Data-Driven Approaches to Load Modeling and Monitoring in Smart Energy Systems

by

Guoming Tang
B.Eng., National University of Defense Technology, 2010
M.Eng., National University of Defense Technology, 2012

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Computer Science

© Guoming Tang, 2017
University of Victoria

All rights reserved. This dissertation may not be reproduced in whole or in part, by photocopying or other means, without the permission of the author.
Data-Driven Approaches to Load Modeling and Monitoring in Smart Energy Systems

by

Guoming Tang
B.Eng., National University of Defense Technology, 2010
M.Eng., National University of Defense Technology, 2012

Supervisory Committee

Dr. Kui Wu, Supervisor
(Department of Computer Science)

Dr. Alex Thomo, Departmental Member
(Department of Computer Science)

Dr. Wu-Sheng Lu, Outside Member
(Department of Electrical and Computer Engineering)
Supervisory Committee

Dr. Kui Wu, Supervisor
(Department of Computer Science)

Dr. Alex Thomo, Departmental Member
(Department of Computer Science)

Dr. Wu-Sheng Lu, Outside Member
(Department of Electrical and Computer Engineering)

ABSTRACT

In smart energy systems, load curve refers to the time series reported by smart
meters, which indicate the energy consumption of customers over a certain period
of time. The widespread use of load curve (data) in demand side management and
demand response programs makes it one of the most important resources. To capture
the load behavior or energy consumption patterns, load curve modeling is widely
applied to help the utilities and residents make better plans and decisions. In this
dissertation, with the help of load curve modeling, we focus on data-driven solutions
to three load monitoring problems in different scenarios of smart energy systems,
including residential power systems and datacenter power systems and covering the
research fields of i) data cleansing, ii) energy disaggregation, and iii) fine-grained
power monitoring.

First, to improve the data quality for load curve modeling on the supply side,
we challenge the regression-based approaches as an efficient way to load curve data
cleansing and propose a new approach to analyzing and organizing load curve data.
Our approach adopts a new view, termed portrait, on the load curve data by ana-
lyzing the inherent periodic patterns and re-organizing the data for ease of analysis.
Furthermore, we introduce strategies to build virtual portrait datasets and demon-
strate how this technique can be used for outlier detection in load curve. To identify
the corrupted load curve data, we propose an appliance-driven approach that particularly takes advantage of information available on the demand side. It identifies corrupted data from the smart meter readings by solving a carefully-designed optimization problem. To solve the problem efficiently, we further develop a sequential local optimization algorithm that tackles the original NP-hard problem by solving an approximate problem in polynomial time.

Second, to separate the aggregated energy consumption of a residential house into that of individual appliances, we propose a practical and universal energy disaggregation solution, only referring to the readily available information of appliances. Based on the sparsity of appliances’ switching events, we first build a sparse switching event recovering (SSER) model. Then, by making use of the active epochs of switching events, we develop an efficient parallel local optimization algorithm to solve our model and obtain individual appliances’ energy consumption. To explore the benefit of introducing low-cost energy meters for energy disaggregation, we propose a semi-intrusive appliance load monitoring (SIALM) approach for large-scale appliances situation. Instead of using only one meter, multiple meters are distributed in the power network to collect the aggregated load data from sub-groups of appliances. The proposed SSER model and parallel optimization algorithm are used for energy disaggregation within each sub-group of appliances. We further provide the sufficient conditions for unambiguous state recovery of multiple appliances, under which a minimum number of meters is obtained via a greedy clique-covering algorithm.

Third, to achieve fine-grained power monitoring at server level in legacy datacenters, we present a zero-cost, purely software-based solution. With our solution, no power monitoring hardware is needed any more, leading to much reduced operating cost and hardware complexity. In detail, we establish power mapping functions (PMFs) between the states of servers and their power consumption, and infer the power consumption of each server with the aggregated power of the entire datacenter. We implement and evaluate our solution over a real-world datacenter with 326 servers. The results show that our solution can provide high precision power estimation at both the rack level and the server level. In specific, with PMFs including only two nonlinear terms, our power estimation i) at the rack level has mean relative error of 2.18%, and ii) at the server level has mean relative errors of 9.61% and 7.53% corresponding to the idle and peak power, respectively.
Contents

1 Introduction 1
  1.1 Load Curve Data . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
  1.2 Load Modeling and Monitoring . . . . . . . . . . . . . . . . . . . . 2
  1.3 Research Objectives and Contributions . . . . . . . . . . . . . . . . 3
    1.3.1 Load Curve Data Cleansing . . . . . . . . . . . . . . . . . . . 4
    1.3.2 Energy Disaggregation . . . . . . . . . . . . . . . . . . . . . . 5
    1.3.3 Fine-Grained Power Monitoring . . . . . . . . . . . . . . . . . 7
  1.4 Dissertation Organization . . . . . . . . . . . . . . . . . . . . . . . 9

2 Data-Driven Load Curve Analysis 10
  2.1 Overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
  2.2 Inherent Properties of Residential Load . . . . . . . . . . . . . . . . 10
    2.2.1 Power Consumption Pattern of Appliances . . . . . . . . . . . 10
    2.2.2 Sparsity of Appliance Switching Events . . . . . . . . . . . . 11
  2.3 Hidden Corrupted Data in Individual Residential Load . . . . . . . . 13
  2.4 Periodic Pattern in Aggregated Residential Load . . . . . . . . . . . 14

Supervisory Committee ii
Abstract iii
Table of Contents v
List of Tables ix
List of Figures xi
Acknowledgements xv
Dedication xvi
2.5 Conclusions ......................................................... 16

3 Load Curve Data Cleansing ........................................ 17
  3.1 Overview ....................................................... 17
  3.2 Related Work .................................................. 17
    3.2.1 Regression-Based Methods .............................. 18
    3.2.2 Univariate Statistical Methods ....................... 18
    3.2.3 Data Mining Techniques ................................. 18
  3.3 Outlier Detection for Aggregated Residential Load ............ 19
    3.3.1 Portrait Data Definition ............................... 19
    3.3.2 Construction of Portrait Data .......................... 23
    3.3.3 Load Curve Data Cleansing ............................. 26
    3.3.4 Handling Non-Stationary Landscape Data ............... 29
    3.3.5 Experimental Evaluations ............................... 30
  3.4 Corrupted Data Identification for Individual Residential Load .... 37
    3.4.1 CDIP Definition ......................................... 37
    3.4.2 An Important Step Towards Solving CDIP ............... 39
    3.4.3 Sequential Local Optimization .......................... 41
    3.4.4 Experimental Evaluations ............................... 45
    3.4.5 Robustness Testing ...................................... 53
    3.4.6 Further Discussion ...................................... 56
  3.5 Conclusions .................................................. 56

4 Energy Disaggregation .............................................. 58
  4.1 Overview ..................................................... 58
  4.2 Related Work ................................................ 58
    4.2.1 Non-Intrusive Appliance Load Monitoring ............... 58
    4.2.2 Real-Time Appliance State Monitoring .................. 59
  4.3 A Universal Approach to Non-Intrusive Load Monitoring .......... 60
    4.3.1 Problem Representation and Definition .................. 61
    4.3.2 Sparse Switching Events Recovering .................... 63
    4.3.3 Parallel Local Optimization ............................. 64
    4.3.4 Experimental Evaluations ............................... 67
  4.4 A Scalable Approach to Semi-Intrusive Load Monitoring .......... 72
    4.4.1 Problem Representation and Definition .................. 72
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4.2</td>
<td>Semi-Intrusive Appliance Load Monitoring</td>
<td>74</td>
</tr>
<tr>
<td>4.4.3</td>
<td>Unambiguous State Recovery</td>
<td>78</td>
</tr>
<tr>
<td>4.4.4</td>
<td>Meter Deployment Optimization</td>
<td>81</td>
</tr>
<tr>
<td>4.4.5</td>
<td>Experimental Evaluation</td>
<td>84</td>
</tr>
<tr>
<td>4.5</td>
<td>SmartSaver: a Consumer-Oriented Web Service for Energy Disaggregation</td>
<td>93</td>
</tr>
<tr>
<td>4.5.1</td>
<td>Overview</td>
<td>93</td>
</tr>
<tr>
<td>4.5.2</td>
<td>Architecture and Functions</td>
<td>94</td>
</tr>
<tr>
<td>4.5.3</td>
<td>Implementation and User Cases</td>
<td>96</td>
</tr>
<tr>
<td>4.6</td>
<td>Conclusions</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>Fine-Grained Power Monitoring in Datacenters</td>
<td>101</td>
</tr>
<tr>
<td>5.1</td>
<td>Overview</td>
<td>101</td>
</tr>
<tr>
<td>5.2</td>
<td>Related Work</td>
<td>103</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Fine-Grained Power Monitoring</td>
<td>103</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Key Points of Server-Level Power Monitoring</td>
<td>104</td>
</tr>
<tr>
<td>5.3</td>
<td>NIPD: Overview and Rationale</td>
<td>105</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Why NIPD?</td>
<td>105</td>
</tr>
<tr>
<td>5.3.2</td>
<td>How to Develop NIPD?</td>
<td>106</td>
</tr>
<tr>
<td>5.4</td>
<td>Model Design for NIPD</td>
<td>108</td>
</tr>
<tr>
<td>5.4.1</td>
<td>Formal Definition</td>
<td>108</td>
</tr>
<tr>
<td>5.4.2</td>
<td>PMFs Modeling</td>
<td>109</td>
</tr>
<tr>
<td>5.4.3</td>
<td>PMFs Training</td>
<td>111</td>
</tr>
<tr>
<td>5.4.4</td>
<td>Adaptive PMFs Update</td>
<td>114</td>
</tr>
<tr>
<td>5.5</td>
<td>Implementation</td>
<td>118</td>
</tr>
<tr>
<td>5.5.1</td>
<td>Data Collection and Feature Elimination</td>
<td>119</td>
</tr>
<tr>
<td>5.5.2</td>
<td>Estimation of Idle Power</td>
<td>121</td>
</tr>
<tr>
<td>5.6</td>
<td>Evaluation</td>
<td>122</td>
</tr>
<tr>
<td>5.6.1</td>
<td>Experiment Configuration</td>
<td>122</td>
</tr>
<tr>
<td>5.6.2</td>
<td>Power Monitoring at Rack Level</td>
<td>124</td>
</tr>
<tr>
<td>5.6.3</td>
<td>Power Monitoring at Server Level</td>
<td>126</td>
</tr>
<tr>
<td>5.7</td>
<td>Conclusions</td>
<td>129</td>
</tr>
<tr>
<td>6</td>
<td>Conclusions and Future Work</td>
<td>130</td>
</tr>
<tr>
<td>6.1</td>
<td>Conclusions</td>
<td>130</td>
</tr>
</tbody>
</table>
6.2 Future Work ........................................... 131

