12-2013

Algorithms for the Implementation of a Dynamic Energy Pricing Policy

Manek Biswas

Abstract

In the upcoming smart grid era, dynamic pricing using primarily data collected from smart meters installed in homes and businesses is a virtual certainty. The possibilities this feature may offer are many, and we examine in this study how subunits of the electricity grid may implement load scheduling given a model for predicting ever-changing energy prices over a day. We present a realistic linear model that utilities may use for implementing dynamic pricing, and argue how accurate prediction of this model by its customers may be profitable both for the utilities and their customers.

The thesis provides algorithms for achieving balancing using this pricing model. Using the total demand on the grid for a 24-hour period, we provide a dynamic programming algorithm and a greedy algorithm that find the partitioning of the total demand into intervals which will minimize the total cost. The output of the above algorithms can be seen as providing us with the global ideal load curve. Using this, each subunit of the grid creates its own ideal load curve, which is the scaled version of the global load curve.

The issue then for each grid subunit is to find a schedule for its flexible loads so that its load profile is as close to its ideal load curve as possible. We have proposed a number of algorithms for this purpose using different search heuristics. One needs to resort to heuristics as the optimization problem can be shown to be NP-hard. The search heuristics that we have considered are: a Greedy algorithm to traverse through our search space of possible schedules, a randomized Greedy algorithm with random restarts to explore the search space further, the Metropolis algorithm, the Tabu search, and finally, a randomized Tabu search with random restarts.

Algorithms for the Implementation of a Dynamic Energy Pricing Policy

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

Manek Biswas
B.E., Manipal University, 2011

Syracuse University

December 2013

Acknowledgements

I am deeply grateful to my advisor, Professor Chilukuri Mohan, for his continuous guidance on this work- it would not have been possible to complete it without him. I am thankful to Dr. Tomislav Bujanovic, for always having words of encouragement and suggestions whenever I needed them. I am grateful to Professor Prasanta Ghosh and Professor Kishan Mehrotra for being part of this work. The work reported in this thesis has been under the financial support of the U.S. Department of Energy grant to Syracuse University for Smart Grid-related work. As such, I would like to express my gratitude to the U.S. Department of Energy.

A huge thanks to my parents- Somenath and Kiran. Your gentle prodding was invaluable in getting this work done.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

1.1. Need for Load Balancing

It has been observed that typical energy usage by utility customers varies very predictably and sharply over a day. The figure below shows typical load profiles observed in a residential unit in Calgary, Canada over a 24 hour period[13]:

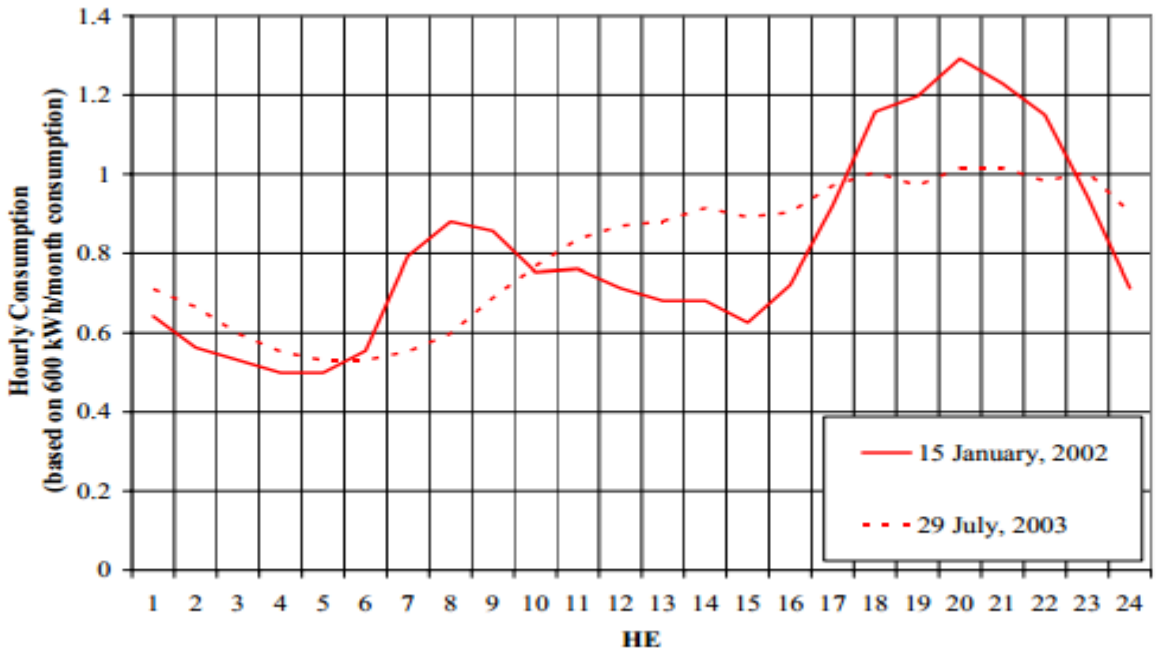


Figure 1: Residential Load Profile for Calgary, Canada

As is evident from the load over a day in summertime and wintertime, there are sharp peaks and valleys in the curve. On particularly hot or cold days, with heating or cooling being used maximally, electricity usage spikes even more sharply.

The consequences for the utility are twofold. Since supply must always equal demand in the case of the electricity grid, having to constantly change energy production according to the day's fluctuations in demand raises challenges for the utility. Additionally, utilities also need to have special "peaker" power plants which supply energy when usage is abnormally high. Although the need for peaker plants arises only a few times for a short duration in any year, building and maintaining infrastructure for the peaker plants is a massive investment for the utility. Furthermore, since peaker plants are rarely used, their chances of failure are relatively high. This leads to blackouts at times of particularly high energy demands.

A natural solution to the problem is to do away with peaks and valleys in load curves by having an almost flat, averaged-out demand. Such demand can be supported by base-load power plants with minimal intervention for generating excess power or for removing energy sources (when demand is low). Load shedding may accomplish the former by eliminating excess demand completely, but does not address the problem of points of low demand, and is an inconvenience to utility customers. Load balancing, however, offers a viable alternative. Traditionally, load balancing meant storing excess power at times of low demand to be released when there is a demand of more power. This often involves the use of batteries, which is, with existing technology, not a low-cost or particularly effective solution, since battery storage requires huge investments and frequent replacements. With the advent of the smart grid, however, there is a possibility of the concept of a 'demand response' approach, with the installation of smart meters at the customer locations.

1.2. The Smart Grid

“"Smart Grid" consists of devices connected to transmission and distribution lines that allow utilities and customers to receive digital information from and communicate with the grid. These devices allow a utility to find out where an outage or other problem occurs on the line and sometimes even fix the problem by sending digital instructions. Smart devices in the home, office, or factory inform customers of times when an appliance is using relatively high-cost energy and allow customers to remotely adjust their settings.

Smart devices make a Smart Grid as they help utilities reduce line losses, detect and fix problems faster, and help customers conserve energy, especially at times when demand reaches significantly high levels or an energy demand reduction is needed to support system reliability.”

- U.S. Energy Information Administration

The latter part of the above description of the Smart Grid, provided by the U.S. Energy Information Administration, states that the Smart Grid may be capable of helping customers save energy at high demand-levels. While this leads to a marked decrease in the customers’ energy bill, it also benefits the utility by minimizing or doing away with infrastructure needed to support these critical peak points in the form of peaker plants as mentioned earlier.

Implementing such a scheme requires customers to be aware of high demand-levels in the first place. This information can be provided to them through the use of “Smart Meters”, which are an advancement of traditional electricity meters installed at customer points. The earlier traditional model of the electricity meter involved the utility reading customer usages, and thus supported a one way communication between the customer and the utility. Smart meters, on the other hand, work by providing a two-way communication in place of the one-way communication, by

allowing the customer to be constantly aware of real-time energy prices. The proposed time for the customer to read changes in energy price is an hour or less, according to the FERC Report on Demand Response & Advanced Metering. [1]

Smart appliances are appliances that may be controlled through a central controller installed at the customer side. A customer with the smart meter, a controller, and smart appliances may schedule her requirements or loads in a manner which aims to reduce her energy consumption, and thus lower her energy bill.

1.3. Problem Considered In the Thesis

As mentioned earlier, load balancing has many benefits for the electricity grid. The problem to be addressed is how to go about load scheduling in a manner that is beneficial for both the utility and the customer. It has been proposed to make the load curve as flat as possible, thereby eliminating peaks in energy demand, and to have the demand at any point of the day be close to the average of total demand for that day. While this proposal does indeed eliminate peak points in a load profile, it assumes incorrectly that generation costs for a unit of energy are constant over a day.

Renewable energy accounted for 12.4% of total domestically-produced generation in the US in 2012, and the renewable energy target hopes to see that figure increase to 20% by the end of the decade in 2020. [2] While the use of renewable energy provides huge gains in pollution control and (ideally) free sources of energy, there is an associated problem of unpredictability in supply. Solar photovoltaic (PV) energy and wind energy are the primary sources of renewable energy, and energy generated using PVs and wind can drop to zero momentarily and rapidly. The use of

energy storage devices has been proposed to overcome this unpredictability, wherein surplus energy produced may be stored in storage devices for use at times when the current demand exceeds all possible supply. However, batteries and other forms of energy storage are, as of the technology available today, expensive and unable to store significant amounts of energy that may be needed to compensate for minimal generation capacity at times of poor irradiation (for PVs) and a lull in wind (for wind energy). To consider a flat load profile to be ideal would be to assume that energy supply cost tends to be uniform throughout the day. This may be almost correct under current circumstances with prevalent use of non-renewable energy sources and renewable energy penetration being only around 10%, but that assumption is unreasonable in the near future, with a much higher percentage of renewable energy sources in the energy mix.

This thesis presents an energy pricing mechanism for the utility which can take into consideration the cost incurred to produce energy over a day. Chapter 2 explores the need for an appropriate energy pricing mechanism in the Smart Grid era, and why it will likely be linear in terms of demand. In subsequent chapters, we explore how the pricing model can be used in conjunction with total predicted load to form an ideal load profile for the customers. Furthermore, given the ideal load profile, we discuss possibilities for algorithms which may enable customers to optimally schedule out their loads or demands so as to conform to the ideal load profile. If followed, the proposal has the potential of massive monetary gains for both utilities and customers, as well as allowing for the provision to successfully incorporate greener, pollution-free methods of energy generation by using renewable energy sources.

The next chapter examines why an appropriate energy pricing mechanism must be defined and implemented for lowering energy production costs and the subsequent reduction in energy bills for utility customers, as can be a potential benefit of the Smart Grid.

Chapter 2: Need for an Appropriate Energy Pricing Mechanism

2.1. Introduction

As the world is increasingly making use of more appliances within households and as new industries and commercial establishments are being set up, the demand of electricity is constantly increasing. Matching demand with supply has therefore been a central issue in the design and operation of electrical networks. If the demand is not moderated in an appropriate manner then peaking in far excess of the average is inevitably seen at multiple intervals in a twenty-four hour period.

The conventional solution to take care of such peaking points has been to increase energy generation capacity and/or buying shortfall amounts from other networks. Capacity increases require the addition of substantial infrastructure which will lie idle in the non-peak periods. Also, as fuel consumption for most conventional generators has a quadratic dependence on output power, the amount of pollution will be far more in case of uneven demand over a day than when the demand is relatively uniform. Also, if we would like to encourage the usage of green energy sources, as is a critical aspect in the nation's energy policy, we need to shift demand to periods when such greener sources of energy will be available.

These considerations require us to rely on appropriate ***demand response*** (DR) programs to encourage users to shift their loads away from peak times.[3] It has been noted that "Reduced peak demand lowers electricity bills and benefits utilities by reducing complexity of grid

stability, occurrences of equipment failures, and blackouts. It also enables utilities to comply with government mandates to cut peak demand" [4],[5]

It is clear that DR programs need to provide appropriate economic incentives to encourage users to shift their demand away from the peak periods. However, a simple-minded approach of having substantial difference in tariff between peak periods and low-demand periods will not solve the problem, as it will merely shift the peak rather than ensuring relatively uniform demand over a twenty-four hour period. This has been established in a simulation study. [6]

Instead, as economists have argued, appropriate ***real time pricing*** (RTP) tariffs are needed, as discussed in the next section.

2.2. Real Time Pricing (RTP)

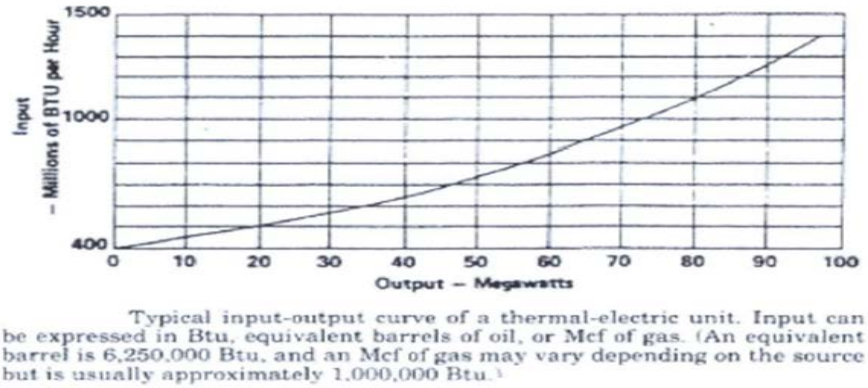
Conventional electricity tariffs reflect average supply cost over a long period of time and are fixed for months. In contrast, in RTP as practiced now, customers are charged prices that can vary over very short time intervals to reflect “contemporaneous marginal supply costs”. The supplier is not required to quote the price for such short intervals even at the start of the intervals, it may simply provide the users with recent past tariffs data. RTP achieves twin benefits: it will result in a relatively flat demand curve, and will also provide the customers with substantial reduction in their electricity costs. For these reasons, policymakers have identified RTP as a potential strategy for developing demand response (DR). [7]

RTP tariffs were first introduced in the mid-1980’s. In 2004, more than seventy utilities in the U.S. offered voluntary RTP tariffs. However, in spite of the potential benefits, RTPs have not

been popular. There are two reasons; the first is that on the demand side what was expected often was manual response to dynamically changing tariffs. It is necessary to alleviate that issue to automate responses to changing tariffs with appropriate technology. Second, as of now, RTP appears to be too volatile as there is no rational explanation offered for RTP tariffs except for declaring that these represent the current market rate. What is needed is to base RTP tariff on a sound economic principle, and allow utility customers to be aware of the rationale behind the energy bill they pay based on an RTP tariff. In our proposal of a new pricing policy, we take care of the second aspect, and also show how automated decision making can be carried out at the demand side by providing appropriate scheduling algorithms.

