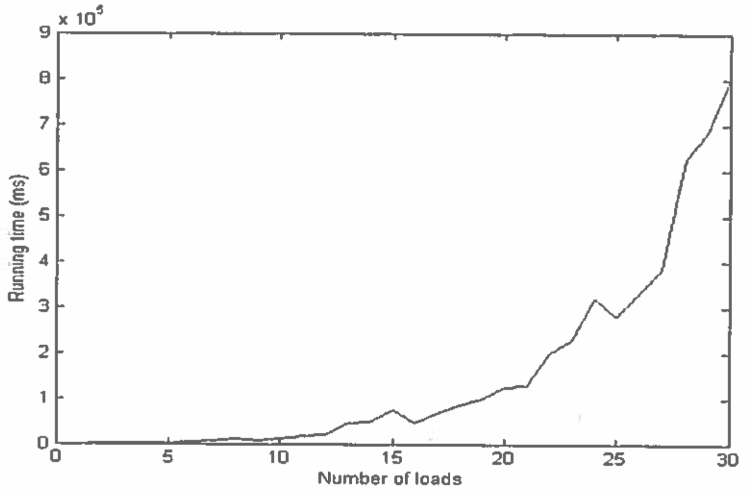


Figure 13: Running time VS Number of loads

4.2. Running time and required memory

To illustrate the running time and required memory (without the memory reducing trick mentioned in the overview), randomly generated loads and phases are used for testing. In figure 13 and 14, the horizontal axis is the number of loads and the vertical axes are running time in ms and allocated memory in *bytes*.

5. Conclusion

Of all the algorithms examined in our earlier work [11], dynamic programming was the most promising in its ability to provide optimal solutions

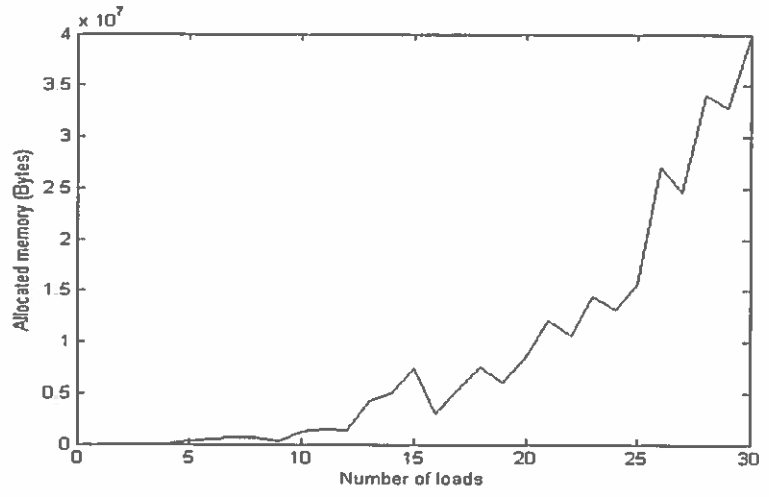


Figure 14: Running time VS Allocated memory

without using an exhaustive search approach. This paper has examined and discussed dynamic programming in much greater detail and also adapted it to include a consideration of spatially distributed loads. Many variations on the basic approach described here are possible. This includes the use of different objective functions and data structure implementation of the algorithm. Most significantly the use of dynamic programming allows a better quality combinatorial solution at much less the cost of an exhaustive search.

Acknowledgment

This material is based upon work supported by the Department of Energy under Award Number DE-OE0000220, NSF grants DBI-1060572 and IIS-1017181.

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, pro process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

This report was prepared as an account of work performed by The Research Foundation of *SUNY* as sub-recipient of any award made by an agency of the United States Government to the Long Island Power Authority.

References

[1] J. Zhu, MY. Chow and F. Zhang. IEEE Transaction on Power Systems, " Phase balancing using Mixed-Integer Programming". Vol. 13, No. 4, November 1998, pp. 1487-1492.

[2] J. Zhu, G. Bilbro and M. Chow. IEEE Transactions on Power Systems,

"Phase Balancing using Simulated Annealing". Vol. 14, No. 4, November 1999, pp. 1508-1513.

[3] M. N. Gaffney. http://www.ndia-mich.org/workshop/Papers, "Intelligent Power Management: Improving Power Distribution in the Field".

[4] Y. David and R. Hasharon. United States Patent. "Apparatus for and method of evenly distributing an electrical load across an N-phase power distribution network". US Patent Num. 6018203. Jan. 25. 2000.

[5] Y. David. et al. United States Patent. "Apparatus for and method of evenly distributing an electrical load across a three phase power distribution network". US Patent Num. 5604385. Feb. 18. 1997.

[6] M. Gandomkar. 39th International Universities Power Engineering Conference, "Phase Balancing Using Genetic Algorithm". Sept, 2004, pp. 377-379.

[7] T. H Chen and J. T. Cherng. IEEE Transactions on Power Systems, "Optimal Phase Arrangement of distribution Transformers Connected to a Primary Feeder for System Unbalance Improvement and Loss Reduction Using a Genetic Algorithm". Vol. 15, NO. 3, August 2000, pp 994-1000.

[8] Chia-Hung Lin, Chao-Shun Chen, Hui-Jen Chuang and Cheng-Yu Ho. IEEE Transactions on Power Systems, "Heuristic rule-based phase balancing of distribution systems by considering customer load patterns". VOL. 20, NO. 2, May 2005. pp 709-716.

[9] M.-Y. Huang, C.-S. Chen, C.-H. Lin, M.-S. Kang, H.-J. Chuang and C.-W. Huang. IET Generation, Transmission and Distribution, "Three-phase balancing of distribution feeders using immune algorithm". 17th August 2007, pp. 383-392.

[10] Steven Skiena. "The Algorithm Design Manual" 2nd edition. Springer, 2008.

[11] K. Wang, S. Skiena and T.G. Robertazzi, Electric Power System Research, "Phase Balancing Algorithms", Volume 96, March 2013, Pages 218224.

[12] Nikhil Gupta, Anil Swarnkar and K. R. Niazi. Power and Energy Society General Meeting, 2011 IEEE. "A novel strategy for phase balancing in three phase four wire distribution systems". July 2011.