A NP-Hardness Proof of CDIP and SSER 133
   A.1 NP-Hardness of CDIP ........................................ 133
      A.1.1 Preparation ........................................... 133
      A.1.2 d-CDIP is NP ........................................... 134
      A.1.3 d-CDIP is NP-Hard ....................................... 135
   A.2 NP-Hardness of SSER ......................................... 136

B Equation Transformation, Complexity Analysis and Proof in NIPDI 137
   B.1 Equation Transformation ................................... 137
      B.1.1 Transformation of Equation (5.12) .................... 137
      B.1.2 Transformation of Equation (5.13) .................... 138
   B.2 Proof of Remark 7 .......................................... 138
   B.3 PMFs Training Complexity ................................... 138

Bibliography 140
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 3.1</td>
<td>The characteristic vectors of portrait data of the first 10 hours in Fig. 3.2, compared with landscape data (unit: kWh)</td>
<td>22</td>
</tr>
<tr>
<td>Table 3.2</td>
<td>Performance on small-scale data: virtual portrait data cleansing vs. B-spline smoothing</td>
<td>32</td>
</tr>
<tr>
<td>Table 3.3</td>
<td>Performance on large-scale data: virtual portrait dataset (VPD) cleansing vs. B-spline smoothing</td>
<td>34</td>
</tr>
<tr>
<td>Table 3.4</td>
<td>Results of corrupted data identification on university facility data: appliance-driven approach vs. B-spline smoothing</td>
<td>47</td>
</tr>
<tr>
<td>Table 3.5</td>
<td>Results of corrupted data identification on household data ($w = 1$)</td>
<td>49</td>
</tr>
<tr>
<td>Table 3.6</td>
<td>Parameter settings for load data generation and corruption</td>
<td>50</td>
</tr>
<tr>
<td>Table 3.7</td>
<td>Results of corrupted data identification on synthetic data: appliance-driven method vs. B-spline smoothing</td>
<td>51</td>
</tr>
<tr>
<td>Table 3.8</td>
<td>Robustness tests with incorrect power ranges of appliances</td>
<td>55</td>
</tr>
<tr>
<td>Table 4.1</td>
<td>Powers information of appliances</td>
<td>69</td>
</tr>
<tr>
<td>Table 4.2</td>
<td>Accuracy and overhead of energy disaggregation, using Sparse Switching Event Recovering (SSER), Least Square Estimation (LSE) based integer programming and iterative Hidden Markov Model (HMM)</td>
<td>70</td>
</tr>
<tr>
<td>Table 4.3</td>
<td>Accuracy of Energy Disaggregation using SSER with inaccurate estimation on power deviation</td>
<td>71</td>
</tr>
<tr>
<td>Table 4.4</td>
<td>Deployment and performance results from the SIALM approach with real data</td>
<td>87</td>
</tr>
<tr>
<td>Table 4.5</td>
<td>Performance results from experiments with real data</td>
<td>87</td>
</tr>
<tr>
<td>Table 4.6</td>
<td>Parameter settings for power model and load data generation</td>
<td>88</td>
</tr>
<tr>
<td>Table 4.7</td>
<td>Performance results of appliances deployment optimization and energy disaggregation: NIALM (N) vs. SIALM (S)</td>
<td>89</td>
</tr>
</tbody>
</table>
Table 4.8 Performance results of energy disaggregation for sub-groups of 30 appliances using different disaggregation models

Table 4.9 Performance of SIALM approach with inaccurate estimation of power deviation

Table 4.10 Performance of SIALM approach with newly added appliances

Table 4.11 Performance of SIALM approach with sufficient condition violations

Table 5.1 Comparison of current fine-grained power monitoring solutions vs. our NIPD solution

Table 5.2 Configuration of Server Nodes

Table 5.3 State metrics collected using dstat tool

Table 5.4 Parameter settings of our experiments

Table 5.5 Workloads/benchmarks for NIPD evaluations

Table 5.6 Performance of linear and nonlinear PMFs for power estimations at rack level and server level
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Load curve of an individual residential house in Waterloo, Ontario, Canada, from 23/01/2011 to 13/02/2011, provided by Singh et al., University of Waterloo [1]</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>The three research problems with corresponding scenarios.</td>
<td>3</td>
</tr>
<tr>
<td>2.1</td>
<td>Power consumption information of a microwave specified in the user’s manual.</td>
<td>11</td>
</tr>
<tr>
<td>2.2</td>
<td>Energy consumption and appliances’ on/off switching events in a house over the course of a day [2]</td>
<td>12</td>
</tr>
<tr>
<td>2.3</td>
<td>An example showing hidden corrupted data generated with three appliances</td>
<td>14</td>
</tr>
<tr>
<td>2.4</td>
<td>Average energy consumption of 112 residential houses in US for one year from 01/04/2006 to 31/03/2007 (above) and data for one month from 01/08/2006 to 31/08/2006 (below), provided by Pacific Northwest National Laboratory [3]</td>
<td>15</td>
</tr>
<tr>
<td>3.1</td>
<td>Divide timeline into 31 pieces by 24 hours and reposition the pieces in parallel</td>
<td>21</td>
</tr>
<tr>
<td>3.2</td>
<td>Switch the view to portrait</td>
<td>21</td>
</tr>
<tr>
<td>3.3</td>
<td>Result of outlier detection from IQR-based virtual portrait data cleansing</td>
<td>33</td>
</tr>
<tr>
<td>3.4</td>
<td>Result of outlier detection from gamma distribution based virtual portrait data cleansing (7 VLDs)</td>
<td>34</td>
</tr>
<tr>
<td>3.5</td>
<td>Under-fitted B-spline smoothing ($df = 100$)</td>
<td>35</td>
</tr>
<tr>
<td>3.6</td>
<td>Over-fitted B-spline smoothing ($df = 200$)</td>
<td>35</td>
</tr>
<tr>
<td>3.7</td>
<td>Results of B-spline smoothing for large-scale data ($df = 100$)</td>
<td>36</td>
</tr>
<tr>
<td>3.8</td>
<td>Energy monitoring platform and appliances’ power ranges</td>
<td>45</td>
</tr>
<tr>
<td>3.9</td>
<td>One-week and one-day load data collected via the energy monitoring platform.</td>
<td>46</td>
</tr>
</tbody>
</table>
Figure 3.10 Result of corrupted data identification on university facility data with our appliance-driven method \((w = 1, \delta = 2)\); Estimated bounds denote the upper and lower power bounds based on current state vector; Corrupted degree indicates the value of the virtual appliance (Section 3.3.1.) .................................. 48

Figure 3.11 Result of corrupted data identification on university facility data with the B-spline smoothing method \((df = 258)\) .................................. 48

Figure 3.12 Result of corrupted data identification on synthetic data with our appliance-driven method \((w = 1, \delta = 5)\) .................................. 51

Figure 3.13 Result of corrupted data identification on synthetic data with B-spline smoothing method \((df = 160)\) .................................. 52

Figure 3.14 Identification of consecutive corrupted data with our appliance-driven method .................................. 53

Figure 3.15 Identification of consecutive corrupted data with B-spline smoothing method \((df = 100)\) .................................. 53

Figure 3.16 Fast recovery of estimated load starting from a random initial state .................................. 54

Figure 3.17 Difference between estimated state \(S_e\) and real state \(S_r\) .................................. 55

Figure 4.1 A sketch map to illustrate the concepts of active epoch and baseline power using three appliances .................................. 65

Figure 4.2 Energy monitoring platform, monitored appliances, and measuring devices .................................. 68

Figure 4.3 Actual and estimated energy contributions of each appliance to the total energy consumption for one-week time period. .................................. 71

Figure 4.4 A real example of one-day power consumption from a residential house and four types of split time windows. .................................. 77

Figure 4.5 The power range of \(A_2\) is covered by that of \(A_1\). Therefore, when \(A_2\) is actually on, there will be two optimal solutions, as the switching events of \(A_1\) and the switching events of \(A_2\) are indistinguishable. .................................. 79
Figure 4.6 The power range of $A_3$ is submerged by the power deviation of $A_1$. Therefore, when $A_1$ is on, the state switching events of $A_3$ only result in small power changes, and due to the objective function of (4.32), such small changes will be considered as part of power deviation of $A_1$. In other words, the switching events of $A_3$ will not be identified, resulting in an optimal solution sparser than the ground-truth.