2.3. Electricity Production Cost

It has been long accepted that electricity generation cost is a monotonically increasing quadratic function of the output power, especially for conventional generators that use fossil fuel. The following figure from [8, page 69] clearly shows the quadratic nature of the function:



Typical input-output curve of a thermal-electric unit. Input can be expressed in Btu, equivalent barrels of oil, or Mcf of gas. (An equivalent barrel is 6,250,000 Btu, and an Mcf of gas may vary depending on the source but is usually approximately 1,000,000 Btu.)

Figure 2: Typical input-output curve of a thermal-electric unit

For a quantitative appreciation of the phenomenon, we reproduce the following (costing as per early 1970's) [8]:

Table 1: Fuel Amount, Fuel Cost, Charge for load increment, and Incremental Rate corresponding to Load

Load (MW)	Fuel Amount (Btu/hour)	Fuel Cost (Dollars/hour)	Charge for load increment (Dollars/hour)	Incremental Rate (Dollars/MWh)
0	200	700	-	-
20	300	1050	350	17.50
40	450	1575	525	26.30
60	650	2275	700	35.00
80	950	3320	1045	52.20
100	1500	5250	1930	96.50

Even for virtual power plants, it is standard to assume such a quadratic relationship. For example, [9] uses the following relationship: Given that *C* is the hourly cost in currency unit per

hour, E_e is the amount of generated active power in MW, and p, q, and r are coefficients that will depend on the technology used in electricity production, it is defined that

$$C(E_e) = p\,E_e^2 + q\,E_e + r$$

2.4. Marginal Cost Pricing Principle

Usually the price is taken to be the average revenue per unit of product. However, for public utilities there has been a strong case to set price to be the marginal cost. Let us suppose for producing x amount of a product, the cost incurred is $C(x)$. The marginal cost here is defined to be $C'(x)$, where $C'(x)$ is the function (in x) obtained by differentiating $C(x)$ with respect to the variable x. The marginal cost pricing principle is that at production level x, the price of the product is equated to the marginal cost at x, namely, $C'(x)$. We notice, due to the differentiation, any constant term in $C(x)$ will have no effect on the price function. Often, the constant term will reflect the amortized set-up cost. For public utilities, one can make a case that since such utilities are vital for the community, the set-up cost is not to be passed to the customer, but will be paid by the government through appropriate subsidy. We see this happening especially with respect to renewable energy infrastructure; in many countries the set-up cost as well as the production cost is subsidized by the respective governments due to their commitment to lessening pollution.

2.5. Our Pricing Proposal

Let a 24-hour period be divided into M (usually 24) equal intervals. As in Section 2.4 above, for the interval i, let the production cost $C_i(x)$ to produce x amount of electricity be given as

$$C_i(x) = p_i x^2 + q_i x + r_i$$

To this cost, we add a linear dx mark-up to reflect profit by utility in selling x units of electricity. Without loss of generality, we assume the profit depends only on amount of electricity sold, and not the interval.

Let $D_i(x)$ denote the marked-up cost to generate x amount of electricity in the i^{th} interval:

$$D_i(x) = p_i x^2 + n_i x + r_i \text{, where } n_i = q_i + d.$$

Using the marginal cost pricing principle, we obtain the price for the i^{th} interval to be:

$$a_i x + b_i \text{, where } a_i = 2p_i.$$

Thus, our pricing policy is to define the price to be a linear function of demand, taking into account the time of the day to reflect, for example, overtime cost for personnel, availability of alternate energy, etc., which can certainly differ from one interval to another. It is to be emphasized that the price depends on the total demand for electricity in the electrical network. It is clear that if the total demand over a 24-hour period is distributed in the intervals in such a way that the total cost of consumption is minimized, then in such a situation we shall not see high peaks in load profiles. Our proposal will later elaborate scheduling policies for individual customers designed to lead to global minimization of total cost over the electrical network. The

next chapter explores algorithms for optimal allocation of the total demand into intervals over a day, using the aforementioned pricing policy.

Chapter 3: Algorithms for Globally Optimal Allocation

3.1. Introduction

In the previous chapter we defined the electricity pricing policy that we explore in this study. In this policy, a twenty-four hour period is divided into M equal intervals. For each of these intervals i, we assume that the electricity price in that interval is $a_i.x_i + b_i$, where x_i is the *total* usage of electricity in the interval, and a_i and b_i are constants. Thus, the *total cost C*, as incurred by all customers in that interval is $a_i x_i^2 + b_i x_i$. We then face the following algorithmic problem: given the total power consumption, say P, in a twenty-four hour period, what is the best way of distributing P into the M time slots. The 'best' has a straight-forward interpretation; we consider that distribution to be best which minimizes C, the total cost as defined above. The notion of optimality can be justified by noting that by minimizing the total cost, we minimize the physical and personnel resources used in producing the total amount P of power. We provide here two solutions to this algorithmic problem- the first is a dynamic programming algorithm, and the other is a greedy algorithm. The dynamic programming solution provides the exact optimum solution, whereas the greedy provides an approximate solution. On the other hand, the dynamic programming runtime complexity is square of the runtime of the greedy algorithm.

We note that it is feasible to estimate the total demand P on a utility for a twenty-four hour period. P depends on the season of the year, and therefore we can estimate P with some accuracy by extrapolating historical data pertaining to that season of the year. Once we have an estimate for P, and the coefficients a_i's and b_i's, our algorithms will provide the price for each of the M

periods: for the i^{th} period, the price is $a_i x_i + b_i$, where x_i is the amount of power which is to be consumed in the i^{th} period as determined by the algorithm being used. However, we need to specify to individual customers how best each should decide on his demand. The problem is the following:

If the j^{th} customer decides to consume $y_{\{j,i\}}$ units of power, the cost that customer incurs is $(a_i x_i + b_i)y_{\{j,i\}}$, which depends on the total load x_i on the utility in that interval. x_i is the total demand for all customers of the utility. Therefore, to decide how much to consume in an interval, an individual customer needs to know, a priori, the demands of all customers for that interval, including his own. This circularity in decision making needs to be avoided.

We provide two ways to address the problem. The first solution is to compute the optimal cost for a customer from the global cost in proportion to his usage. We elaborate the idea below:

Once we have the total power demand, P, and the coefficients a_i's and b_i's, we compute the optimal distribution of P into the M intervals. In this optimal distribution, let the value for the i^{th} interval be P_i. Let the total usage of the j^{th} customer for the twenty-four hour period be p_j. We now stipulate that the j^{th} customer will use, for the i^{th} interval, an amount of power, the proportion of which with respect to his total power usage is same as the proportion of P_i to P. In other words, he sets his usage $p_{j,i}$ for the i^{th} interval to be $(p_j/P)P_i$. We can see that if every customer follows this usage policy for each of the intervals, then the distribution of P into intervals will be as per the optimal distribution that was computed; because the sum of $(p_j/P)P_i$ over all customers j will be P_i. Therefore, once we have the optimal distribution of P into M intervals, the above policy defines an ideal load curve for each user. The goal of each customer

will be to schedule his flexible loads in such a manner that his load curve is as close to this ideal load curve as possible.

The problem with this solution is that it provides a greater advantage to customers with higher demands than to those with lesser demands. The reason for this is that in this policy, a user *A* whose usage is *N* times the usage of another customer *B*, will be charged in total (when both the users schedule their demands corresponding to their ideal load curves) *N* times more than *B*. However, the quadratic nature of the cost implies that the cost of *B*'s usage to the utility is more than *N* times that of *A*. However, the advantage with the policy is that it does recover from all the customers the projected total cost on the utility for producing the total demand and therefore, the policy is fair to the utility.

The other possible policy is for each user to pay for the cost of producing an amount of electricity equal to his demand. Therefore, if the i^{th} customer has a demand of p_i in the twenty-four hour period, then his ideal load curve is obtained by optimally distributing this p_i amount into the *M* intervals so that the cost of producing this total amount p_i is minimized. The policy is fair to customers but not to utilities, as the total cost incurred by the utility for production (again, due to the quadratic dependence of the cost on the power generated) will be more than the total revenue collected from all customers. A way to mitigate this is for the utility, keeping this fact in consideration, to provide a_i and b_i values to customers suitably marked up in accordance with their demand.

In either policy the algorithmic problem remains the same: given the power consumption, say P, in a twenty-four hour period, the best way of distributing P into M time slots needs to be computed.

3.2. Algorithms for Optimally Distributing Total Load into Intervals

We are given a total of P units of demand, M intervals and a price for each interval i, which is $a_i.x_i + b_i$, where x_i is the portion of P scheduled at the i^{th} interval and a_i and b_i are constants depending upon the interval. The goal is to distribute P units of load into the M intervals so that the total cost is minimized. To note, if the P units are distributed as $x_1, x_2,\ldots, x_M$, then the total cost is obtained as the sum $(a_i.x_i + b_i)x_i$ over all x_i's.

3.3. Dynamic Programming Solution

Let *best_cost(j,n)* denote the optimum cost of allocating n units of power to the first j intervals $1,\ldots,j$. In this optimal allocation, suppose k units of power are allocated to the interval j. The cost incurred in allocating these k units to the j^{th} interval is $(a_jk + b_j)k$. We then need to optimally allocate $(n - k)$ units of power into the remaining $j-1$ intervals $1,\ldots, j-1$. This provides us with the recurrence:

$$best\_cost(j,n) = minimize\ over\ k,\ 0 \leq k \leq n,\{best\_cost(j-1, n-k) + k(a_jk + b_j)\}.$$

As is usual in dynamic programming, the recurrence is solved bottom-up, beginning from $j = 1$. For $j = 1$, since there is only one interval, the cost of consumption of n units of power is $(na_1 + b_1)n$, which is *best_cost(1, n)*, for any $n = 0$ to P. This forms the base case of the recurrence.

Once we have computed *best_cost(j-1,n)* for all values of *n* ranging from *0* to *P*, we can use the above recurrence relation to compute *best_cost(j,n)*, for all *n* ranging from *0* to *P*. In this manner, we can compute *best_cost(M, P)*, which is the minimum cost for allocating *P* units of power into *M* intervals, using the interval prices defined by the coefficients a_i's and b_i's. To obtain power allocated to each interval in the optimal allocation, a backward pass may be used: for the M^{th} interval, as can be seen in the recurrence, *k* units will be allocated for that *k*, in the range *0* to *P*, for which the value of $best\_cost(M-1, P-k) + k(a_M.k + b_M)$ is minimized. By performing memoization and thereby storing *best_cost(j,n)* values for $1 \leq j \leq M,\ 0 \leq n \leq P$, we can compute allocation for intervals *M-1* down to *1*. If, as explained above, we have determined that k_M units of power is allocated to the M^{th} interval, then $P - k_M$ units of power will be allocated to the first *M-1* intervals. From the memoization, we can get the value of $best\_cost(M-1, P - k_M)$. This value, is in turn obtained for that *k* which minimizes $best\_cost(M-2, P - k_M - k) + k(a_{M-1}.k + b_{M-1})$. This *k* is to be allocated in the optimal allocation to the interval *M-1*. In this manner, we can decide on the allocations to be made to all the intervals.

However, this memoization will require a huge amount of space, *M.(P+1)*. *M* is often chosen to be 24, assuming the day is divided into hourly intervals. *M* can be conceivably larger, for example, one may consider 15 minute intervals, as smart household meters can communicate at similar intervals. *P*, the total 24-hour power demand on a utility is large; a typical value may be 100 megawatts. Even if we take the unit conservatively to be a kilowatt, the value *P* for 100 megawatts is 100,000. Our implementation of the dynamic program therefore will not perform the usual memoization in an effort to conserve space. Instead, we make use of two lists of length *(P+1)* - we refer to them in the implementation as the *'from'* and *'to'* lists. Initially, the *'from'* list

is initialized with *best_cost(1,n)*, *n* ranging from *0* to *P*. We now use this *'from'* list to compute *best_cost(2,n)* values using the recurrence, *n* ranging from *0* to *P*. These new values are then computed into the *'to'* array. At this stage, we copy the *'to'* list to the *'from'* list, and compute into the *'to'* array the values of *best_cost(3,n), $0 \leq n \leq P$*. Iterating in this manner, we eventually obtain *best_cost(M,n)*, *n* ranging from *0* to *P*. At this time, we can compute, as explained earlier, the allocation k_M for the M^{th} interval. The problem is now reduced to allocating optimally $P - k_M$ units into the first *M-1* intervals. We now solve this new problem to find the allocation for the current last interval, namely, *M-1*; in the same manner as we had computed the allocation for the M^{th} interval. We iterate in this manner until the allocation for all the intervals are computed.

3.3.1. Input: List representing '*a*' and '*b*' coefficients for the pricing scheme; number of intervals *M* and total power to be allocated *P*.

3.3.2. Output: Lists representing optimal scheduling of flexible demands.

3.3.3. Pseudo code:

```
current_p := P;
Iterate with m from M to 2:

        Iterate with r from 0 to P:
                from[r] := (r*a[1] + b[1])*r;


        Iterate with i from 2 to m -1:
                to[0] := 0;
                Iterate with x from 1 to current_p:
                        to[x] is set to that value of k for which from[x-k] + k*(a[i]*k + b[i])
                        is minimized;
                copy to[] list into from[] list;

        alloc[m] is set to that value of k for which {from(current_p - k) + k*(a[m]*k + b[m])}
        is minimized;
        When m is M, the minimum value over K of {from(current_p - k) + k*(a[m]*k + b[m])
```

is set to global_min_cost;
current_p := current_p - alloc[m];

/ at this point we have the allocations for levels 2 to M */*
Whatever is left of P, is allocated to alloc[1];

3.3.4. Time and space complexity of the Dynamic programming algorithm

Space complexity:

The space usage is dominated by the two lists *from[0..P]* and *to[0..P]*. Therefore, the space complexity is *O(P)*.

Time complexity:

The algorithm makes *O(M)* iterations wherein in each iteration, the *from[0..P]* list is initialized which takes *O(P)* time, followed by an inner iteration *O(M)* times, each computing the *to[]* list from the current *from[]* list. Each inner iteration loop body takes $O(P^2)$ time.

Therefore, the time complexity is $O(M^2 P^2)$.

3.4. Greedy Solution

The Greedy strategy we use is: For each of the *P* units we allocate these units successively from *unit 1,…, unit P*. Given *i* allocated units, we check for each interval *j* the price for the interval if the $(i+1)^{th}$ unit is allocated to this j^{th} interval. We then allocate the $(i+1)^{th}$ unit to that interval for which the price is lowest.

3.4.1. Input: List representing '*a*' and '*b*' coefficients for the pricing scheme; number of intervals *M* and total power to be allocated *P*.

3.4.2. Output: Lists representing optimal scheduling of flexible demands.

3.4.3. Pseudo code:

```
for each of the intervals i from 1 to M, set alloc[i] to 0;
Iterate with j from1 to P:
        k is computed to be that interval for which
        (a[k]*(alloc[k]+1) + b[k])*(alloc[k]+1) value is minimum;
        alloc[k] := alloc[k] +1;
greedy_total_cost is set to the sum over j, j ranging from 1 to M, (a[j]*alloc[j] + b[j])*alloc[j];
```

3.4.4. Time and space complexity of the Greedy solution:

Space Complexity:

The space usage is dominated by the three lists *a[], b[],* and *alloc[],* each of size *M*. Therefore, the space complexity is *O(M)*.

Time Complexity:

Essentially, the algorithm runs a loop with P iterations, each iteration taking $O(M)$ time. Hence, the time complexity of the algorithm is $O(PM)$.

3.5. Comparison of the Dynamic Programming and Greedy solutions:

In terms of space, the greedy algorithm complexity is $O(M)$ whereas the dynamic programming space usage is $O(P)$. In terms of time, the greedy algorithm takes $O(PM)$ steps whereas the dynamic programming takes $O(M^2 P^2)$ steps, implying that the dynamic programming algorithm's runtime complexity is the square of that of its greedy algorithm counterpart.

We note that the dynamic programming algorithm computes the true optimum cost, while the greedy algorithm an approximation to the true solution. We run implementations of the two algorithms, varying the total power (P) and number of intervals (M), to obtain a measure of the differences between the true optimum cost and the approximation of the optimum as computed by the greedy algorithm and to witness the effect of increasing the number of intervals on the optimum solution costs.

We considered ten sets of price coefficients a_i's and b_i's (whole numbers within the range 1 through 5, uniformly distributed), namely: $(a_1, b_1) = (1, 3)$, $(a_2, b_2) = (3, 1)$, $(a_3, b_3) = (5, 3)$, $(a_4, b_4) = (3, 2)$, $(a_5, b_5) = (2, 2)$, $(a_6, b_6) = (3, 5)$, $(a_7, b_7) = (2, 3)$, $(a_8, b_8) = (4, 1)$, $(a_9, b_9) = (2, 4)$, and $(a_{10}, b_{10}) = (3, 4)$. In the simulation, we find the costs as computed by the two algorithms for

total power ranging from 1000 units to 10,000 units, in steps of 1000 units. The number of intervals are varied from 5 up till 10, using for *i* intervals, the first *i* price coefficients, *i* ranging 5 to 10. The results as follows:

Table 2: Dynamic Programming Optimal Cost and Greedy algorithm Best Cost corresponding to *M = 5* intervals and Total Power *P*

Total Power	Dynamic Programming Optimal Cost	Greedy algorithm Best Cost
1000	406514	406748
2000	1602737	1602989
3000	3588436	3588655
4000	6363606	6363844
5000	9928251	9928507
6000	14282370	14282593
7000	19425962	19426204
8000	25359029	25359287
9000	32081567	32081795
10000	39593580	39593826

Table 3: Dynamic Programming Optimal Cost and Greedy algorithm Best Cost corresponding to *M = 6* intervals and Total Power *P*

Total Power	Dynamic Programming Optimal Cost	Greedy algorithm Best Cost
1000	359796	360064
2000	1417505	1417745
3000	3172889	3173145

4000	5625946	5626180
5000	8776680	8776932
6000	12625087	12625311
7000	17171167	17171409
8000	22414923	22415191
9000	28355354	28356592
10000	34995460	34995712

Table 4: Dynamic Programming Optimal Cost and Greedy algorithm Best Cost corresponding to *M = 7* intervals and Total Power *P*

Total Power	Dynamic Programming Optimal Cost	Greedy algorithm Best Cost
1000	306764	307014
2000	1207384	1208061
3000	2702962	2703216
4000	4792150	4792379
5000	7475398	7475652
6000	10752704	10752935
7000	14624072	14624326
8000	19089496	19089727
9000	24148982	24149238
10000	29802526	29802757

Table 5: Dynamic Programming Optimal Cost and Greedy algorithm Best Cost corresponding to *M = 8* intervals and Total Power *P*

Total Power	Dynamic Programming Optimal Cost	Greedy algorithm Best Cost
1000	285606	285836
2000	1124459	1124695
3000	2516308	2516555
4000	4461151	4461414
5000	6958989	6959265
6000	10009824	10010062
7000	13613653	13613904
8000	17770477	17770739
9000	22480297	22480570
10000	27743114	27743349

Table 6: Dynamic Programming Optimal Cost and Greedy algorithm Best Cost corresponding to *M = 9* intervals and Total Power *P*

Total Power	Dynamic Programming Optimal Cost	Greedy algorithm Best Cost
1000	251370	251646
2000	988824	989069
3000	2212107	2212372
4000	3921220	3921461
5000	6116165	6116425
6000	8796937	8797174

7000	11963540	11963799
8000	15615973	15616206
9000	19754237	19754492
10000	24378330	24378605

Table 7: Dynamic Programming Optimal Cost and Greedy algorithm Best Cost corresponding to *M = 10* intervals and Total Power *P*

Total Power	Dynamic Programming Optimal Cost	Greedy algorithm Best Cost
1000	232821	233080
2000	915334	915609
3000	2047286	2047528
4000	3628674	3628937
5000	5659501	5659737
6000	8139767	8140022
7000	11069471	11069741
8000	14448614	14448855
9000	18277195	18277454
10000	22555213	22555489

We see from the results observed above that the greedy solution gives results close to the optimal as are generated by the dynamic programming solution. A trend is observed in the above data: for each value of total power, the cost decreases as the number of intervals are increased. For example, for total power value of 10,000, when we have five intervals using the first five of the price coefficients, the optimum cost is 39593580. This optimum cost decreases to 22555213 when the number of intervals is ten- a decrease of around 43%. We attribute the reason for this trend to be that as we increase the number of intervals (using price coefficients which are not very different), the total power spreads more evenly across the intervals. Therefore, the power consumption in each interval when the number of intervals is greater will be, in general, less than the power consumptions in intervals when there is a lesser number of intervals. Since there is a quadratic dependence of the cost on power consumption, the total cost continually becomes less as we increase the number of intervals.

In the next chapter, a number of heuristic algorithms are discussed and evaluated for the purpose of load scheduling on the customer end, given an optimal allocation of total demand over a day.

Chapter 4: Heuristic Algorithms for Load Scheduling

4.1. The Load Scheduling Problem and the Need for Heuristic Approximation

We have seen in the last chapter how an ideal load curve can be defined for each grid subunit customer. Such a customer will have a number of requirements or loads, called inflexible loads whose schedules are a priori fixed, and another set of loads, called flexible loads, for which there are margins in scheduling. (The following section 4.2 explains inflexible and flexible loads in detail). The load scheduling problem that a grid subunit customer faces is the following: Given the details of his inflexible and flexible loads, and his ideal load curve (as obtained in Chapter 3), schedule the flexible loads in such a manner that the resultant load curve is as close to his ideal load curve as possible. This section examines why there is a need for heuristic algorithms for reaching an optimum and how the scheduling problem can be proven to be NP-hard by reducing the Partition problem to it. The proof is as follows:

We consider the Partition problem given a set *S* of *n* numbers, the sum of all these numbers being *2M*. Is there a subset *S'* of *S* such that the sum of numbers in *S'* is exactly *M*, i.e. half of the total sum *2M?* The reduction of this problem to the load scheduling problem is as follows:

For every number in *S*, we consider a corresponding flexible demand whose power requirement is equal to the value of the number, its duration for running is 1, and it can be scheduled at the start of any hour in the 24-hour period. We consider an ideal load profile to be *M* amount of power to be consumed between say 10 AM – 11 AM, and a further *M* amount to be consumed

between 1 PM – 2 PM. Thus, there is an optimal scheduling of the flexible demands to exactly correspond to the load profile if and only if the solution to the Partition problem is 'yes'.

The heuristic algorithms used in the thesis for the purpose of load scheduling are the Greedy search, randomized Greedy search, Tabu search, randomized Tabu search, and finally, the Metropolis algorithm.

4.2. Experiment Simulation Setup

We consider a domestic dwelling unit whose loads we seek to schedule in a way that conforms to an ideal load profile. The loads for the unit are divided into two kinds: flexible loads and inflexible loads.

4.2.1. Inflexible Loads:

These are loads that may not be scheduled any differently than they were originally scheduled to run. The duration that these loads run for, as well as the time that they were to be started at, may not be modified under any circumstances. For instance, lighting in a dwelling unit can be considered to be an inflexible load, since a resident typically needs lighting between certain times of a 24 hour period that cannot be compromised.

4.2.2. Flexible Loads:

Flexible loads are unlike inflexible loads in that they may be rescheduled for a different point in the day than they were originally scheduled for. It is permissible to deviate their starting times, but not the duration for which they had originally been slotted. For instance, a washing machine is an example of a flexible load. While the use of a washing machine in a day may be essential to the resident, the time that it runs is not typically of concern as long as there is an assurance that its operation shall be completed within the 24 hour period.

We studied electrical appliances that may be present typically in a residence, along with their power usages and durations they run for on an average day, as presented in [12]. The assumption is made that the requirements or demands of a typical household is relatively stable over a season. These appliances were then separated out into the categories of flexible loads and inflexible loads.

Table 8: Household Appliance Data

Appliance	Category	Power (kW)	Mean duration (hours)	Start time (without delay)
Air conditioning	Inflexible	3.5	8	12
Interior lighting	Inflexible	0.84	6	18
Exterior lighting	Inflexible	0.3	2	19
Oven	Inflexible	5	0.4	17
Microwave	Inflexible	1.7	0.2	17
Grill	Inflexible	1	0.1	17
Refrigerator & Freezer	Inflexible	0.7	10	5
Kettle	Inflexible	3.8	0.25	6, 18
Toaster	Inflexible	0.5	0.1	6
Electric shower with heater	Inflexible	9	0.5	6
Hair dryer	Inflexible	1.5	0.1	7
Home computing	Flexible/Inflexible	0.6	3 (Flexible), 1 (Inflexible)	19 (Flexible), 18 (Inflexible)
Television	Inflexible	0.6	4	18
Sensors	Inflexible	0.01	24	0
Dishwasher	Flexible	1	1.5	11, 18
Washing machine	Flexible	1	1.5	19
Spin Dryer	Flexible	3	20	1
Clothes Iron	Flexible	2	0.5	19
Vacuum Cleaner	Flexible	1.2	0.5	19
DVD Player	Flexible	0.2	2	18
Lawn Mower	Flexible	1	0.5	16

Swimming Pool Pumps	Flexible	0.2	12	10
Swimming Pool Heating	Flexible	4.5	6	17

By considering these appliances as loads, and making educated assumptions for their start times, we arrive at a load profile for a typical residential customer of the utility.

To successfully test the efficacy of our heuristic algorithms, we generated many different realistic load profiles for a household to experiment on, with the above load profile as a template. Start times of loads are generated by adding a random variable conforming to the exponential probability distribution with the rate parameter 0.5, to the start time of that particular load in the model in order to simulate a delay. Durations of loads are generated by selecting a random variable conforming to the normal probability distribution with the duration of that particular load as the mean, and a percentage of the mean as the standard deviation.

4.2.3. Neighborhood Search

We consider each schedule as a node in a search space graph. Neighbors of a node will consist of schedules formed by shifting the start time of each flexible load by one unit earlier, and one unit later, where possible. To increase our reach and improve upon the quality of the local optimum that the search process yields, we define another set of neighbors of a node, shifting a start time three units instead of one. Therefore, each node will have $4n$ neighbors, where n is the number of flexible requirements.

We associate ***essential*** and ***desirable*** properties with the search space graph defined above. The first essential property is that every possible schedule must be defined by a node in the graph, the second being that every such schedule must be reachable from any node in the graph, i.e., the graph must be connected. The only desirable property is that the number of neighbors for a node should be polynomial.

Arriving at the optimum schedule will be done through traversals of this neighborhood search space. These traversals occur by successively examining neighboring nodes' ***schedule costs***, and choosing a node based on the algorithm being used. Ultimately, the node (among all explored nodes) that has the least value of schedule cost is chosen as the search result.

4.2.4. Schedule Cost: The schedule cost of a node is defined as the total area between the schedule that the node represents and the ideal load profile, as discussed earlier.

For the subsequent experiment simulations, without loss of generality, an ideal load profile is considered to be a flat curve corresponding to the average of the total demand of a residential unit over 24 time intervals. Thus, the cost of a schedule (or node in the search space) will be the total area between the load profile corresponding to the schedule and the flat ideal load profile. For example, for the cost of a schedule to be 91.67, the absolute sum of the area between the load profile for that schedule and the ideal load profile for the simulation in question is equal to 91.67 units.

4.2.5. Helper Functions for Heuristic Searches

a) Function to compute a load profile:

This function computes and then returns the load profile corresponding to the flexible schedule given as its argument.

b) Function to calculate cost of a given schedule:

This function is used to calculate the cost of a schedule by using its load profile and the ideal load profile, summing up their absolute differences for each hour.

c) Function to compute neighbors of a schedule:

A neighbor of a flexible schedule is obtained by choosing one of the flexible demands and then shifting its start time, either to the left or the right. We consider a shift to be either by one hour or three hours.

We represent all possible neighbors of a schedule using a bit array- the bits $4i$, $4i + 1$, $4i + 2$, and $4i + 3$ are for the i^{th} flexible demand. The $4i^{th}$ bit being 1 represents a neighboring schedule by shifting the start time of the i^{th} demand one hour to the left. We note that such a neighboring schedule could be defined if and only if the start time of the i^{th} flexible demand is not 0 in the current schedule. Similarly, the bit $4i + 1$ being 1 indicates that a neighbor can be obtained by shifting the start time of the i^{th} demand by one hour to the right of the current start time of the i^{th} demand. Such a neighbor exists if the sum of the start hour of the i^{th} job and its duration is strictly less than 23.

The bits $4i + 2$ and $4i + 3$ represent neighbors obtained by shifting the i^{th} demand's start time by -3 or +3 hours.

d) Function to generate a random schedule:

We use this function to randomly generate a schedule of flexible demands, which obeys the constraints of start times and durations for each load. This function is used to restart exploration of the search space when a particular node is the local optimum. A restart may be necessary for the Greedy and Tabu Search algorithms, but is not a requirement for the Metropolis algorithm.

4.3. Criteria for Evaluation of Performances between Heuristic Algorithms

Given an initial node that corresponds to an unscheduled residential load profile, the heuristic algorithms are run a number of times, changing certain parameters that are unique to the algorithm in question. The average final costs of the scheduled load profile are compared, as are the standard deviations in the average final costs computed for each algorithm. Averages of times taken for execution of the algorithms are also considered as a performance criterion. Thus, average final costs, the standard deviation between final costs, and the execution times of algorithms are chosen to evaluate performances of the different algorithms. A number of simulations to measure these performance criteria are performed for different unscheduled load profiles, and a final conclusion on performance is presented in Section 9 of this chapter.