Figure 4.7 Distributed appliances power monitoring platform at the fifth floor of ECS building, University of Victoria, Canada.

Figure 4.8 Real and estimated energy contributions of each appliance to the total consumption for one week.

Figure 4.9 Number of meters needed for unambiguous state recovery in SIALM and RTASM, respectively.

Figure 4.10 System architecture of SmartSaver.

Figure 4.11 Plug-in power meters, gateway and client application used for online load data collection.

Figure 4.12 Website screenshot: aggregated energy consumption display.

Figure 4.13 Website screenshot: request forms for online and offline analysis.

Figure 4.14 Website screenshot: energy disaggregation results.

Figure 5.1 Power distribution hierarchy of the IT facilities in a typical datacenter.

Figure 5.2 Classification of fine-grained power monitoring for datacenters.

Figure 5.3 Framework of non-intrusive power disaggregation over a datacenter.

Figure 5.4 Total system power consumption with respect to three categories of CPU at high-end working frequency [4]. As a power model to depict the relationship between CPU frequency and system power consumption, the quadratic function is fitting more tightly than the linear one.

Figure 5.5 On/off events captured during turning on/off servers in our datacenter consisting of 326 servers.
Figure 5.6 Power architecture, network topology and data collection modules in our experimental environment. Note that the rack-level power measurements provided by PDMMs are only used for validation purpose. .......................................................... 119

Figure 5.7 The overview of MRE from three PMFs along with the training data size (left) and that of a zoomed-in view when MRE ≤ 5.0% (right). .......................................................... 124

Figure 5.8 The estimated power of one rack with corresponding ground truth values: a global view (left) and a local view (right). .... 125

Figure 5.9 Disaggregating datacenter power: estimated power consumption vs. referred idle/peak power. ................................................. 126

Figure 5.10 Disaggregating rack power: estimated power consumption vs. referred idle/peak power. ................................................. 127

Figure A.1 An example showing the construction of $T$ with $G$ .... 135
ACKNOWLEDGEMENTS

I would like to thank:

my supervisor, Dr. Kui Wu, for continuously mentoring and supporting me during my Ph.D. study.

my family, for their unconditional and endless love.

my friends and collaborators, for the valuable advices and fun time together.

my committee members, Dr. Alex Thomo and Dr. Wu-Sheng Lu, for the precious comments.
DEDICATION

To my beloved wife, Iris Wu.
Chapter 1

Introduction

The emerging *smart grid* technology revolutionized traditional power grid with state-of-the-art information technologies in sensing, control, communications, data mining, and machine learning [5, 6]. Worldwide, significant research and development efforts and substantial investment are being committed to the necessary infrastructure to enable intelligent control of power systems (which we called *smart energy systems* in this dissertation), by installing advanced metering systems and establishing data communication networks throughout the grid. Consequently, power networks and data communication networks are envisioned to harmonize together to achieve highly efficient, flexible, and reliable power systems.

1.1 Load Curve Data

In smart energy systems, the *load curve* refers to the electrical load versus time and represents the electric energy consumption of an electrical system over a certain period of time. It is essentially a time series composed by *load curve data* that are sampled by smart meters in a certain frequency. As an example, the three-week load curve of a residential house is illustrated in Fig. 1.1. While load curve data plays an important role in big data applications, its value has not been fully explored. A recent news article appeared in Forbes [7] said, “*But for the most part, utilities have yet to realize the potential of the flood of new data that has begun flowing to them from the power grid, . . ., And in some cases, they may not welcome it.*” Yet, existing power grid is facing challenges related to efficiency, reliability, environmental impact, and sustainability. For instance, the low efficiency of current electric grid could lead to
8% of electric energy loss along its transmission lines, and the maximum generation capacity is in use only 5% of the time [6].

Figure 1.1: Load curve of an individual residential house in Waterloo, Ontario, Canada, from 23/01/2011 to 13/02/2011, provided by Singh et al., University of Waterloo [1].

1.2 Load Modeling and Monitoring

*Load modeling* is the process to capture the load behavior or energy consumption patterns from certain load curves and represent/formulate them with mathematical models, e.g., the derived regression coefficients in regression-based approaches. The resulted load curve models (also termed as *load profile*) is the core for various applications in smart grid, such as electricity settlement, load monitoring, and load forecast.

To lower risks and reduce losses in energy planning and decision-making, the utilities are trying their best to model the load curves and capture the load profiles as precisely as possible. This effort can bring profits to both the utilities and customers, e.g., by supporting the utilities to promote their demand response (DR) programs and facilitating the customers to participate the DR programs. Meanwhile, various critical *load monitoring* problems in smart energy systems are being studied by making use of load curve data.

We have the following observations when performing load modeling and monitoring in smart energy systems.

- For both power suppliers and consumers, they are much concerned about the quality of load curve data, as load curve data directly impact the precision
of power companies’ generation forecasting, as well as the accuracy of power consumers’ payment.

- Nowadays, more and more residents begin to care about their energy consumption and look for better strategies towards energy saving.

- Datacenters consume a significant portion of energy worldwide. Fine-grained power monitoring, which refers to power monitoring at the server level, is critical to the efficient operation and energy saving, whereas it is extremely challenging due to lack of power monitoring sensors.

1.3 Research Objectives and Contributions

Motivated by the aforementioned observations and demands in smart energy systems, we focus on three critical problems in this dissertation, covering the research domains of data cleansing, energy disaggregation, and fine-grained power monitoring, respectively. The three problems along with corresponding scenarios are shown in Fig. 1.2, which will be interpreted in detail by the following sections.

Figure 1.2: The three research problems with corresponding scenarios.
1.3.1 Load Curve Data Cleansing

Outlier Detection on the Supply Side

Due to the critical meaning of load curve data, its quality is of vital importance, especially for the supply side who relies on it for energy planning and decision making. Nevertheless, the load curve data collected by the utilities (e.g., a power company) is usually subject to pollution caused by many factors, such as communication failures, meter malfunctions, unexpected interruption or shutdown of power stations, unscheduled maintenance, and temporary closure of production lines [8]. Load curve data is called *corrupted* when it significantly deviates from its regular patterns or when some data items are missing. Due to its huge volume, it would be nearly impossible to manually identify the corrupted load curve data. Therefore, efficient, automatic methods are needed to solve the *load curve data cleansing* problem, i.e., to detect and fix corrupted data in load curve.

To help the utilities clean the load curve data, regression-based approaches have been developed to find out the outliers [8, 9, 10, 11]. Nevertheless, such methods are established by referring to empirical knowledge and relevant parameters are regulated manually based on domain knowledge of experts. They therefore easily lead to underestimation or overestimation. We challenge the regression-based approaches as an efficient way to load curve data cleansing and propose a new approach to analyzing load curve data. The method adopts a new view, termed *portrait*, on the load curve data by analyzing the inherent periodic patterns and re-organizing the data for ease of analysis. Then, we introduce algorithms to build the virtual portrait load curve data, and demonstrate how this technique can be used for load curve data cleansing. We evaluate our approach with real-world trace data, including a small-scale stationary dataset and a large-scale non-stationary dataset. The experimental results demonstrate that our approach is much more effective and efficient than existing regression-based methods over both small-scale and large-scale load curve data. This contribution has been published in [12].

Corrupted Data Identification on the Demand Side

On the demand side, the data quality of load curve has direct impact on customers’ electricity bills and their trust on the still nascent smart grid technology. It can also provide important information for home automation [13, 14]. Nevertheless, it is *unavoidable* that load curves contain corrupted data. As a concrete example,
according to the news reports [15, 16], some customers in the province of British Columbia, Canada, were baffled by energy bills that are more than double what they were charged before the smart meter installation. While the problem could be identified by common sense and certain agreement might be reached by good faith negotiations [15, 16], fixing the questionable bill is another head-scratching and embarrassing issue to the utility. As a response to customer complaints, the utility normally took remedy actions, such as replacing the smart meters or taking back the smart meters for lab testing [16]. Such a remedy, however, can hardly be effective.

To help the customers identify the corruption in their load curve data, we propose an appliance-driven approach that particularly takes advantage of information available on the demand side. Our appliance-driven approach considers the operating ranges of appliances that are readily available from users’ manual, technical specification, or public websites. It identifies corrupted data by solving a carefully-designed optimization problem. To solve the problem efficiently, we develop a sequential local optimization algorithm (SLOA) that practically approaches the original NP-hard problem approximately by solving an optimization problem in polynomial time. We evaluate our method using both real-world trace data and large-scale synthetic data. The results demonstrate that i) our identification approach can precisely capture corrupted data for the consumers, and ii) SLOA is resilient to inaccurate power range information or inaccurate power state estimation. This contribution has been published in [17].

1.3.2 Energy Disaggregation

Non-Intrusive Appliance Load Monitoring

Nowadays, more and more customers can access to their smart meter data showing the aggregated power readings of their houses. Is there any way we could distill such smart meter data to gain more knowledge? What if we could make use of such data and figure out the energy consumption for each appliance in our houses? Energy disaggregation, also known as non-intrusive appliance load monitoring (NIALM), aims to learn the energy consumption of individual appliances from their aggregated energy consumption values, e.g., the total energy consumption of a house. With accurate energy disaggregation, the house owner can i) learn how much energy each appliance consumes, ii) take necessary actions to save energy, and iii) participate in demand response programs. Furthermore, with smart meters broadly deployed in many coun-
tries, sufficiently high resolution of energy data can be collected, making it feasible to develop effective energy disaggregation solutions.

Due to its critical meaning, the NIALM problem has attracted more and more attention since 1980s and has become an important application domain in smart grid [18, 19, 20]. Recently, it has also drawn attention from both large electronics companies and small start-ups, such as Intel, Belkin, GetEmme, and Navetas. Although many methods have been tested for energy disaggregation, according to [19], no solutions work well for all types of household appliances. They either work poorly for new types of appliances or require complex machine learning method to learn appliances’ (latent) features. For example, in [21], extra equipments are needed to detect the activities of appliances based on high frequency electromagnetic interference (EMI).

To deal with the challenge of developing easy-to-use approaches to NIALM, we propose a simple, universal energy disaggregation model, only referring to the readily available information of appliances. Based on the sparsity of appliances’ switching events, we first build a sparse switching event recovering model. Then, we make use of the active epochs of switching events and develop a parallel local optimization algorithm to solve our model efficiently. In addition to analyzing the complexity and correctness of our algorithm, we test our method with the real-world trace data from a real-world energy monitoring platform that collects high-resolution power data from a group of household appliances. The results demonstrated that our method can achieve better performance than the state-of-the-art solutions, including the popular Least Square Estimation (LSE) methods and a recently-developed machine learning method using iterative Hidden Markov Model (HMM). This contribution has been published in [22], and its application to a consumer-oriented website can be found in [23].

**Semi-Intrusive Appliance Load Monitoring**

With a single point of measurement, NIALM on one hand simplifies the task of load monitoring, but on the other hand its accuracy suffers as the number of appliances increases. While large-scale, diverse appliance groups consisting of hundreds or thousands of appliances are common in commercial buildings, many NIALM approaches were developed for and validated with small-scale appliance groups, and their accuracy with large-scale appliance groups may be unclear. Recently, low-cost energy
meters can be plug-and-play and have become popular on market. With such low-cost meters, we can easily monitor the power consumption of a single appliance or the aggregated power consumption of a small group of appliances without changing existing electric circuitry in the building. As such, we see no convincing need to stick with a single point of measurement.

We explore the benefit of introducing low-cost energy meters for monitoring large-scale appliance groups. This effort is not targeted at providing a complete, universal solution to NIALM; instead it is to demonstrate, in theory and in practice, that the accuracy of NIALM can be improved significantly with a small number of extra meters. While this is intuitively true, an in-depth analysis is needed to better understand the tradeoff between the benefit and the metering overhead. In addition, design problems such as where and how many meters should be installed must be addressed.

For such an investigation, we propose a semi-intrusive appliance load monitoring (SIALM) approach to energy disaggregation for large-scale appliances. Instead of using only one meter, multiple meters are distributed in the power network to collect the aggregated load data from sub-groups of appliances. Based on a simple power model, we establish a sparse switching event recovering (SSER) model and propose a parallel optimization algorithm to recover appliance states from the aggregated load curve data. We further provide the sufficient conditions for unambiguous state recovery of multiple appliances, under which a minimum number of meters is obtained via a greedy clique-covering algorithm. Both real-world trace data and synthetic data are used to evaluate our solution. The results show that the SIALM approach can provide high-precision appliance states estimation and improve the accuracy of energy disaggregation with a small number of extra meters. This contribution has been published in [24].

1.3.3 Fine-Grained Power Monitoring

Deployed all over the world to host computing services and data storage, datacenters have become indispensable in the modern information technology (IT) landscape. With rapid expansion of datacenters in both number and scale, their energy consumption is increasing dramatically. To tackle the problem, more attention than ever has been paid to power management (PM) in today’s datacenters [25, 26]. Power monitoring is the foundation of power management. Fine-grained power monitoring,
which refers to power monitoring at the server level, is of particular importance. It facilitates the implementation of various power management strategies, such as power capping and accounting [27, 28], idle power eliminating [29], and even cooling control and load balancing [30]. A fine-grained power monitoring platform not only helps audit the total energy use of the datacenter but also continuously shows the real-time server-level power consumption. Such a platform can greatly help the datacenter operators to adjust their power management policies and explore potential benefits. Taking the cooling control as an example, the real-time feedback of server-level power distribution can provide important information to optimize the air flow and locate the thermal “hot spot’ in a datacenter, which refers to server input air conditions that are either too hot or too dry and may hamper the efficiency of the datacenter.

To measure the power consumption in datacenters, one solution is to use power measurement hardware. For instance, SynapSense [31] has developed power monitoring solutions using power clamps or intelligent power strips. The IBM PowerExecutive solution [32] installs dedicated power sensors on servers during manufacturing to provide real-time power information of individual servers. Despite the above solutions, many legacy or even most recent server systems used in datacenters, such as DELL PowerEdge M100e and IBM BladeCenter H series, are not equipped with power measuring units. In this case, it is inconvenient for datacenter operators to install extra power meters on racks and it is extremely hard and costly to assemble power meters to individual blade servers as they are highly compacted in racks. This difficult task is typically contracted out to companies specialized in datacenter power monitoring, such as NobleVision [33] and ServerTechnology [34], that combine special hardware and intelligent software for fine-grained power monitoring.

Due to the above difficulties and also for cost saving, we present a zero-cost, purely software-based solution to this challenging problem. We use a novel technique of non-intrusive power disaggregation (NIPD) that establishes power mapping functions (PMFs) between the states of servers and their power consumption, and infer the power consumption of each server with the aggregated power of the entire datacenter. The PMFs that we have developed can support both linear and nonlinear power models via the state feature transformation. To reduce the training overhead, we further develop adaptive PMFs update strategies and ensure that the training data and state features are appropriately selected. We implement and evaluate NIPD over a real-world datacenter with 326 servers. The results show that our solution can provide high precision power estimation at both the rack level and the server level.
In specific, with PMFs including only two nonlinear terms, our power estimation i) at the rack level has mean relative error of $2.18\%$, and ii) at the server level has mean relative errors of $9.61\%$ and $7.53\%$ corresponding to the idle and peak power, respectively. This contribution has been originally published in [35], and an extended version can be found in [36].

1.4 Dissertation Organization

Based on our observations and data-driven analysis of residential load curve, this dissertation solves three critical load monitoring problems in smart power systems: load curve data cleansing, energy disaggregation, and fine-grained power monitoring. The rest of this dissertation is organized as follow.

In Chapter 2, we analyze the inherent properties of both individual and aggregated residential loads. Specifically, the general power consumption pattern of household appliances, the temporary sparsity of their switching events, and the hidden corrupted data problem are investigated.

In Chapter 3, making use of the inherent periodic pattern of aggregated load curve and residential load generation rules, we develop novel approaches to load curve cleansing for aggregated residential load and individual residential load, respectively.

In Chapter 4, a universal non-intrusive load monitoring solution is developed for small-scale appliances, after which a scalable semi-intrusive load monitoring approach is proposed to energy disaggregation for large-scale appliances.

In Chapter 5, we introduce a zero-cost, purely software based power monitoring solution in legacy datacenters, which provides the datacenter administrators with fine-grained power consumption information and thus helps them make better decisions in datacenter power management.

In Chapter 6, we conclude the dissertation and propose future research in relevant research fields.
Chapter 2

Data-Driven Load Curve Analysis

2.1 Overview

Due to the large energy consumption of residential buildings, modeling their load curves has attracted more and more attention. Capturing the behaviors of such load curves and representing them with mathematical models have significant meanings in solving load monitoring problems. In this chapter, we analyze the inherent properties and show some insights to residential load curves, for both individual and aggregated loads, which will be utilized in solving later load monitoring problems.

2.2 Inherent Properties of Residential Load

2.2.1 Power Consumption Pattern of Appliances

In a typical household, the electricity energy is mostly consumed by the household appliances, such as fridge, television, microwave, and lighting bulbs. The power consumption pattern of an appliance shows the energy consumption value when it is turned on or in stand-by states. According to real-world observations, most appliances work under one or multiple modes, each with relatively stable but distinguished power values [18, 37, 19]. For example, a light bulb usually works steadily under one mode, while a hair dryer can work with four different modes (low/high + cooling/heating).

For a certain operating mode of an appliance, we have the following domain knowledge:
• The value of rated power\(^1\) can be accessible by referring to technical specifications from the vendors \([38]\), e.g., Fig. 2.1 shows the rated power information of a microwave with multiple operating modes, which is specified in the user’s manual.

• The value of power deviation\(^2\) can be easily evaluated from the power readings, e.g., using the plug-in power meters from \([39]\).

Figure 2.1: Power consumption information of a microwave specified in the user’s manual.

Based on the above power consumption pattern, we will develop a simple appliance power model to interpret the generation of load curve data from a group of appliances. This power model, as introduced in Chapter 3.4 and Chapter 4, will be applied in solving the corrupted data identification problem and energy disaggregation problem in individual residential houses.

2.2.2 Sparsity of Appliance Switching Events

It has been known that most appliances’ state (operating mode) switching events have the so-termed sparsity feature \([18, 40, 41]\). Fig. 2.2 shows an example of energy consumption and appliances on/off switching events in a typical house during one day.

From the figures, we have the following observations:

• As shown in Fig. 2.2-(a), i) for a short time interval, the number of state switching events for all appliances is quite small; ii) for the whole time interval, the

---

\(^1\)The rated power here refers to the mean value of real power consumption of an appliance under a certain operating mode, with unit of Watt.

\(^2\)The power deviation here refers to the maximum difference between the real power and rated power, with unit of Watt. Thus, the real power consumption of a running appliance with rated power \(p\) and power deviation \(\theta\) is bounded by \([p - \theta, p + \theta]\).
total number of state switching events is significantly smaller than the number of samples.