4.4. Greedy Search

In this section, we discuss a Greedy algorithm to walk through the search space, with the aim to find an acceptable local optimum.

4.4.1. Input: List representing initial schedule of flexible demands; Iteration count.

4.4.2. Output: List representing optimal scheduling of flexible demands.

4.4.3. Pseudo code:

Iterate from 0 to Iteration count:

Compute neighbors for current schedule;

Best Schedule := First neighbor for current schedule;

Best Cost := Cost associated with the first schedule;

Iterate through all neighbors of the current schedule:

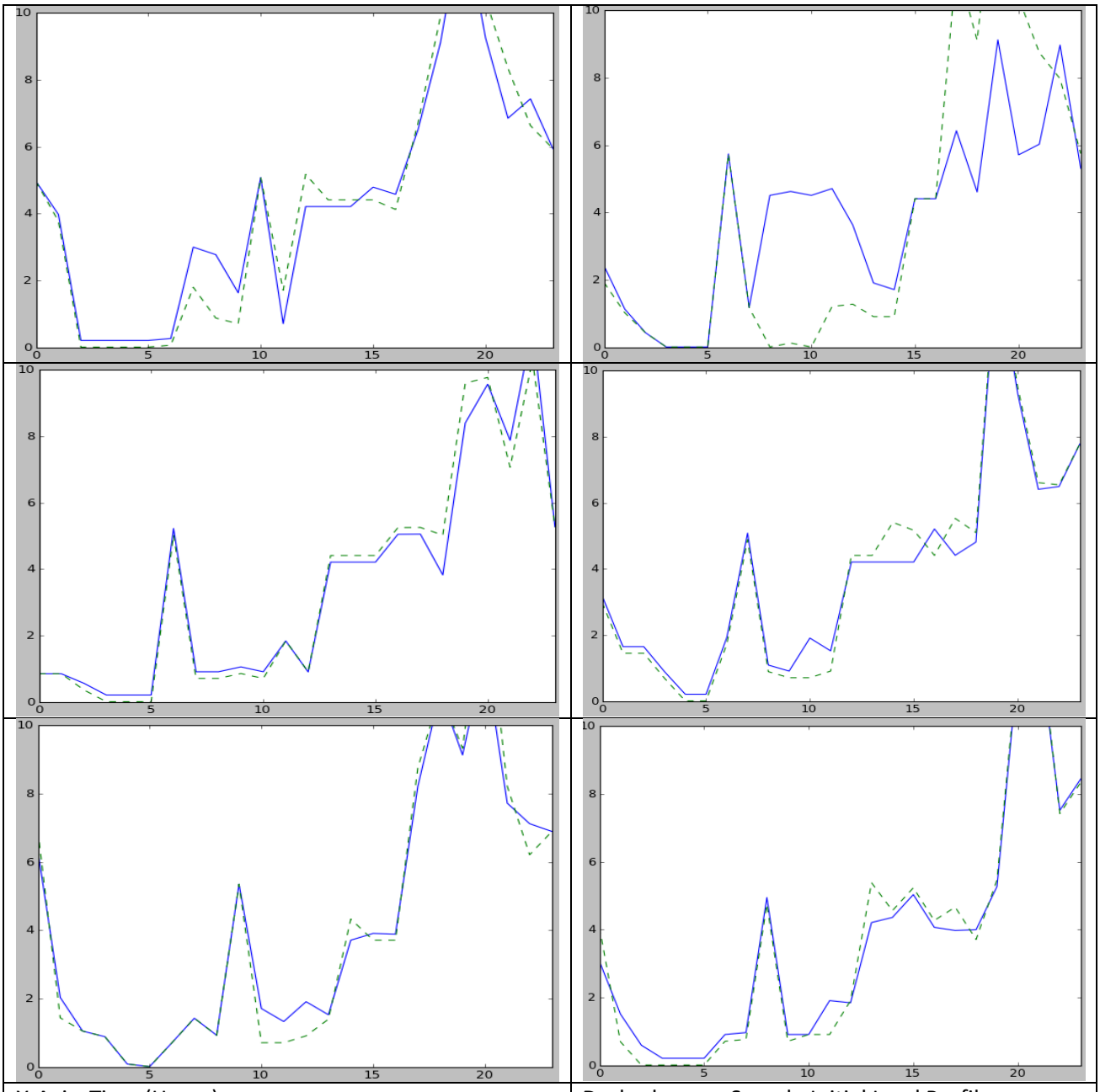
If cost of neighbor being examined < Best Cost:

Best Cost := Cost of neighbor being examined;

Best Schedule := Current neighbor;

Until Best Neighbor Cost < Current Cost;

4.4.4. Observations



X-Axis: Time (Hours)	Dashed curve: Sample Initial Load Profiles
Y-Axis: Power (kW)	Solid curve: Final load profile after scheduling with Greedy algorithm

Figure 3: Sample Initial Unscheduled Load Profiles and Final Scheduled Load Profiles after scheduling with the Greedy Algorithm

Table 9: Greedy Algorithm Analysis

Initial Cost	Final Cost	Time Taken	Iterations
91.67	**27.43**	**0.10**	**1000**
74.12	**65.72**	**0.03**	**1000**
101.92	**50.28**	**0.04**	**1000**
92.28	**88.32**	**0.02**	**1000**
78.34	**53.33**	**0.05**	**1000**
91.67	**27.43**	**0.10**	**10000**
74.12	**65.72**	**0.03**	**10000**
101.92	**50.28**	**0.04**	**10000**
92.28	**88.32**	**0.02**	**10000**
78.34	**53.33**	**0.05**	**10000**
91.67	**27.43**	**0.10**	**50000**
74.12	**65.72**	**0.03**	**50000**
101.92	**50.28**	**0.04**	**50000**
92.28	**88.32**	**0.02**	**50000**
78.34	**53.33**	**0.05**	**50000**

As we see from the table above, for our residential unit scenario, the final cost and time taken for the algorithm to compute for an initial cost is not dependent on the number of iterations that the Greedy algorithm is run for. Thus, the number of iterations in the Greedy algorithm is not important, as long as it is above a certain threshold.

This can be explained by understanding that the Greedy algorithm executes only while there exists a neighboring node with a lower cost, for any given current node. We can conclude from the results above that the Greedy algorithm reaches a local optimum for all observed cases, and ceases to execute further.

4.5. Randomized Greedy Search

Here we discuss a similar Greedy algorithm to walk through the search space, with a random component added to maximize its reach for finding the optimum in the search space.

4.5.1. Input: List representing initial schedule of flexible demands; Iteration count.

4.5.2. Output: List representing optimal scheduling of flexible demands.

4.5.3. Pseudo code:

Iterate from 0 to Iteration count:

Compute neighbors for current schedule;

Best Schedule := First neighbor for current schedule;

Best Cost := Cost associated with the first schedule;

Iterate through all neighbors of the current schedule:

If cost of neighbor being examined < Best Cost:

Best Cost := Cost of neighbor being examined;

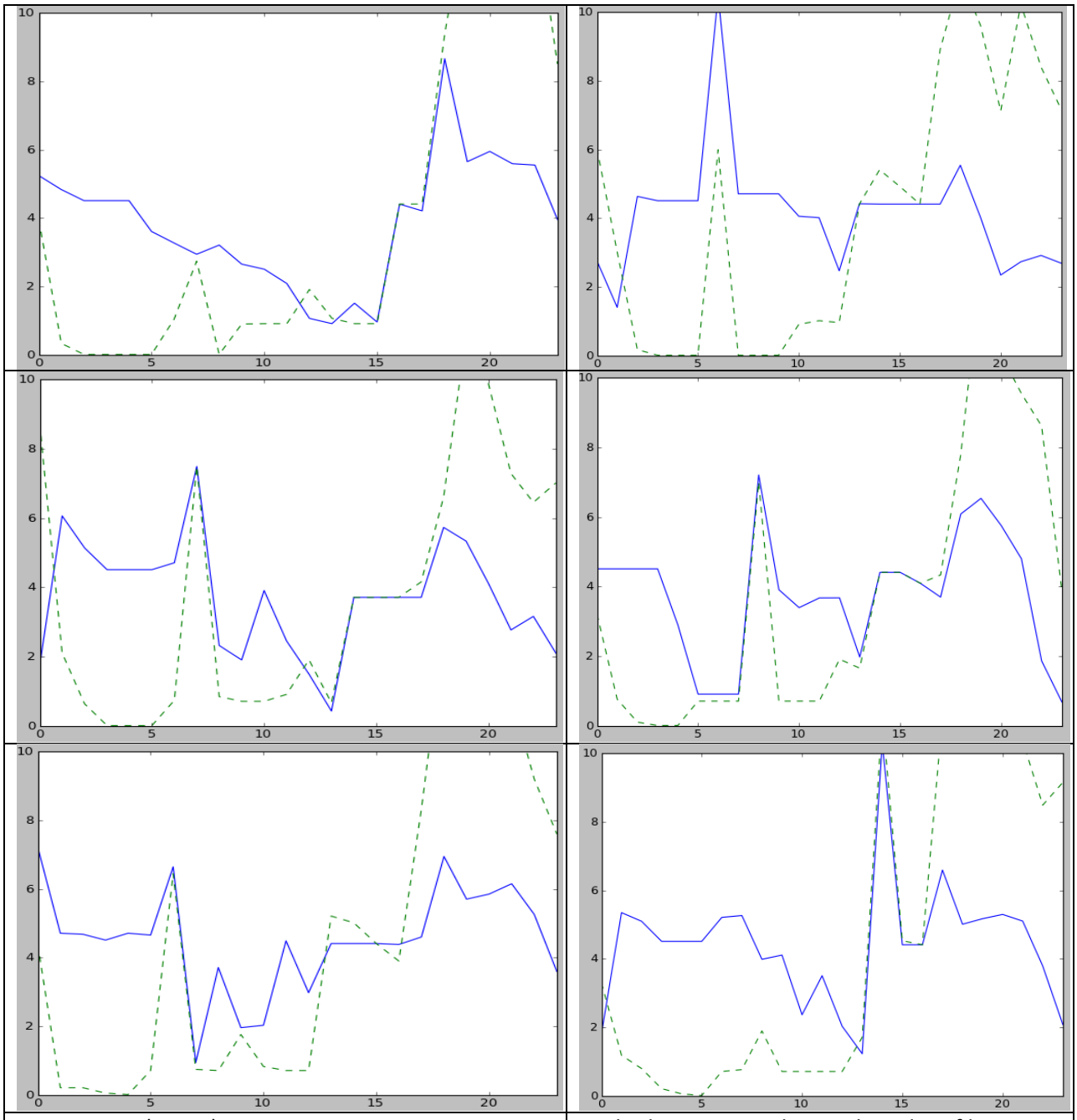
Best Schedule := Current neighbor;

If Current Cost < Best Neighbor Cost:

Compute a random schedule;

Current Schedule := Random schedule computed in last step;

4.5.4. Observations



X-Axis: Time (Hours)	Dashed curve: Sample Initial Load Profiles
Y-Axis: Power (kW)	Solid curve: Final load profile after scheduling with Randomized Greedy algorithm

Figure 4: Sample Initial Unscheduled Load Profiles and Final Scheduled Load Profiles after scheduling with the Randomized Greedy Algorithm

Table 10: Randomized Greedy Algorithm Analysis with Schedule of Initial Cost 91.67 as Input

Final Cost	Percentage Reduction	Number of Iterations
25.32	72.37	1000
25.32	72.37	1000
25.32	72.37	1000
25.32	72.37	1000
25.32	72.37	1000
25.32	72.37	1000
25.56	72.11	1000
25.32	72.37	1000
25.32	72.37	1000
25.43	72.25	1000
25.32	72.37	5000
25.32	72.37	5000
25.32	72.37	5000
25.32	72.37	5000
25.32	72.37	5000
25.32	72.37	5000
25.32	72.37	5000
25.32	72.37	5000
25.32	72.37	5000
25.32	72.37	5000

Average Final Cost (1000 iterations) = 25.36

Average Time Taken for each run (1000 iterations) = 4.73 s

Standard deviation in Final Cost (1000 iterations) = 0.078

Average Final Cost (5000 iterations) = 25.32

Average Time Taken for each run (5000 iterations) = 23.78 s

Standard deviation in Final Cost (5000 iterations) = 0

Performing a similar simulation for load balancing with a scenario having an initial cost of 74.12, the following results are obtained:

Average Final Cost (1000 iterations) = 25.13

Average Time Taken for each run (1000 iterations) = 4.17 s

Standard deviation in Final Cost (1000 iterations) = 0.019

Average Final Cost (5000 iterations) = 25.12

Average Time Taken for each run (5000 iterations) = 20.99 s

Standard deviation in Final Cost (5000 iterations) = 0

Performing a similar simulation for load balancing with a scenario having an initial cost of 101.92, the following results are obtained:

Average Final Cost (1000 iterations) = 36.79

Average Time Taken for each run (1000 iterations) = 4.14 s

Standard deviation in Final Cost (1000 iterations) = 0

Average Final Cost (5000 iterations) = 36.79

Average Time Taken for each run (5000 iterations) = 20.64 s

Standard deviation in Final Cost (5000 iterations) = 0

Performing a similar simulation for load balancing with a scenario having an initial cost of 92.28, the following results are obtained:

Average Final Cost (1000 iterations) = 26.49

Average Time Taken for each run (1000 iterations) = 4.50 s

Standard deviation in Final Cost (1000 iterations) = 0.756

Average Final Cost (5000 iterations) = 25.52

Average Time Taken for each run (5000 iterations) = 22.20 s

Standard deviation in Final Cost (5000 iterations) = 0.194

Performing a similar simulation for load balancing with a scenario having an initial cost of 78.34, the following results are obtained:

Average Final Cost (1000 iterations) = 36.59

Average Time Taken for each run (1000 iterations) = 4.25 s

Standard deviation in Final Cost (1000 iterations) = 0.116

Average Final Cost (5000 iterations) = 36.52

Average Time Taken for each run (5000 iterations) = 21.23 s

Standard deviation in Final Cost (5000 iterations) = 0

It can be observed from the results of the simulations above that the randomized greedy algorithm consistently generates a schedule of a much lower cost as compared to the cost of a given initial schedule. The algorithm is different from the deterministic greedy search in that it does not cease execution upon reaching a local optimum, and instead computes a random schedule from the initial schedule. Using the random schedule as the current node, it continues exploring the neighborhood search space. This randomization has clear benefits as can be seen from the results above, where the randomized greedy search yields much better results for the same initial schedules, as compared to its deterministic counterpart.