- Most switching events happened in a small number of time intervals, which we call \textit{active epochs} (refer to Chapter 4.3.3 for formal definition) and are illustrated with shaded windows in Fig. 2.2-(b).

The above sparsity feature of appliance switching events, as introduced in Chapter 4, will be incorporated in the optimization problem and used to build the \textit{sparse switching event recovery model} for energy disaggregation.
2.3 Hidden Corrupted Data in Individual Residential Load

As we have mentioned in Chapter 1.3.1, techniques of load data cleansing have been proposed to deal with the problem of corrupted load curve data [8]. Most existing load data cleansing methods are designed for the supply side, to help the utility companies find the corrupted data and protect their profits. From the supply side, the collected load data is usually aggregated data, i.e., the energy consumption of a billing unit such as a house or a commercial building. When performing data cleansing on the supply side, due to the difficulty of obtaining extra knowledge behind the aggregated load data, most existing approaches apply outlier detection methods, i.e., the data that deviates remarkably from the regular pattern is identified as corrupted data. Various assumptions about the data generation mechanism are required for outlier detection, but due to limited information, those assumptions are usually based on empirical knowledge or statistic features of the data. Such outlier detection methods are oblivious of appliances’ various energy consumption models and may not be accurate or fair to customers. Since they are appliance-oblivious, such methods suffer from a few important deficiencies.

For example, the regression-based outlier detection methods find statistical patterns of load data and claim the data significantly deviating from the patterns as corrupted data. Nevertheless, such resulted outliers are not necessarily corrupted data. In addition, without the knowledge of appliances’ energy consumption models, some “hidden” corrupted data is hard to detect. To be specific, the energy consumption of a group of appliances in a house or a building is a stochastic process. The stochastic feature makes it hard to establish a fixed pattern. Turning on/off any high-power appliance may lead to a steep change in load curve. Using appliance-oblivious data cleansing methods, the data generated under such a condition is likely to be captured as outliers.

As another example, appliance-oblivious methods cannot deal with “hidden” corrupted data. Fig. 2.3 shows an example of three appliances, $A_1$, $A_2$, and $A_3$, which have power ranges of $[2, 4]$, $[10, 12]$ and $[30, 32]$, respectively. The load data within some ranges such as $(4, 10)$, $(16, 30)$, and $(36, 40)$ cannot be generated by any combination of the three appliances. Nevertheless, such data may not be identified by existing outlier detection as corrupted data.

The above two examples directly motivate our work in [17]. Based on the easily-
available appliance knowledge, we develop an *appliance-driven approach* to corrupted data identification on the customer side. The new data cleansing technique will be introduced and validated in Chapter 3.4.

### 2.4 Periodic Pattern in Aggregated Residential Load

According to [42], the residential houses consumed 39% of the total electricity consumption of U.S. in 2010, which held the largest share among all types of buildings. Therefore, the load profiles of residential houses have attracted more attention from the utilities. In specific, the utilities are interested in the behaviors of *aggregated residential load* that is from hundreds or even thousands of residential houses.

Usually, the aggregated load curve data collected by utilities is arranged and organized in chronological order, i.e., the load curve data is strictly treated as a time series. As shown in Fig. 2.4, the hourly energy consumption of over one hundred residential houses was recorded for one year (8,760 hours) and displayed in the 2D Coordinate System, with *x*-axis representing the time and *y*-axis the load values (*kWh*). From the aggregated residential load, a clear periodic pattern can be observed.

We call such type of arrangement of load data as *landscape* data. Landscape data is easy to understand, but it poses several barriers to efficient analysis:
Figure 2.4: Average energy consumption of 112 residential houses in US for one year from 01/04/2006 to 31/03/2007 (above) and data for one month from 01/08/2006 to 31/08/2006 (below), provided by Pacific Northwest National Laboratory [3].

- First, in a short time window (say 1 to 2 hours), the correlation between time and the load values may be hard to capture due to two reasons: i) some random events may play a dominant role in electric load, and ii) it is hard to obtain a unified model to capture the local pattern, which may change over time.

- Second, in a relatively long time (say days), even though certain regular patterns of the load curve can be found, the load curve over time is nonlinear and may be too complicated to model with fixed parameters.

- Third, with landscape data, each sample is usually treated equally, making it difficult to effectively capture special behavioral features. For instance, the energy consumption for a cafeteria is low and stable when it is closed and high during breakfast and lunch times. In this sense, it would be better if load data could be treated differently during the former period (say from 7:00 pm to 7:00 am) and during the latter periods (say from 7:00 am to 9:00 am and from 11:00 am to 1:00 pm).

By analyzing the inherent periodic patterns and making use of Fourier analysis, we have adopted a new perspective (termed portrait data) and re-organized the load curve data as virtual portrait data [12]. The data transformation from landscape to portrait will be introduced in Chapter 3.3, in which we also demonstrate how this
technique can be used for effectively cleaning load curve data.

2.5 Conclusions

In this chapter, we introduced the inherent properties of both individual and aggregated residential load, e.g., the general power consumption pattern of household appliances and the sparsity of their switching events. Furthermore, the hidden corrupted data problem was also proposed for individual residential load. In the following two chapters, we will show that how these properties can be utilized to solve different load monitoring problems.
Chapter 3

Load Curve Data Cleansing

3.1 Overview

Accurate load curve data is important for the energy demand side as well as the supply side management [6, 5, 8]: for electric utilities, the analysis of load curve data plays a significant role in day-to-day operations, system reliability, and energy planning; for the energy consumers, load curve data provides them with abundant information on their daily and seasonal energy cost, helping them make timely response to save expense. Nevertheless, the data is subject to corruption caused by many factors, such as communication failures, meter malfunctions, unexpected interruption or shutdown in electricity use, unscheduled maintenance, and temporary close of production lines. Therefore, Load curve data cleansing has caught more and more attention recently.

In this chapter, we develop different solutions for i) the utilities to detect the outliers in the load curves aggregated by those from hundreds or thousands of houses, and ii) the residents of individual households to identify corrupted data in their load curves, respectively. Note that the solution to the latter problem cannot be applied for the former, as they serve different stake holders in the power grid market and the required input information is also different.

3.2 Related Work

So far, most related work considers the polluted data as outliers in load pattern and focuses on outlier detection.
3.2.1 Regression-Based Methods

Regression-based methods have been widely studied for outlier detection in time series [8, 9, 10, 11]. In [8], a non-parametric regression method based on B-spline and kernel smoothing was proposed and applied to identify polluted data. In [11], the residual pattern from regression models was analyzed and applied to construct outlier indicators, and a four-step procedure for modeling time series in the presence of outliers was also proposed. Greta et al. [10] considered the estimation and detection of outliers in time series generated by a Gaussian auto-regression moving average (ARMA) process, and showed that the estimation of additive outliers was related to the estimation of missing observations. The ARMA model was also utilized in [43, 44, 11, 45] as the basic model for outlier detection. In general, the regression-based methods are established on empirical knowledge and their parameters are regulated manually according to the domain knowledge of experts. As a result, such methods are subject to either underestimation or overestimation.

3.2.2 Univariate Statistical Methods

Since load curve data consists of one-dimensional real values, univariate statistical methods can deal with outliers in such dataset [46, 47, 48, 49]. Most univariate methods for outlier detection assume that the data values follow an underlying known distribution. Then, the outlier detection problem is transformed to the problem of finding the observations that lie in a so-called outlier region of the assumed distribution [49]. Even though those methods have been proved simple and effective, we may not always know the underlying distribution of the data. This is unfortunately true for load curve data, e.g., the distribution of the data shown in Fig. 1.1 is unknown.

3.2.3 Data Mining Techniques

In addition to the above methods, data mining techniques have also been developed to detect outliers, such as $k$-nearest neighbor [50, 51], $k$-means [52, 53], $k$-medoids [54], density-based clustering [55], etc. In general, these methods classify the observations with similar features, and find the observations that do not belong strongly to any cluster or far from other clusters. Nevertheless, most data mining techniques are designed for structured relational data, which may not align well for the need of outlier detection in load curve data. In addition, these methods are normally time
consuming because they need a training process on a large dataset.

3.3 Outlier Detection for Aggregated Residential Load

Based on the analysis in Chapter 2.4, we re-organize load curve data from landscape to portrait and demonstrate how this data transformation can facilitate outlier detection in aggregated residential load. The following contributions are made in this section:

- A new view, called *portrait*, is proposed for load data analysis. Switching perspective from landscape to portrait, some hidden behavioral patterns in the load data become prominent, such as the numerical stability of load curve data in the same hours of different days.

- With Fourier analysis, an algorithm is designed to *automatically* transform a landscape data to portrait data. We further extend the method to build *virtual portrait datasets*, meaning of which will be disclosed later, to address the problem in the third observation raised above.

- A data pre-processing method is proposed, so that non-stationary load data can be effectively handled with the help of virtual portrait datasets.

- Efficient algorithms are designed to use virtual portrait data for both small-scale and large-scale load data cleansing. Our experimental results show that our portrait based method is faster and more accurate, compared to the state-of-the-art regression-based methods.

3.3.1 Portrait Data Definition

We propose a new view of load curve data and organize them via a model of portrait data, which can facilitate the analysis and cleansing of load curve data.