4.6. Tabu Search

4.6.1. Input: List representing initial schedule of flexible demands; Iteration count.

4.6.2. Output: List representing optimal scheduling of flexible demands.

4.6.3. Pseudo code:

Iterate from 0 to Iteration count:

Compute neighbors for current schedule;

Initialize best and next best costs and schedules;

Append the schedule selected to a Tabu list (implemented as a circular queue containing pairs (D, x) where D denotes the demand, and x denotes the time by which it was shifted);

If (D, -x) present in Tabu list:

// If the pair is present, there is a possibility of a cycle

If a cycle is present:

Current schedule := Next-best neighborhood schedule;

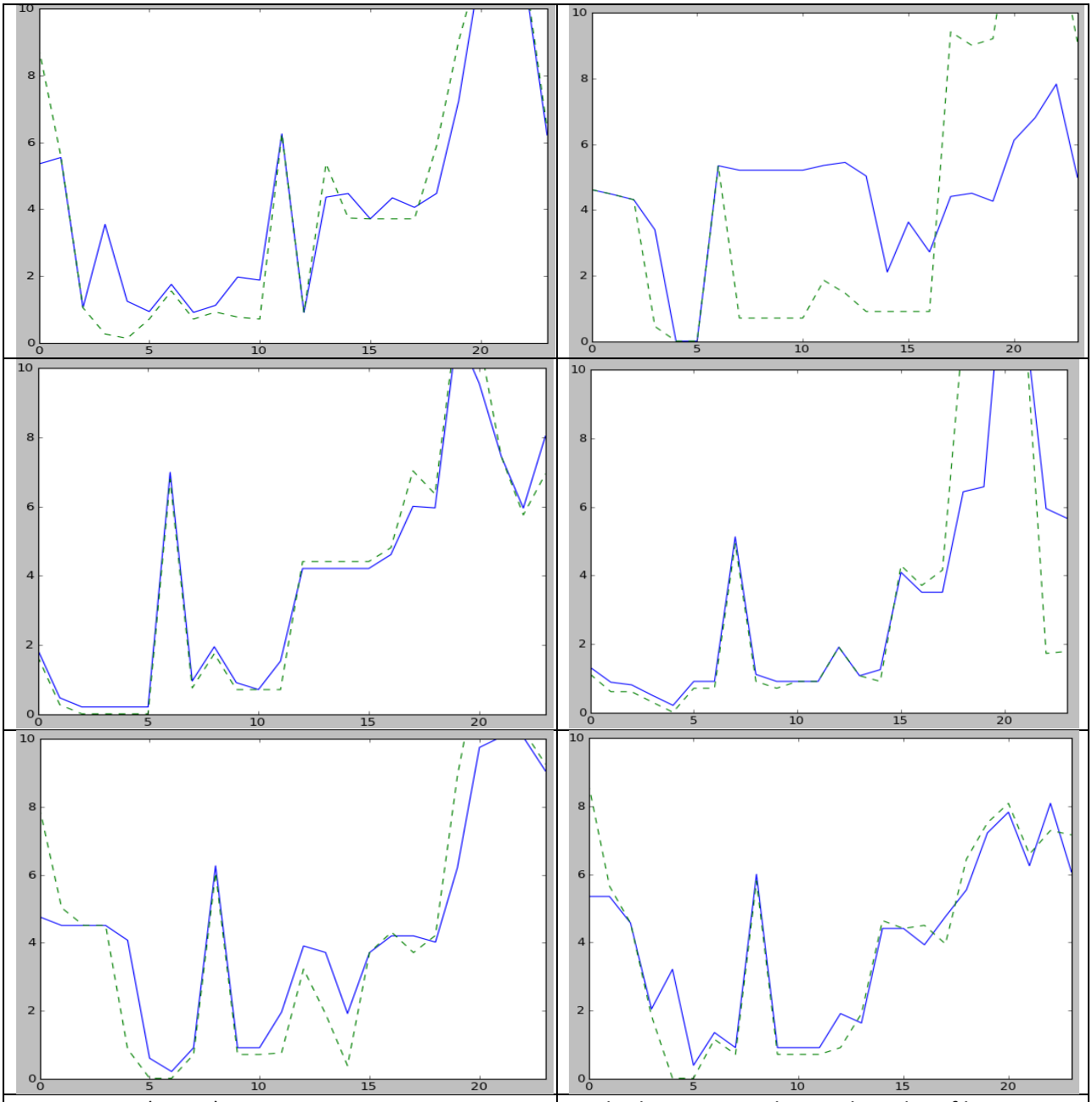
Current cost := Next-best neighborhood schedule cost;

Else:

Current schedule := Best neighborhood schedule;

Current cost := Best neighborhood schedule cost;

4.6.4. Observations



X-Axis: Time (Hours)	Dashed curve: Sample Initial Load Profiles
Y-Axis: Power (kW)	Solid curve: Final load profile after scheduling with Tabu search

Figure 5: Sample Initial Unscheduled Load Profiles and Final Scheduled Load Profiles after scheduling with the Tabu Search Algorithm

Table 11: Tabu Search Algorithm Analysis

Initial Cost	Final Cost	Time Taken	Number of Iterations
91.67	**27.43**	**2.51**	**500**
91.67	**27.43**	**5.31**	**1000**
74.12	**65.72**	**1.89**	**500**
74.12	**65.72**	**3.75**	**1000**
101.92	**50.28**	**1.78**	**500**
101.92	**50.28**	**3.58**	**1000**
92.28	**84.91**	**2.09**	**500**
92.28	**84.91**	**4.09**	**1000**
78.34	**53.33**	**1.98**	**500**
78.34	**53.33**	**3.96**	**1000**

From the table above, it can be observed that increasing the number of iterations for the algorithm, with the size of the Tabu list remaining constant, is not having any effect on the final cost computed. We next examine if, for a given initial schedule, changing the Tabu list size has any effect on the final cost computed.

Table 12: Tabu Search Algorithm Analysis for Initial Schedule of Cost 91.678

Average Final Cost	Tabu List Size	Number of iterations	Average Time Taken (seconds)
27.43	**5**	**500**	**2.51**
27.43	**10**	**500**	**2.51**
27.43	**15**	**500**	**2.51**
27.43	**20**	**500**	**2.51**
27.43	**25**	**500**	**2.51**
27.43	**30**	**500**	**2.52**
27.43	**35**	**500**	**2.50**
27.43	**40**	**500**	**2.51**
27.43	**45**	**500**	**2.51**
27.43	**50**	**500**	**2.52**
27.43	**5**	**1000**	**5.31**
27.43	**10**	**1000**	**5.03**

27.43	**15**	**1000**	**5.03**
27.43	**20**	**1000**	**5.02**
27.43	**25**	**1000**	**5.03**
27.43	**30**	**1000**	**5.04**
27.43	**35**	**1000**	**5.04**
27.43	**40**	**1000**	**5.38**
27.43	**45**	**1000**	**5.04**
27.43	**50**	**1000**	**5.21**

The above simulation is run for scenarios with initial costs of 74.12, 101.92, 92.28 and 78.34. Similar results are observed wherein there is no change observed in the average final cost computed, and with minimal changes in the average running times of the Tabu implementation.

Therefore, it is noted that changing the size of the Tabu list does not have any effect on the final cost computed.

4.7. Randomized Tabu Search

4.7.1. Input: List representing initial schedule of flexible demands; Iteration count.

4.7.2. Output: List representing optimal scheduling of flexible demands.

4.7.3. Pseudo code:

Iterate from 0 to Iteration count:

Compute neighbors for current schedule;

Initialize best and next best costs and schedules;

Append the schedule selected to a Tabu list (implemented as a circular queue containing pairs (D, x) where D denotes the demand, and x denotes the time by which it was shifted);

If (D, -x) present in Tabu list:

// If the pair is present, there is a possibility of a cycle

If a cycle is present:

With a 0.5 probability,

Current schedule := Next-best neighborhood schedule;

Current cost := Next-best neighborhood schedule cost;

With a 0.5 probability,

Current schedule := A random schedule computed;

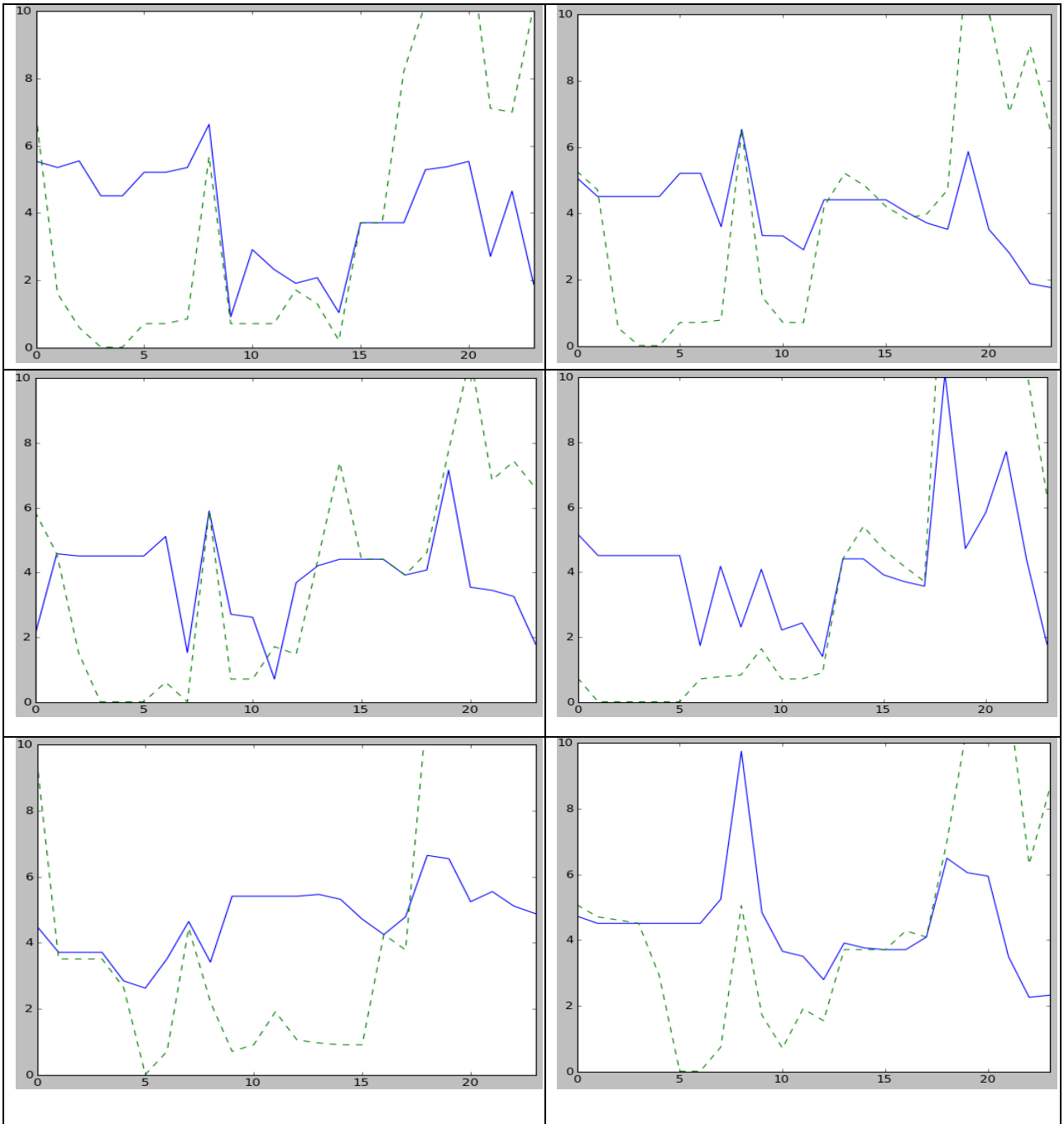
Current cost := Random schedule cost;

Else:

Current schedule := Best neighborhood schedule;

Current cost := Best neighborhood schedule cost;

4.7.4. Observations



X-Axis: Time (Hours)	Dashed curve: Sample Initial load profiles
Y-Axis: Power (kW)	Solid curve: Final load profile after scheduling using Randomized Tabu search

Figure 6: Sample Initial Unscheduled Load Profiles and Final Scheduled Load Profiles after scheduling with the Randomized Tabu Search Algorithm

Starting with a node of a certain initial cost, we run the randomized Tabu search of 500 iterations a total of ten times, keeping the Tabu list size constant for the ten simulations. The size of the Tabu list is incrementally increased to evaluate whether it has any effect on the quality of the solutions computed by the algorithm. The resulting final costs and times taken for execution are analyzed, and their averages for the ten simulations are displayed in the tables below:

Table 13: Randomized Tabu Search Analysis for Schedule with Initial Cost 91.67

Average Final Cost	Tabu Size	Number of iterations	Standard Deviation in Final Cost	Average Time Taken (seconds)
25.39	5	500	0.135	2.40
25.47	10	500	0.202	2.41
25.47	15	500	0.276	2.41
25.52	20	500	0.269	2.42
25.69	25	500	0.426	2.62
25.83	30	500	0.729	2.60
25.66	35	500	0.423	2.42
25.99	40	500	0.599	2.44
25.20	45	500	0.659	2.63
26.08	50	500	0.733	2.61
25.33	5	1000	0.033	5.00
25.43	10	1000	0.207	4.82
25.39	15	1000	0.225	4.99
25.42	20	1000	0.135	5.00
25.44	25	1000	0.192	5.04
25.37	30	1000	0.148	5.21
25.63	35	1000	0.547	5.24
25.35	40	1000	0.050	5.20
25.43	45	1000	0.150	4.89
25.53	50	1000	0.489	5.42

Table 14: Randomized Tabu Search Analysis for Schedule with Initial Cost 74.12

Average Final Cost	Tabu Size	Number of iterations	Standard Deviation in Final Cost	Average Time Taken (seconds)
25.20	5	500	0.125	2.01
25.49	10	500	0.541	1.98
25.38	15	500	0.302	2.01
25.33	20	500	0.288	2.00
25.79	25	500	0.797	2.02
25.57	30	500	0.432	1.99
25.78	35	500	0.689	1.96
25.82	40	500	1.319	1.96
25.70	45	500	0.432	1.95
28.29	50	500	8.185	1.93
25.13	5	1000	0.026	4.02
25.13	10	1000	0.038	3.97
25.17	15	1000	0.121	3.99
25.16	20	1000	0.122	3.98
25.29	25	1000	0.333	3.93
25.28	30	1000	0.310	3.92
25.28	35	1000	0.322	3.95
25.22	40	1000	0.170	3.89
25.23	45	1000	0.166	3.88
25.26	50	1000	0.185	4.02

Table 15: Randomized Tabu Search Analysis for Schedule with Initial Cost 101.92

Average Final Cost	Tabu Size	Number of iterations	Standard Deviation in Final Cost	Average Time Taken (seconds)
36.79	5	500	0	2.01
36.87	10	500	0.168	2.01
37.02	15	500	0.729	2.01
37.00	20	500	0.393	2.01
37.36	25	500	1.006	1.99
37.09	30	500	0.467	1.99
36.93	35	500	0.288	1.98
37.18	40	500	0.598	1.98
37.27	45	500	1.101	1.99
37.99	50	500	0.887	2.00
36.82	5	1000	0.087	4.05
36.79	10	1000	0	4.06
36.79	15	1000	0	4.06

36.85	**20**	**1000**	**0.191**	**3.99**
36.83	**25**	**1000**	**0.126**	**3.99**
36.96	**30**	**1000**	**0.392**	**4.02**
36.79	**35**	**1000**	**0**	**4.00**
36.92	**40**	**1000**	**0.203**	**3.97**
36.87	**45**	**1000**	**0.178**	**4.02**
37.06	**50**	**1000**	**0.258**	**4.03**

Table 16: Randomized Tabu Search Analysis for Schedule with Initial Cost 92.28

Average Final Cost	Tabu Size	Number of iterations	Standard Deviation in Final Cost	Average Time Taken (seconds)
26.84	**5**	**500**	**0.671**	**2.20**
27.89	**10**	**500**	**1.470**	**2.21**
27.31	**15**	**500**	**1.254**	**2.20**
28.50	**20**	**500**	**1.670**	**2.20**
28.77	**25**	**500**	**1.386**	**2.18**
27.83	**30**	**500**	**1.448**	**2.17**
28.71	**35**	**500**	**1.209**	**2.20**
28.38	**40**	**500**	**1.683**	**2.18**
29.04	**45**	**500**	**1.325**	**2.15**
28.88	**50**	**500**	**1.750**	**2.14**
26.17	**5**	**1000**	**0.428**	**4.41**
26.22	**10**	**1000**	**0.566**	**4.42**
26.54	**15**	**1000**	**0.990**	**4.43**
26.67	**20**	**1000**	**0.912**	**4.41**
26.98	**25**	**1000**	**0.795**	**4.42**
27.29	**30**	**1000**	**1.058**	**4.42**
27.34	**35**	**1000**	**0.828**	**4.40**
27.64	**40**	**1000**	**1.097**	**4.35**
27.63	**45**	**1000**	**1.239**	**4.39**
27.78	**50**	**1000**	**0.897**	**4.35**

Table 17: Randomized Tabu Search Analysis for Schedule with Initial Cost 78.34

Average Final Cost	Tabu Size	Number of iterations	Standard Deviation in Final Cost	Average Time Taken (seconds)
36.81	5	500	0.311	2.09
36.87	10	500	0.179	2.10
36.95	15	500	0.503	2.07
37.55	20	500	1.646	2.07
37.03	25	500	0.337	2.09
37.29	30	500	0.656	2.08
38.75	35	500	3.630	2.09
37.55	40	500	1.597	2.10
38.61	45	500	2.307	2.05
38.42	50	500	2.634	2.06
36.70	5	1000	0.136	4.22
36.69	10	1000	0.163	4.18
36.80	15	1000	0.165	4.20
36.82	20	1000	0.233	4.17
36.75	25	1000	0.213	4.18
36.78	30	1000	0.238	4.18
36.80	35	1000	0.220	4.20
37.21	40	1000	0.811	4.14
36.80	45	1000	0.270	4.17
36.99	50	1000	0.746	4.19

As can be observed from the results of the simulations above, the size of the Tabu list and the number of iterations (beyond a certain threshold) do not have a significant impact in computing final costs.

4.8. Metropolis Algorithm

4.8.1. Input: List representing initial schedule of flexible demands; Metropolis temperature, Iteration count.

4.8.2. Output: List representing optimal scheduling of flexible demands.

4.8.3. Pseudo code:

Best schedule := Current schedule;

Best cost := Cost of current schedule;

Iterate from 0 to Iteration count:

Pick at random a neighbor of the current schedule, say v;

If cost of neighbor v < Cost of current schedule:

Current schedule := v;

Best schedule := v;

Else:

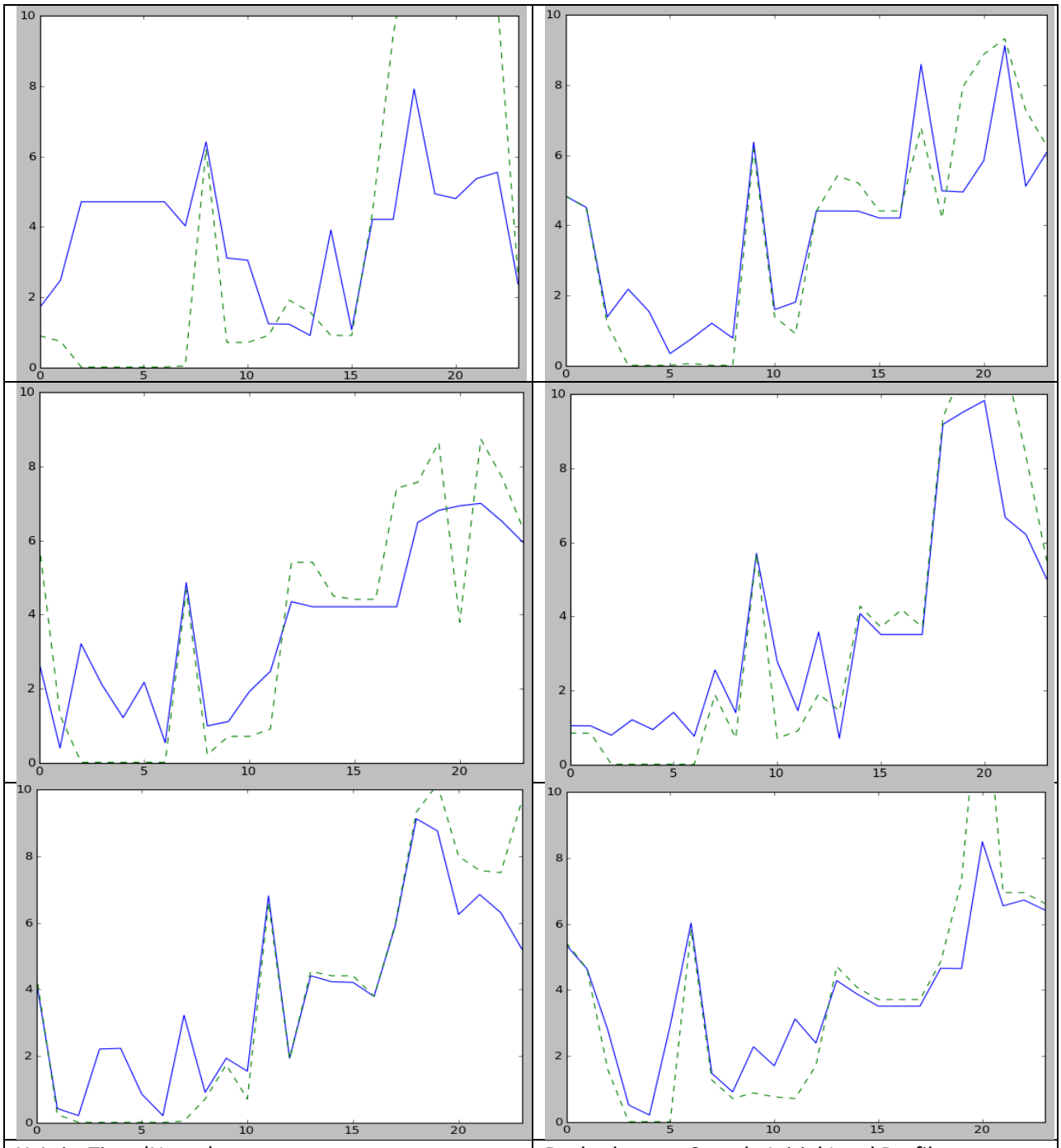
Compute probability p := Exp((Cost of current Best schedule – Cost of v)/ Metropolis temperature);

Compute a uniformly random number x between 0 and 1;

If $x \leq p$:

Current schedule := v ;

4.8.4. Observations



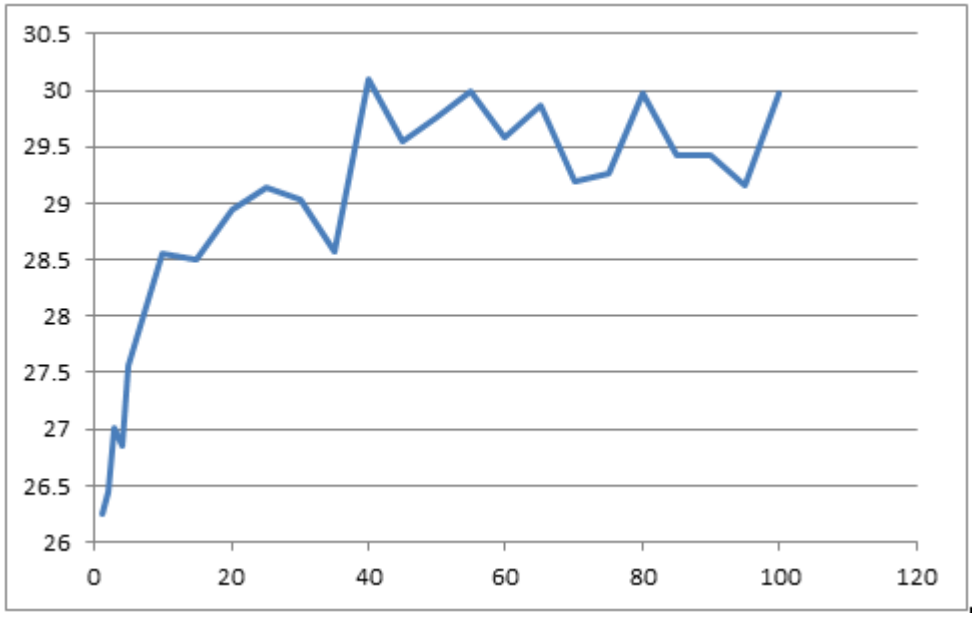
X-Axis: Time (Hours)	Dashed curve: Sample Initial Load Profiles
Y-Axis: Power (kW)	Solid curve: Final load profile after scheduling using Metropolis algorithm (Temperature: 1)

Figure 7: Sample Initial Unscheduled Load Profiles and Final Scheduled Load Profiles after scheduling with the Metropolis Algorithm

Table 18: Metropolis Algorithm Analysis for Schedule with Initial Cost 91.67

Average Final Cost	Temperature	Number of Iterations	Standard Deviation in Final Cost	Average Time Taken (seconds)
27.05	1	1000	1.065	0.27
28.21	2	1000	0.971	0.27
29.59	3	1000	1.378	0.27
30.71	4	1000	1.027	0.27
29.61	5	1000	1.924	0.27
31.01	10	1000	1.273	0.27
32.54	15	1000	1.061	0.27
32.14	20	1000	1.323	0.43
32.22	25	1000	1.732	0.27
31.95	30	1000	1.336	0.27
32.95	35	1000	1.515	0.27
32.53	40	1000	1.548	0.27
31.82	45	1000	1.256	0.27
31.60	50	1000	1.689	0.27
32.07	55	1000	1.550	0.27
32.59	60	1000	2.105	0.27
33.00	65	1000	2.150	0.27
31.92	70	1000	1.077	0.27
31.40	75	1000	1.537	0.27
32.60	80	1000	1.780	0.27
32.48	85	1000	2.106	0.27
33.01	90	1000	2.550	0.47
32.19	95	1000	1.547	0.27
32.24	100	1000	1.967	0.28
26.25	1	10000	0.877	2.92
26.45	2	10000	0.723	2.92
27.01	3	10000	0.477	2.75
26.85	4	10000	1.007	2.91
27.56	5	10000	0.700	3.15
28.54	10	10000	0.700	2.92
28.50	15	10000	0.940	2.91
28.95	20	10000	0.592	3.28
29.14	25	10000	1.308	2.97
29.03	30	10000	0.943	2.91
28.56	35	10000	0.855	2.92
30.10	40	10000	0.680	2.75
29.54	45	10000	1.146	2.75
29.75	50	10000	0.912	2.95
29.99	55	10000	0.820	3.13
29.58	60	10000	0.874	2.75
29.86	65	10000	0.618	2.75

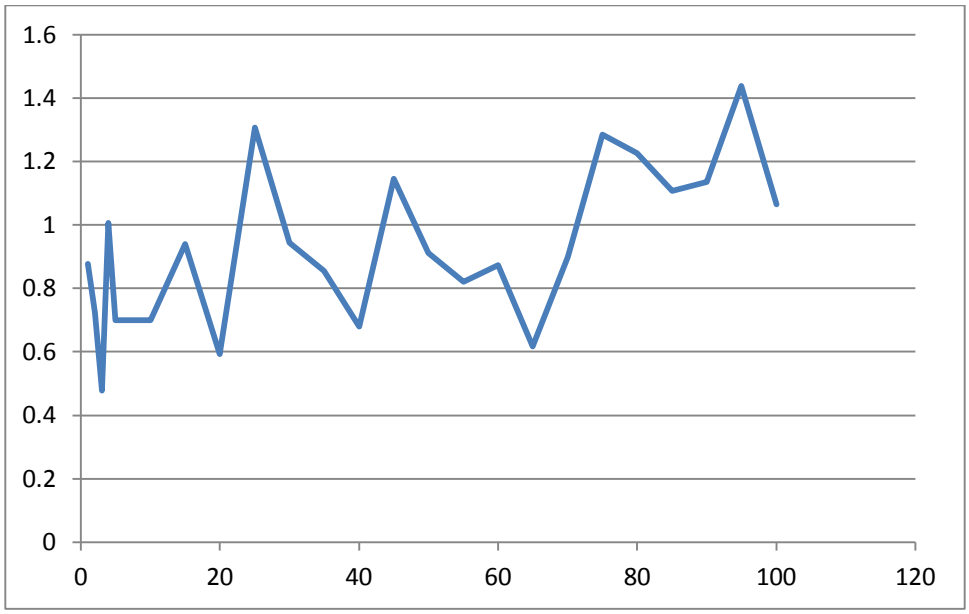
29.20	70	10000	0.900	2.96
29.27	75	10000	1.284	2.91
29.97	80	10000	1.227	2.91
29.41	85	10000	1.107	2.75
29.42	90	10000	1.135	2.92
29.15	95	10000	1.438	2.92
29.96	100	10000	1.064	2.92



X-Axis: Metropolis temperature

Y-Axis: Average final cost after running algorithm for 10000 iterations

Figure 8: Curve denoting Average Final Costs over Metropolis Temperatures for an Initial Schedule of Cost 91.67