**Portrait Data**

**Definition 1.** Consider a function $f(x)$ with periodic pattern defined over $[0, NT]$, and the period is $T$. We split one period of time $[0, T]$ into $n$ ($n \geq 1$) even slices, i.e.,
0 = x_0 < x_1 < x_2 \ldots < x_n = T. The **portrait data** of function \( f(x) \) corresponding to the \( i \)-th time slice \( (0 \leq i \leq n) \), denoted by \( p_i \), is defined as the dataset:

\[
p_i := \{ f(x) \mid x \in [x_i + kT, x_{i+1} + kT], 0 \leq k \leq N \}.
\] (3.1)

**Definition 2.** The **span** of a portrait data \( p_i \) is defined as

\[
sp_i := x_{i+1} - x_i.
\] (3.2)

Similarly, for discrete periodic load curve data with even spacing labeled as \( \{y(0), y(1), y(2), \ldots \} \), the portrait data are composed with the data points falling within the corresponding time intervals, i.e., the portrait data \( p_i \) is constructed as:

\[
p_i := \{ y(t) \mid t = t_i + kT, 0 \leq k \leq N, 0 \leq i \leq n \},
\] (3.3)

where \( N \) is the total number of periods and \( n \) is the number of data points within one period.

**Example of Portrait Data**

To help better understand the portrait data, we use the one-month load curve data in Fig. 2.4 as an example to illustrate portrait data visually.

Noticing that the data exhibits a periodicity of 24 hours, we divide the original time line by 24 hours into 31 slices (days) and re-arrange the slices in parallel. In this way, we transform the 2D landscape data into 3D space, with \( x \)-axis representing hours, \( y \)-axis days, and \( z \)-axis the load values, as shown in Fig. 3.1. To view the energy consumption of each hour in the 31 slices, we rotate the figure in the \( x-y \) coordinate by 90 degrees, and re-draw the data into 24 slices. Each slice represents a portrait data consisting of the energy consumption at the same hour in the 31 days, as shown in Fig. 3.2. Immediately, we can observe that: the values in each portrait dataset are relatively stable.

**Characteristic Vector of Portrait Data**

Intuitively, a portrait dataset should include values with a very small variation. There are many ways to model this phenomenon, and we use the following characteristic vector to describe the portrait data.
Figure 3.1: Divide timeline into 31 pieces by 24 hours and reposition the pieces in parallel

Figure 3.2: Switch the view to portrait

Definition 3. The characteristic vector of portrait data $p_i$ is defined as

$$e_i := [\theta_i, M_i],$$ (3.4)

where $\theta_i$ and $M_i$ represent the median and the median absolute deviation (MAD) of values in $p_i$, respectively.

Since the data may be contaminated by outliers, we use the median and MAD instead of mean and standard deviation to represent central tendency and statistical
dispersion of a portrait dataset, respectively. The median and MAD are more robust measures [56].

For the data in Fig. 3.2, the characteristic vectors of portrait data of the first 10 hours are summarized in Table 3.1. The last column of the table shows that the MAD value of landscape data is significantly higher. The results indicate that each portrait dataset is much more stable than the landscape data.

Table 3.1: The characteristic vectors of portrait data of the first 10 hours in Fig. 3.2, compared with landscape data (unit: kWh)

<table>
<thead>
<tr>
<th>Hour</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>1-10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\theta$</td>
<td>0.79</td>
<td>0.78</td>
<td>0.77</td>
<td>0.84</td>
<td>0.99</td>
<td>1.30</td>
<td>1.69</td>
<td>1.76</td>
<td>1.69</td>
<td>1.60</td>
<td>1.14</td>
</tr>
<tr>
<td>$M$</td>
<td>0.04</td>
<td>0.01</td>
<td>0.02</td>
<td>0.04</td>
<td>0.05</td>
<td>0.08</td>
<td>0.09</td>
<td>0.07</td>
<td>0.09</td>
<td>0.11</td>
<td>0.42</td>
</tr>
</tbody>
</table>

**Definition 4.** The similarity of two portrait datasets $p_i, p_j$ with characteristic vectors $e_i, e_j$, respectively, is defined as

$$s_{ij} := \begin{cases} \infty, & \text{if } e_i = e_j \\ 1/\|e_i - e_j\|_2 & \text{otherwise.} \end{cases}$$

We can develop heuristic algorithms to merge multiple portrait datasets with a high similarity into a virtual portrait dataset, which will be introduced in detail in Section 3.3.2.

**Properties of Portrait Data**

Compared to landscape data, the portrait data has the following desirable properties:

- The data values within the same portrait are similar and can be processed together even if they are separated in the original time domain.

- The data values within the same portrait dataset can be captured with a simple model, for which numerous fast data cleansing methods can be applied. In contrast, landscape data is normally nonlinear and requires complicated nonlinear regression-based methods.

- With portrait data, users’ behavioral patterns in different time periods can be modeled. In Fig. 3.2, the energy consumption in the first hour of each day is quite stable and low, but the situation for the seventh hour is quite different.
As such, a data point with small deviation in the first slice should be captured as an outlier, but may be considered regular in the seventh slice. In this way, we can improve the accuracy of outlier detection.

It is worth noting that portrait data is not just a data visualization trick. It is helpful to design efficient algorithms for load curve data analysis and cleansing. In specific, due to the stability in each portrait data, it is much easier to build simple models to capture the outliers. In addition, by combining similar portrait slices into one virtual slice, we can build virtual portrait dataset, which further speeds up data processing.

### 3.3.2 Construction of Portrait Data

**Compute Time Period of Landscape Data**

In order to automatically construct portrait data, we need to find out the time period of landscape load curve data. In our daily life, the energy consumption of different houses or buildings is usually periodic, either hourly, daily or weekly. When the volume of landscape data is big, an automatic method is needed to quickly discover the periodic behaviour hidden in the landscape data. Therefore, Fourier analysis [57] is used for this purpose.

According to Fourier transform, given a non-sinusoidal periodic function:

\[ f(t) = f(t + kT), k = 0, 1, 2, \cdots, \]  

(3.6)

if in one cycle of the periodic function there are finite maximum and minimum values, as well as the finite number of first category discontinuous points\(^1\), the function can be unfolded into a convergent Fourier Series, i.e.,

\[ f(t) = A_0 + A_1 \cos(\Omega t + \psi_1) + \sum_{k=2}^{\infty} A_k \cos(k\Omega t + \psi_k), \]  

(3.7)

where \(A_0\) is called the constant component and \(A_1 \cos(\Omega t + \psi_1)\) the fundamental component. The frequency of the fundamental component discloses the lowest frequency in the original function \(f(t)\), which can be used to construct the portrait data.

\(^1\)A discontinuous point \(x\) is called the first category discontinuous point where there exist finite limits from the left \(f(x - 0)\) and from the right \(f(x + 0)\) for \(f\).
Since the load curve data is discrete, we should use another form for Fourier transform, Discrete Fourier Transform (DFT), to convert a finite list of equally-spaced samples of a function into the list of coefficients of a finite combination of complex sinusoids, ordered by their frequencies. To speed up the process, Fast Fourier Transform (FFT) is adopted, which is developed upon DFT and works much faster.

In practice, the sampling interval for residential energy consumption on the utility side is normally 15 minutes [58]. Considering the periodic pattern in load curve is relatively longer, such as one day (24 hours), the sampling rate is high enough to acquire the time period of load curve data.

**Construct Basic and Virtual Portrait Data**

Having identified the periods in landscape data, the next step is to decide how many slices of portrait data should be split.

One solution is to split the load curve data with the span of sampling interval, which will result in portrait data with the highest resolution. However, since the sample rate may be significantly high, such kind of splitting may result in too many portrait data slices. Considering that the characteristic vectors of some portrait datasets are similar, we merge them together into a virtual portrait dataset to speed up data cleansing. Therefore, a two-phase method is developed.

**Build Basic Portrait Datasets:** The portrait datasets obtained in this phase are called basic portrait dataset (BPD). With FFT, the fundamental period of load curve data can be obtained. Assuming that there are \( r \) samples in one period, we can obtain \( r \) basic portrait datasets \( \{p_0, p_1, \cdots, p_r\} \). Accordingly, we can calculate the characteristic vector of each basic portrait dataset, denoted by \( \{e_0, e_1, \cdots, e_r\} \), respectively.

**Build Virtual Portrait Datasets:** We merge multiple basic portrait datasets with similar characteristic vectors into one virtual portrait dataset (VPD). As such, a clustering algorithm is needed to partition the basic portrait datasets into exclusive clusters such that within each cluster, the pairwise similarity of basic portrait datasets is no less than a given threshold. In order to accelerate data analysis, it is desirable to minimize the total number of clusters. This optimization problem can be formulated as follow:

- **Input:** Basic portrait data \( \{p_1, p_2, \cdots, p_r\} \) and their corresponding characteristic vectors \( \{e_1, e_2, \cdots, e_r\} \). A given threshold \( s_0 \) on similarity.
• **Output:** Minimum number of virtual portrait datasets, denoted by \( \{P_1, P_2, \cdots, P_n\} \) such that within each virtual portrait dataset, the pairwise similarity of the basic portrait datasets is no less than \( s_0 \).

\[
\begin{align*}
\text{minimize} & \quad n \\
\text{s.t.} & \quad \bigcup P_i = \{p_1, p_2, \cdots, p_r\} \\
& \quad P_i \cap P_j = \emptyset, i \neq j \\
& \quad P_i = \{\{p_{i_1}, p_{i_2}, \cdots, p_{i_m}\} \mid s_{l_i l_i} \geq s_0\} \\
& \quad 1 \leq i, j \leq n; 1 \leq m \leq r; 1 \leq l_i, l_i \leq m
\end{align*}
\]  

(3.8)

In order to solve the above problem, a graph \( G = (V, E) \) is constructed, where each vertex \( v \in V \) represents a BPD and an edge is built between two vertices if their similarity is no less than \( s_0 \). It is easy to see that the problem is equivalent to the clique-covering problem, which has been proven to be NP-complete [59]. Hence, a greedy clique-covering algorithm is adopted to obtain an approximate solution. Algorithm 1 shows the pseudo code of the greedy clique-covering problem.