X-Axis: Metropolis temperature

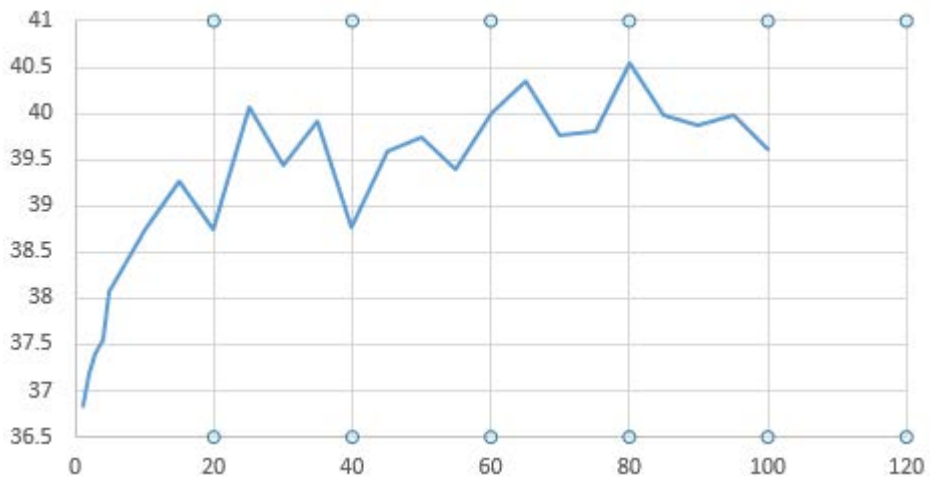
Y-Axis: Standard deviation in final costs after running algorithm for 10000 iterations

Figure 9: Curve denoting Standard Deviation in Final Costs over Metropolis Temperatures for an Initial Schedule of Cost 91.67

Table 19: Metropolis Algorithm Analysis for Schedule with Initial Cost 101.92

Average Final Cost	Temperature	Number of Iterations	Standard Deviation in Final Cost	Average Time Taken (seconds)
37.97	1	1000	1.07	0.25
39.61	2	1000	1.30	0.25
39.25	3	1000	1.14	0.25
40.38	4	1000	1.23	0.25
40.36	5	1000	1.42	0.25
42.17	10	1000	3.52	0.25
42.23	15	1000	2.66	0.25
42.52	20	1000	2.08	0.25
42.39	25	1000	2.20	0.25
43.13	30	1000	2.12	0.25
44.79	35	1000	4.78	0.25
43.88	40	1000	2.20	0.25
45.05	45	1000	4.15	0.25
44.99	50	1000	3.48	0.25

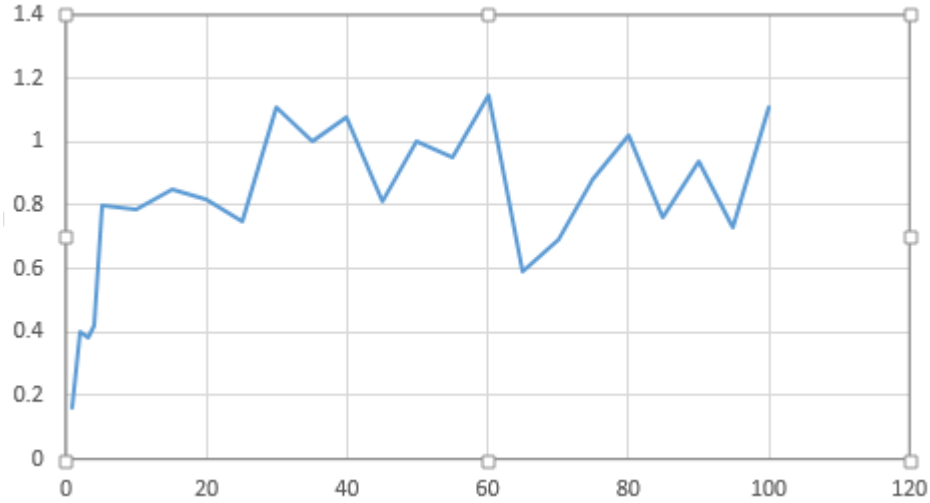
42.49	55	1000	1.89	0.25
43.01	60	1000	2.03	0.25
44.50	65	1000	1.24	0.25
44.81	70	1000	4.15	0.25
42.43	75	1000	2.69	0.25
42.24	80	1000	1.33	0.25
44.65	85	1000	2.63	0.25
45.39	90	1000	2.91	0.25
43.03	95	1000	1.65	0.25
45.69	100	1000	4.63	0.25
36.84	1	10000	0.16	2.49
37.19	2	10000	0.40	2.49
37.40	3	10000	0.38	2.49
37.55	4	10000	0.42	2.49
38.08	5	10000	0.80	2.49
38.74	10	10000	0.79	2.49
39.27	15	10000	0.85	2.49
38.74	20	10000	0.82	2.49
40.06	25	10000	0.75	2.51
39.44	30	10000	1.11	2.50
39.91	35	10000	1.00	2.51
38.76	40	10000	1.08	2.49
39.59	45	10000	0.81	2.49
39.75	50	10000	1.00	2.50
39.40	55	10000	0.95	2.56
40.01	60	10000	1.15	2.53
40.34	65	10000	0.59	2.52
39.76	70	10000	0.69	2.70
39.81	75	10000	0.88	2.53
40.54	80	10000	1.02	2.56
39.97	85	10000	0.76	2.56
39.88	90	10000	0.94	2.71
39.98	95	10000	0.73	2.88
39.61	100	10000	1.11	2.63



X-Axis: Metropolis temperature

Y-Axis: Average final cost after running algorithm for 10000 iterations

Figure 10: Curve denoting Average Final Costs over Metropolis Temperatures for an Initial Schedule of Cost 101.92



X-Axis: Metropolis temperature

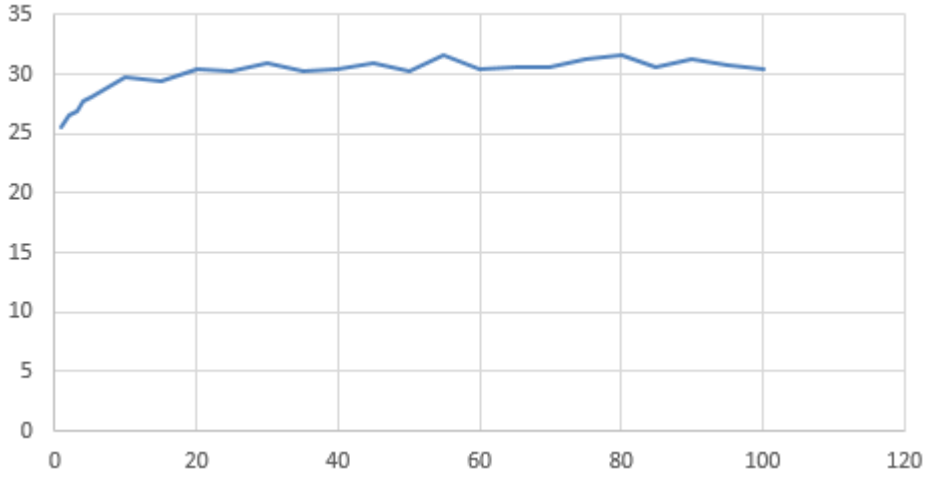
Y-Axis: Standard deviation in final costs after running algorithm for 10000 iterations

Figure 11: Curve denoting Standard Deviation in Final Costs over Metropolis Temperatures for an Initial Schedule of Cost 101.92

Table 20: Metropolis Algorithm Analysis for Schedule with Initial Cost 92.28

Average Final Cost	Temperature	Number of Iterations	Standard Deviation in Final Cost	Average Time Taken (seconds)
28.63	1	1000	1.90	0.26
30.35	2	1000	3.16	0.26
29.83	3	1000	1.76	0.26
33.42	4	1000	2.06	0.26
31.45	5	1000	1.75	0.26
32.30	10	1000	2.34	0.26
35.09	15	1000	2.99	0.26
35.52	20	1000	2.09	0.26
35.81	25	1000	2.18	0.26
35.58	30	1000	2.49	0.26
35.59	35	1000	2.48	0.26
36.15	40	1000	2.63	0.26
33.65	45	1000	2.14	0.26
35.02	50	1000	1.84	0.26
35.04	55	1000	2.66	0.26
34.48	60	1000	3.62	0.26
36.61	65	1000	3.10	0.26
35.44	70	1000	2.15	0.26
35.11	75	1000	2.54	0.26
36.83	80	1000	2.01	0.26
35.03	85	1000	2.40	0.26
36.49	90	1000	2.86	0.26
35.09	95	1000	3.13	0.26
34.46	100	1000	2.17	0.26
25.60	1	10000	0.12	2.59
26.55	2	10000	0.66	2.59
26.94	3	10000	0.49	2.59
27.75	4	10000	0.67	2.59
28.02	5	10000	1.18	2.60
29.78	10	10000	0.90	2.59
29.42	15	10000	1.20	2.67
30.40	20	10000	1.92	2.63
30.23	25	10000	1.13	2.63
30.92	30	10000	1.37	2.66
30.19	35	10000	1.17	2.64
30.47	40	10000	1.07	2.69
30.94	45	10000	0.82	2.67
30.24	50	10000	1.48	2.67
31.65	55	10000	1.83	2.59
30.46	60	10000	1.73	2.59
30.62	65	10000	1.24	2.66

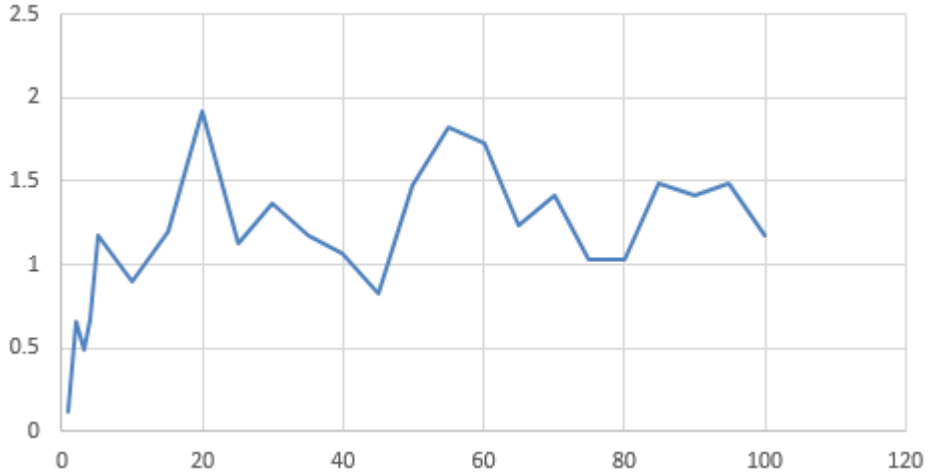
30.69	70	10000	1.42	2.88
31.24	75	10000	1.03	2.60
31.68	80	10000	1.03	2.60
30.66	85	10000	1.49	2.60
31.28	90	10000	1.42	2.67
30.74	95	10000	1.49	2.85
30.43	100	10000	1.17	2.86



X-Axis: Metropolis temperature

Y-Axis: Average final cost after running algorithm for 10000 iterations

Figure 12: Curve denoting Average Final Costs over Metropolis Temperatures for an Initial Schedule of Cost 92.28



X-Axis: Metropolis temperature

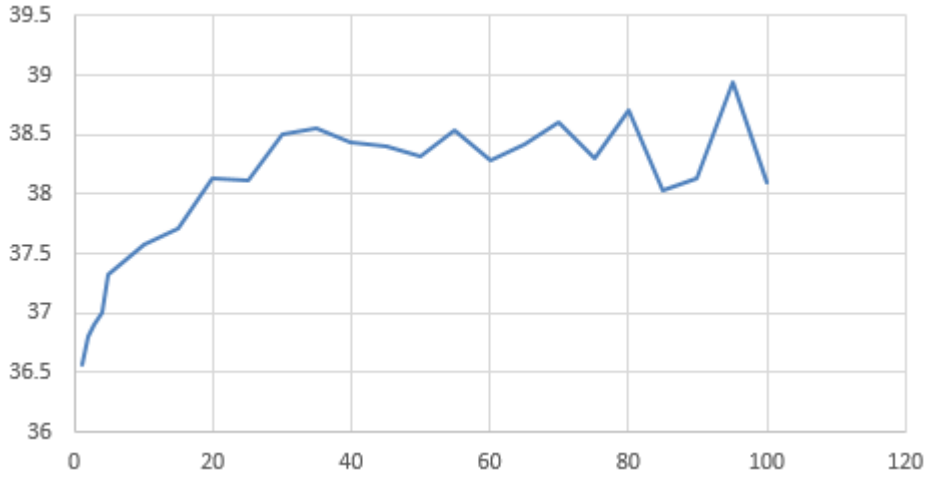
Y-Axis: Standard deviation in final costs after running algorithm for 10000 iterations

Figure 13: Curve denoting Standard Deviation in Final Costs over Metropolis Temperatures for an Initial Schedule of Cost 92.28

Table 21: Metropolis Algorithm Analysis for Schedule with Initial Cost 78.34

Average Final Cost	Temperature	Number of Iterations	Standard Deviation in Final Cost	Average Time Taken (seconds)
41.37	1	1000	4.35	0.25
38.63	2	1000	2.78	0.25
39.03	3	1000	2.45	0.25
40.76	4	1000	3.00	0.25
39.36	5	1000	1.44	0.25
40.40	10	1000	1.82	0.25
40.88	15	1000	1.75	0.25
41.80	20	1000	3.57	0.25
44.01	25	1000	3.54	0.25
43.38	30	1000	3.51	0.25
43.38	35	1000	4.38	0.25
41.69	40	1000	2.42	0.25
43.90	45	1000	3.23	0.25
43.81	50	1000	3.54	0.25
42.95	55	1000	4.52	0.25
43.41	60	1000	4.84	0.25
41.94	65	1000	3.10	0.25
43.09	70	1000	3.43	0.25
41.19	75	1000	3.03	0.25