---

**Algorithm 1 Greedy Clique-Covering Algorithm**

**Input:** Graph \( G = (V, E) \)

**Output:** A set of cliques \( P \) that completely cover \( G \)

1. Initialize uncovered vertex set \( V' \leftarrow V \)
2. Initialize number of cliques, \( n = 0 \)
3. while \( V' \neq \emptyset \) do
4. \( n = n + 1 \)
5. Find \( v \in V' \) with the highest node degree
6. Find \( U \subseteq V' \) with \( u \in U \) and \( (u, v) \in E \)
7. Construct subgraph \( G' = (U, D) \) where \( U \) includes all vertices adjacent to \( v \), and \( D \) includes the associated links
8. Initialize clique \( P_n = \{v\} \)
9. for each \( w \in U \) do
10. \quad if \( w \) is adjacent to all vertices in \( P_n \) then
11. \quad \( P_n \leftarrow P_n \cup \{w\} \)
12. \quad end if
13. end for
14. \( V' \leftarrow V' \setminus P_n \)
15. end while
16. return \( P_1, P_2, \cdots, P_n \)
The basic idea of the algorithm is to find cliques that cover more vertices that have not been clustered. Heuristically, the vertices with larger degrees may have a better chance of resulting in a smaller number of cliques. Thus, the search starts from the vertex with the largest degree, until all vertices are covered. Obviously, a resulted cluster is a clique in the graph. Since each vertex represents a BPD, a clique represents a VPD.

**Lemma 1.** The computational complexity of Algorithm 1 is lower bounded by $O(r \log r)$ and upper bounded by $O(r^2 \log r)$, where $r$ is the number of basic portrait datasets.

**Proof.** Since the similarity of two basic portrait datasets is calculated with their characteristic vectors consisting of two values, the graph $G$ in Algorithm 1 is actually a geometric graph in the 2D plane. Any clique resulted from Algorithm 1 can be bounded by some rectangle region in the 2D plane. According to [60], the largest clique of a rectangle intersection graph can be found with computational complexity no more than $O(r \log r)$. Since in Algorithm 1 the number of iterations in finding cliques could range from 1 to $r$, the computational complexity ranges from $O(r \log r)$ to $O(r^2 \log r)$. □

### 3.3.3 Load Curve Data Cleansing

In this section, we show portrait data can help load curve data cleansing. Load curve data cleansing involves two phases: (1) detecting outliers and (2) fixing the missing or aberrant values in the dataset.

Formally, for a given distribution $F$, the outlier detection problem is to identify those values that lie in a so-called outlier region defined below:

**Definition 5.** For any confidence coefficient $0 < \alpha < 1$, the $\alpha$-outlier region of $F$ distribution with parameter vector $\Theta$ can be defined by

$$\text{out}(\alpha, \Theta) = \{x : x < Q_{\frac{\alpha}{2}}(\Theta) \text{ or } x > Q_{1-\frac{\alpha}{2}}(\Theta)\}, \quad (3.9)$$

where $Q_q(\Theta)$ is the $q$ quantile of function $F(\Theta)$.

Since we usually do not have apriori knowledge on the distribution of portrait data, various possible cases should be considered. Note that performing statistical test to find out the distribution of load curve data does not work well when the load data is polluted. We need to consider several potential cases for outlier detection.
Case 1: Outlier Detection for Normal Distributed Data

The normal distribution can be adopted as an empirical distribution, which has been proved to be effective in general situations [61].

According to Equation (3.9), for a normal distribution $N(\mu, \sigma^2)$, its $\alpha$-outlier region is

$$\text{out}(\alpha, (\mu, \sigma^2)) = \{x : |x - \mu| > \Phi_{1-\frac{\alpha}{2}}(\sigma)\},$$

(3.10)

where $\Phi_q$ is the $q$ quantile of $N(0, 1)$. For normal distributed portrait datasets $P_i, i = 1, 2, \cdots$, we claim that a value $x$ is an $\alpha$-outlier in $P_i$, if $x \in \text{out}(\alpha, (\hat{\mu}_i, \hat{\sigma}_i^2))$, where $\hat{\mu}_i$ and $\hat{\sigma}_i$ are unbiased estimators of $\mu_i$ and $\sigma_i$, respectively. Since the data may be contaminated by outliers, we use the median and MAD instead of mean and standard deviation in our later detection.

Case 2: Outlier Detection for Gamma Distributed Data

It has been shown that the aggregated residential load at a given time instant follows the gamma distribution [62, 63]. In this light, the gamma distribution is also a good candidate distribution for outlier detection.

According to Equation (3.9), for a gamma distribution with shape parameter $\beta$ and scale parameter $\gamma$, $G(\beta, \gamma)$, its $\alpha$-outlier region is

$$\text{out}(\alpha, (\beta, \gamma)) = \{x : x < F^{-1}_q(\beta, \gamma) \text{ or } x > F^{-1}_1-\frac{\alpha}{2}(\beta, \gamma)\},$$

(3.11)

where $F^{-1}$ is the inverse cumulative distribution function of $G(\beta, \gamma)$, and $F^{-1}_q(\beta, \gamma)$ is the $q$ quantile of $G(\beta, \gamma)$.

If we assume that virtual portrait datasets, $P_i, i = 1, 2, \cdots$ follow a gamma distribution $G(\beta, \gamma)$, we can use (3.11) for outlier detection. In this case, $$(\hat{\beta}_i^2 / \hat{\sigma}_i^2) \text{ and } (\hat{\sigma}_i^2 / \hat{\mu}_i)$$ are the moment estimators of $\beta$ and $\gamma$, respectively. Due to the same reason as in Case 1, we use the median and MAD instead of mean and standard deviation in our later detection.

Case 3: Outlier Detection for Small-Size Portrait Data

In the above outlier detection strategies, the size of portrait datasets is assumed to be large. Otherwise, the parameter estimation may be inaccurate. When the size of samples is small, Tukey et al. [64] introduce a graphical procedure called boxplot to summarize univariate data.
The boxplot uses median and lower and upper quartiles (defined as the 25-th and 75-th percentiles). If the lower quartile is \( Q_1 \) and the upper quartile is \( Q_3 \), then the difference \( (Q_3 - Q_1) \) is called interquartile range or \( IQR \). After arranging data in order, the ones falling in the following outlier region are identified as outliers.

\[
\text{out}(\rho, (Q_1, Q_3)) = \{ x : x < Q_1 - \rho \cdot IQR \text{ or } x > Q_3 + \rho \cdot IQR \}, \tag{3.12}
\]

where \( \rho \) is an index of significance, and the outliers are said to be “mild” when \( \rho = 1.5 \) and “extreme” when \( \rho = 3 \).

Overall, the above three cases cover most situations a user may meet in portrait load data cleansing. Nevertheless, other strategies can also be chosen as long as they give more precise model of the portrait data.

### Replacing Missing Data or Aberrant Data

We mainly focus on outlier detection for two reasons.

1. Imputation of missing data can be easily done after we obtain the characteristic vector of portrait data, e.g., we could replace a missing value with the median of the corresponding portrait dataset.

2. Replacing aberrant values requires human interaction, since it needs the user to further confirm whether or not an outlier is a corrupted value. The user can either (1) replace the outlier with an acceptable value of the corresponding dataset, e.g., the mean value for Case 1 and Case 3 and the value of \( \beta/\gamma \) for Case 2, or (2) leave the outlier unchanged, as long as the cause of creating the outlier can be explained, such as the stimulation of holidays/special events.

Note that data imputation is normally carried out after outlier detection. It is common to initially set the missing data to default values of zeros, which are likely to be outliers and then are replaced with acceptable values. This strategy has been used in [58]. Nevertheless, the default value can be altered by the user according to different scenarios. For instance, if there exist valid load values close to zero, we can set the default value for missing data to a very large value, so that missing data can be identified easily as outliers.

If the user has explicitly learned the cause of the aberrant or missing data and found any above or other replacing approach that fits the needs well, the approach can be incorporated with our outlier detection approach and works automatically.
3.3.4 Handling Non-Stationary Landscape Data

The construction and cleansing of portrait data are based on the assumption that the landscape data are stationary along the timeline. Informally, a stationary time series has a well-defined mean around which it can fluctuate with constant finite variance. This assumption may be true during a short time period, while in a significantly long time period, such as one year, the load curve shows seasonal patterns and is usually not stationary. For the one year load values shown in Fig. 2.4, they are not stationary along the whole timeline. Consequently, the portrait data cleansing strategies in Section 3.3.3 may not work well.

To deal with this problem, a pre-processing method is proposed, based on two observations: (1) the load curve data exhibits periodicity in a small time scale (e.g., one month) and (2) the fundamental period of load curve data (e.g., one day) in different small time windows (e.g., January and June) is (nearly) the same. Both observations will be further validated in our late experimental evaluation.