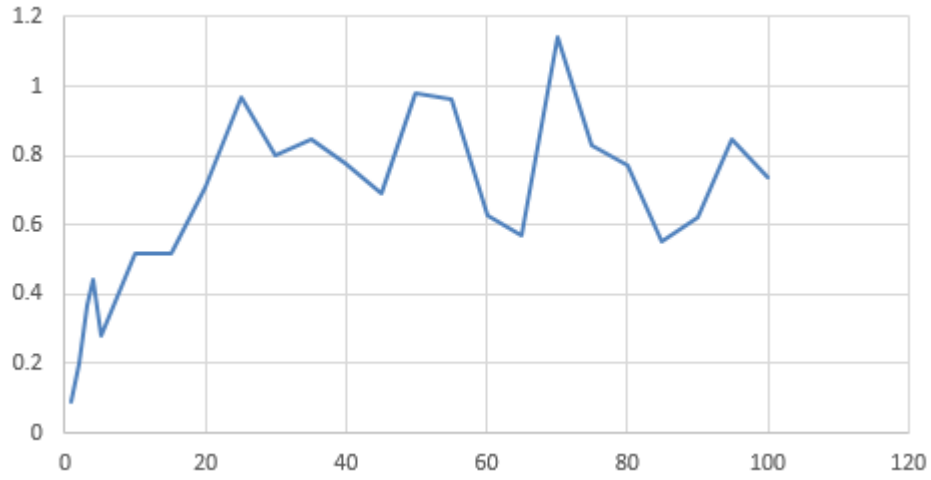
43.75	80	1000	3.55	0.25
43.65	85	1000	5.35	0.25
43.33	90	1000	4.20	0.25
41.95	95	1000	3.44	0.25
42.60	100	1000	3.95	0.25
36.57	1	10000	0.09	2.52
36.80	2	10000	0.20	2.52
36.90	3	10000	0.37	2.52
37.00	4	10000	0.44	2.52
37.33	5	10000	0.28	2.51
37.58	10	10000	0.52	2.51
37.71	15	10000	0.52	2.51
38.14	20	10000	0.71	2.51
38.12	25	10000	0.97	2.52
38.51	30	10000	0.80	2.52
38.56	35	10000	0.85	2.52
38.43	40	10000	0.78	2.53
38.40	45	10000	0.69	2.52
38.32	50	10000	0.98	2.52
38.53	55	10000	0.96	2.53
38.28	60	10000	0.63	2.52
38.42	65	10000	0.57	2.52
38.61	70	10000	1.14	2.52
38.31	75	10000	0.83	2.52
38.71	80	10000	0.77	2.52
38.03	85	10000	0.55	2.52
38.14	90	10000	0.62	2.52
38.95	95	10000	0.85	2.52
38.10	100	10000	0.74	2.52



X-Axis: Metropolis temperature

Y-Axis: Average final cost after running algorithm for 10000 iterations

Figure 14: Curve denoting Average Final Costs over Metropolis Temperatures for an Initial Schedule of Cost 78.34



X-Axis: Metropolis temperature

Y-Axis: Standard deviation in final costs after running algorithm for 10000 iterations

Figure 15: Curve denoting Standard Deviation in Final Costs over Metropolis Temperatures for an Initial Schedule of Cost 78.34

As can be observed from the analyses above of multiple scenarios, executing the Metropolis algorithm with a lower Metropolis temperature ~ 1 yields the best results, both in terms of the average final costs of schedules generated, and the standard deviations between final schedule costs.

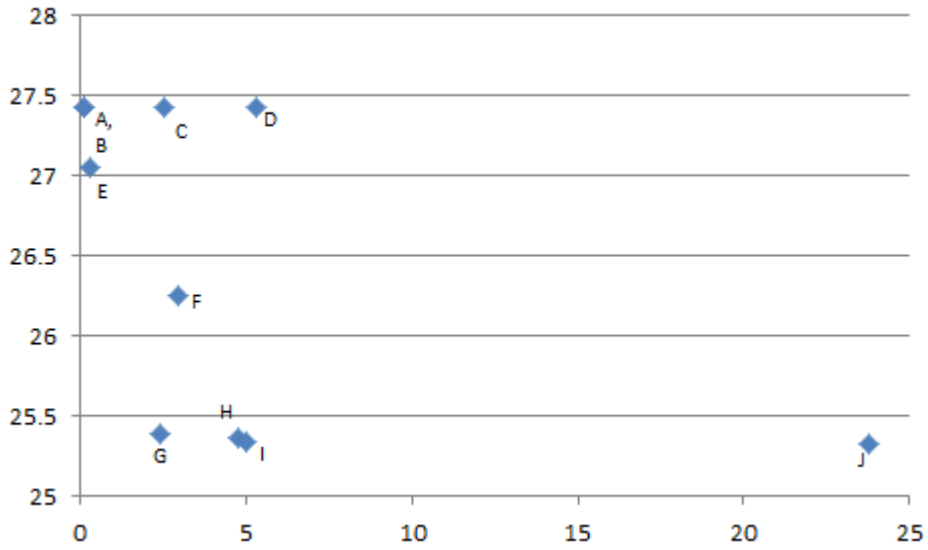
4.9. Comparison between heuristic algorithms

Through a study of the simulation results, it is observed that the Greedy and Tabu searches do not guarantee a good result (a low final cost) given an initial load profile. We note that randomized versions of both consistently yield much better results as compared to their deterministic counterparts. The randomized Greedy and randomized Tabu searches both have low standard deviations in final costs computed from an initial load profile, particularly in the case of the randomized Greedy search. As seen from the analysis of the Metropolis algorithm in the previous section, the algorithm also gives consistent results with low final costs computed, especially when utilizing lower Metropolis temperatures. The following table below summarizes results observed after running the aforementioned algorithms with a load profile of initial cost 91.67:

Table 22: Performance Comparison between Heuristic Algorithms for an Initial Schedule of Cost 91.67

Algorithm	No. of Iterations	Avg. Final Cost	Avg. Time (s)	Std. Dev. (Cost)
Greedy Search	1000	27.43	0.103	0
Greedy Search	10000	27.43	0.103	0
Randomized Greedy Search	1000	25.36	4.731	0.078
Randomized Greedy Search	5000	25.32	23.787	0
Tabu Search (Tabu List Size: 5)	500	27.43	2.51	0
Tabu Search (Tabu List Size: 5)	1000	27.43	5.31	0
Randomized Tabu Search	500	25.39	2.401	0.135
Randomized Tabu Search	1000	25.33	5.009	0.033
Metropolis (Temperature: 1)	1000	27.05	0.275	1.065
Metropolis (Temperature: 1)	10000	26.25	2.920	0.877

The graph below outlines the average final costs over the average time taken for the above algorithms (For the initial schedule of cost 91.678):



X-Axis: Average time taken

Y-Axis: Average final cost computed

A: Greedy search (1000 iterations)

B: Greedy search (10000 iterations)

C: Tabu search (500 iterations)

D: Tabu search (1000 iterations)

E: Metropolis algorithm (1000 iterations)

F: Metropolis algorithm (10000 iterations)

G: Randomized Tabu search (500 iterations)

H: Randomized Tabu search (1000 iterations)

I: Randomized Greedy search (1000 iterations)

J: Randomized Greedy search (5000 iterations)

Figure 16: Graph denoting Average Final Costs computed using Heuristic Algorithms over Time Taken for Computation for an Initial Schedule of Cost 91.67

Similar results are obtained when performing the above analysis on scenarios of initial costs 74.12, 101.92, 92.28 and 78.34. We can conclude that if the final schedule generated by the algorithm and the standard deviation in cost of schedules computed for the same initial schedule are the foremost criteria for evaluation of performances between algorithms, then the randomized Greedy search performs best. The randomized Tabu search yields results close to those of the randomized Greedy search (with slightly worse values of average costs of final schedule and standard deviation in final schedule costs), and executes faster. The Metropolis algorithm when executed with low Metropolis temperatures generates results close to (but not as good as) those generated by the randomized Greedy search and the randomized Tabu search, and executes faster than even the randomized Tabu search.

Chapter 5: Summary and Concluding Remarks

In this thesis, we have a proposed a real-time energy pricing policy and have provided algorithms for implementing the policy. A major issue for a power grid is to optimally use its capacity. Both the economic and the environmental costs will be different at periods in a twenty-four hour timeframe due to various reasons like variable availability of alternate energy sources, variability in personnel cost over various times of the day, and similar reasons. On the other hand, energy demands of customers also vary over the day; moreover, there are intervals in a day when a majority of the customers wish to consume energy and intervals where minimum energy consumption is observed. As a result, the overall demand seen through a load profile exhibits sharp peaks and valleys. If utilities aim only to meet demands of the customers at all times then generation capacity must be augmented through the establishment of special peaker plants which will be used only in the relatively short periods of large demands, and will remain idle at other periods. Also, in general, there may not be any positive correlation between periods of greater demands and periods when substantial supply of alternate energy is available. Overall, there can be large mismatches between the overall demand curve and the supply curve most suited for the utilities. Clearly, there is a need to balance overall demand so that these two curves are better aligned.

One way of achieving the balance mentioned above is to encourage customers to suitably adjust their energy demands over the twenty-four hour period. Every customer has a number of **flexible loads** for which there is a certain amount of flexibility as to when such loads are to be serviced in

the twenty-four hour period. It is then desirable that customers schedule their flexible loads in a manner that will help achieve an overall demand curve which will match as much as possible the ideal supply curve. Clearly, we need to provide economic incentives to customers so that they schedule their flexible loads in this manner. A simple-minded solution would be to have higher energy prices during peak demand hours and correspondingly lower prices during periods when the load profile exhibits valleys. But simulation studies have shown that this strategy does not work, it merely tends to shift a peak in demand from one period to another; the load on the grid still remains far from being balanced.

Our pricing policy, presented in Chapter 2, addresses the above problem. Suppose the twenty-four hour period is divided in M equal intervals. Our pricing policy is that for the interval i, the price is $a_iD_i + b_i$, where D_i is the total demand in the interval i, and a_i and b_i are constants which will essentially reflect the economic and environmental costs of energy generation at the i^{th} interval. As demand increases in a certain interval, the price also increases linearly, thereby discouraging further demand. Similarly, the pricing policy will encourage greater usage in intervals when demands are less. We show how the marginal cost pricing principle leads to such a pricing mechanism.

Apart from economic justifications, it is important for the aforementioned pricing policy to promote load balancing. For each interval i, the utility can specify what the constants a_i and b_i are for the price to be charged at that interval. Assuming a total demand of power on the utility being P, and considering M intervals in a twenty-four hour period, the question then arises how

best to distribute the total demand P into the M periods. We provide in Chapter 3 two algorithms for solving the problem: one is a dynamic programming algorithm which will return the optimal solution, and the second is a greedy algorithm which will return a non-optimal solution. The greedy algorithm, being only marginally poorer in terms of quality of solutions that it returns compared to the optimum-returning dynamic programming algorithm, has the advantage that it runs much faster than the dynamic programming algorithm. Indeed, its runtime complexity is square-root of the runtime complexity of the dynamic programming algorithm. We also discuss in Chapter 3 how the algorithm above can suggest an ideal load curve for each individual customer. We propose two policies: in the first, the total revenue paid by a customer is in proportion to his usage in terms of the total usage. This policy, as we have observed in Chapter 3, is fair to the utility but provides an advantage to customers with larger demand in comparison to those with smaller demands. We, therefore, have proposed another policy in which each customer pays for the cost of producing the amount of electricity equal to his demand. Although the second policy is fair to each customer, it will not be so for the utility. A solution is for the utility to provide to customers coefficients a_i and b_i suitably marked up.

Once each customer has an ideal load curve, the issue then is for him to schedule his flexible loads in a manner that his actual load profile comes as close to the ideal load curve as possible. We address this problem in Chapter 4. As the scheduling problem is NP-hard, we explore a number of heuristic search algorithms for solving the problem. The heuristic searches that we have implemented are: local search, Metropolis algorithm, Tabu search and a new randomized Tabu search. As we find in Section 4.8, Greedy and Tabu searches do not guarantee a good result (a low final cost) given an initial load profile. We note that randomized versions of both

consistently yield much better results as compared to their deterministic counterparts. The randomized Greedy and randomized Tabu searches both have low standard deviations in final costs computed from an initial load profile, particularly in the case of the randomized Greedy search. As seen from the analysis of the Metropolis algorithm in Section 4.7.4, the algorithm also gives consistent results with low final costs computed, especially when utilizing lower Metropolis temperatures.

We identify two issues as possibilities for future work. The first issue is that of exploring the possibility of co-operation amongst customers or amongst groups of customers. This work provides a method to determine an ideal load curve for each customer, followed by the customer seeking to schedule his flexible demands in a way so as to conform to the ideal load curve. However, for a given time period, a user may be unable to limit his usage at certain intervals within the values specified by his ideal load curve, while a neighboring user's demand may be less than what is specified by the neighbor's ideal load curve. A mechanism can be explored wherein, transparent to the utility, the excess energy of the neighbor may be used to compensate the user's shortfall. In effect, such a mechanism will help flatten out undesirable peaks in individual load profiles. This idea can be extended to encompass communities as well, where there may be cooperation between neighboring communities. It can be interesting to explore the kind of hardware and software support necessary to affect such levels of cooperation.

Another open issue with regard to this study is: we have assumed that no flexible demand can be preempted- once a flexible demand is scheduled, it is serviced for the length of its duration.

However, there are flexible jobs which can be preempted, for instance, a deep freezer. Therefore, the algorithms of Chapter 4 can be extended to allow pre-emptive scheduling for a specified subset of the flexible jobs.

References

1. Federal Energy Regulatory Commission, “Assessment of Demand Response & Advanced Metering Staff Report”, 2013

2. Renewable Energy Policy Network for the 21st Century, “Renewables 2007: Global Status Report”, 2007

3. U.S. Department of Energy, “The Smart Grids: An introduction”, 2009

4. Gang Xiong, Chen Chen, Shalinee Kishore, “Smart (In-home) Power Scheduling for Demand Response on the Smart Grid”, 2010

5. Pennsylvania Act 129, 2008. http://www.puc.state.pa.us//electric/Act_129_info.aspx

6. Frontier Economics and Sustainability First, “Demand Side Response in the domestic sector- a literature review of major trials”, 2012

7. Gelen Barbose, Charles Goldman, Bernie Neenan, “A Survey of Utility Experience with Real Time Pricing”, Berkeley National Laboratory, 2004. Available at http://eetd.lbl.gov/ea/EMS/EMS_pubs.html

8. Robert H. Miller and James H. Malinowski, “Power System Operation”, McGraw-Hill Edition, 2009

9. R. Caldon, A. Rossi Patria and R. Turri, “Optimal Control of a Distribution System with a Virtual Power Plant”, Proceedings of Bulk Power System Dynamics and Control Conference, 2004

10. Frank A. Wolak, “Public Utility Pricing and Finance”, The New Palgrave Dictionary of Economics, Second Edition. Eds. Steven N. Durlauf and Lawrence E. Blume, Palgrave

Macmillan, 2008. http://www.dictionaryofeconomics.com/article?id=pde2008_P000358

11. R.H. Coase, "The Theory of Public Utility Pricing and its Application", The Bell Journal of Economics and Management Science, Volume 1

12. Woodbank Communications Ltd, “Electricity Demand”, 2005

13. Market Surveillance Administrator, MSA Report on Residential Load Profiles, 2004

Vita

Manek Biswas was born in Kanpur, India on March 9, 1990. He completed his undergraduate studies in Computer Science from Manipal University in Karnataka, India in May 2011. Currently, he is enrolled in a Master's program in the same field at Syracuse University in New York, set to graduate in December 2013. While at Syracuse University, he was a Research Assistant working on the subject of the Smart Grid, and was also a Teaching Assistant for an introductory course to the topic.