Within a small time-scale (e.g., several days to one month), the fundamental period of the landscape data can be obtained via FFT. We first divide the whole time with the length of the fundamental period, and use the data within each time period as the basic building block. For the landscape data in the \(i\)-th period, which is denoted as \(l_i\), we define its characteristic vector \(e_i := [\theta_i, M_i]\), where \(\theta_i\) and \(M_i\) represent median and median absolute deviation (MAD) of the values in \(l_i\), respectively. Thus, similar to portrait data, the similarity of the landscape data in two different period, \(l_i\) and \(l_j\), can be defined as \(s_{ij} := 1/\|e_i - e_j\|\), or \(\infty\) if \(\|e_i - e_j\| = 0\). Here we slightly abuse the notation by using the \(e_i\) and \(s_{ij}\) to denote the characteristic vector and similarity, respectively, for both landscape data and portrait data. Their meaning, however, is easy to figure out from the context.

For the landscape data of different periods that have similar characteristic vectors, we merge them into one dataset, which is called a virtual landscape dataset (VLD). If the whole landscape data consists of \(n\) (non-overlapping) periods, the problem of constructing its VLDs can be formally defined as follow.

- **Input:** Landscape data \(\{l_1, l_2, \cdots, l_n\}\) and their characteristic vectors \(\{e_1, e_2, \cdots, e_n\}\). A given similarity threshold \(s_0\).

- **Output:** Minimum number of VLDs \(\{L_1, L_2, \cdots, L_m\}\), \(m \ll n\).

Note that the above problem is exactly the same with Problem (3.8). Thus,
Algorithm 1 can be re-used to construct VLDs. Since all data points in each of the VLDs have similar properties, they are stationary and meet the requirement for portrait data construction and cleansing. We can then further build corresponding portrait data for each VLD.

### 3.3.5 Experimental Evaluations

In this section, the real-world trace data shown in Fig. 2.4 is used to construct virtual portrait datasets. We implement the multiple strategies introduced above to detect outliers, and perform numerous experiments to evaluate the performance.

So far, there is limited literature on data cleansing applications to smart grid, and one significant contribution was in [8], in which a non-parametric regression method based on B-spline smoothing was proposed to help users identify outliers. For comparison purpose, we implement the B-spline smoothing method and compare it with our own method.

**Fundamental Period**

By applying FFT on the landscape data, we got the frequency spectrum of landscape data, and the frequency of the second peak corresponds to the fundamental frequency. Its reciprocal is the fundamental period of the landscape data. After calculation, the fundamental frequency of the landscape data in Fig. 2.4 is $1.1574 \times 10^{-5}$, which precisely results in a period of 24 hours ($86400$ seconds).

In addition, a sensitivity experiment is made with 1000 tests on the data. In each of the test, a random time period longer than 1 month but shorter than 1 year was chosen. According to the results, the mean value of the identified periods is $23.9984$ hours with variance of $1.9952 \times 10^{-4}$. Therefore, we can conclude that the accuracy of identified fundamental period is not sensitive to the time period and the starting time of the samples.

**The Optimal Threshold Value**

By applying Algorithm 1, a number of virtual (portrait/landscape) datasets can be built for a given threshold value on the similarity measure. By changing the threshold value from small to large, we can get a series number of virtual datasets. We are thus faced with the following question: what is the optimal threshold value?
In order to answer the above question, mean distance is defined to estimate the “quality” of virtual datasets (i.e., whether or not two virtual datasets are clearly separate). For \( n \) virtual datasets with corresponding characteristic vectors \( e_1, e_2, \ldots, e_n \), the mean distance is defined as:

\[
d_n := \frac{n-1}{\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} s_{ij}} / \left( \frac{n}{2} \right),
\]

(3.13)

where \( s_{ij} \) is defined by Equation (3.5). Obviously, with the same number of virtual datasets, the larger the \( d \), the clearer the separation among the virtual datasets.

By changing the threshold value on the similarity measure, we can obtain different numbers of virtual datasets. Applying the ELBOW criterion\(^2\) [65], we can get the optimal number of virtual datasets. The optimal threshold on the similarity measure is thus the one that leads to this number of virtual datasets.

**Performance Metrics**

In outlier detection, four statistical indicators are widely used: (1) true positive (\( TP \)), the number of points that are identified correctly as outliers; (2) false positive (\( FP \)), the number of points that are normal but are identified as outliers; (3) true negative (\( TN \)), the number of points that are normal and are not identified as outliers; (4) false negative (\( FN \)), the number of points that are outliers but are not identified. Using \( TP, FP, TN \) and \( FN \), we evaluate the following four broadly-used performance metrics: accuracy, precision, recall, and F-measure. Accuracy is the degree of closeness of measurement to the actual situation as a whole; precision is the percentage of correctly detected corrupted regions with regard to the total detected regions; recall is the percentage of correctly detected regions with regard to pre-labeled corrupted regions; the F-measure is a harmonic mean of precision and recall, i.e.,

\[
F\text{-measure} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.
\]

(3.14)

Furthermore, running time (R.T.) and memory usage (M.U.) of program are used to measure the time and space consumption of different methods, respectively. We implement them in R and test them with 32-bit Windows OS with 3.4GHz CPU and

\(^2\)The concept of VPD is in principle the same as clustering. The ELBOW criterion means that we should choose a number of clusters so that adding another cluster would not model the data much better.
4GB RAM.

The real-world data shown in Fig. 2.4 is used for the evaluation. Since this dataset is relatively clean, we ask three students to distort the data with “falsification”, i.e., they are asked to arbitrarily modify the load curve data within the range of \([0, \infty)\). Five percent of the samples are changed and labelled.

In our tests, the confidence coefficient is set as \(\alpha = 0.05\), which results in a confidence interval of 95%. Besides, in the IQR-based method, \(\rho\) is set as 1.5, and in the B-spline smoothing method, the degree of freedom (\(df\)) is treated as a variable and trained when smoothing the load curve.

Results from Small-Scale Data

The one-month data from 01/08/2006 to 31/08/2006 in Fig. 2.4 are firstly used for evaluations. Since these data are stationary, virtual portrait data construction strategy is applied directly. Six virtual portrait datasets are resulted based on the optimal threshold value.

In addition to the strategies introduced in Section 3.3.3, the B-spline smoothing method is also applied for each virtual portrait dataset. Performance metrics are computed with the outcomes from different methods, and the final results are summarized in Table 3.2. Furthermore, an outcome from IQR-based portrait data cleansing is shown in Fig. 3.3.

Table 3.2: Performance on small-scale data: virtual portrait data cleansing vs. B-spline smoothing

<table>
<thead>
<tr>
<th></th>
<th>Virtual Portrait Data Cleansing</th>
<th>B-spline Smoothing</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Normal Gamma IQR B-spline</td>
<td>df = 148 df = 188 df = 228 df = 258 df = 318</td>
</tr>
<tr>
<td><strong>Accuracy</strong></td>
<td>0.9878 0.9865 0.9879 0.9823</td>
<td>0.9582 0.9716 0.9715 0.9724 0.9748</td>
</tr>
<tr>
<td><strong>Precision</strong></td>
<td>0.8857 0.9375 0.9118 0.7500</td>
<td>0.8182 0.7917 0.7308 0.5405 0.4151</td>
</tr>
<tr>
<td><strong>Recall</strong></td>
<td>0.7750 0.7500 0.7750 0.6750</td>
<td>0.2250 0.4750 0.4750 0.5000 0.5500</td>
</tr>
<tr>
<td><strong>F-measure</strong></td>
<td>0.8267 0.8333 0.8378 0.7105</td>
<td>0.3529 0.5938 0.5758 0.5195 0.4731</td>
</tr>
<tr>
<td><strong>R.T. (sec)</strong></td>
<td>0.036 0.036 0.037 0.021</td>
<td>0.040 0.051 0.075 0.089 0.118</td>
</tr>
<tr>
<td><strong>M.U. (MB)</strong></td>
<td>0.067 0.067 0.067 1.427</td>
<td>2.933 3.605 4.327 4.890 5.980</td>
</tr>
</tbody>
</table>

From the above results, we can find that our virtual portrait data cleansing strategies perform much better than B-spline smoothing. For this dataset, both gamma distribution based and IQR-based detection methods perform better than the normal distribution based one. It is interesting to see that applying B-spline smoothing method to virtual portrait data does not bring clear improvement (refer to the results in 5-th column of Table 3.2). This implies that using simpler methods on portrait data
can achieve good performance already. It is unnecessary to use complex approaches such as B-spline smoothing on portrait data.

Furthermore, we can see that the virtual portrait data cleansing runs faster and uses much less memory than B-spline smoothing. In fact, most time and memory spent in our strategies are on the construction of virtual portrait datasets, and the overhead of data cleansing over portrait data is negligible. B-spline smoothing, however, spent over 99% of the running time and memory on the calculation of basis functions, which are used to fit the landscape load curve data.

**Results from Large-Scale Non-Stationary Data**

In practice, the size of load curve data is usually large and covers a time period as long as several years. Therefore, we also test the performance of our method on the one-year data shown in Fig. 2.4.

Note that the landscape data are not always stationary during the whole time window, so we pre-process the data with the method introduced in Section 3.3.4. For comparison, three solutions with 5, 7 and 10 VLDs are provided for tests and evaluations (7 VLDs are resulted from the optimal threshold value). Then for each VLD of each solution, following the same operations for small-scale dataset, we construct its virtual portrait datasets and apply portrait data cleansing strategies to identify outliers.

The results are summarized in Table 3.3, and an outcome from 7 VLDs and gamma distribution based portrait data cleansing is shown in Fig. 3.4.

From the results in Table 3.3, we can see that with non-stationary landscape data, virtual portrait data cleansing strategies are still effective and perform well. According to $F$-measure, gamma distribution based cleansing strategy does better than the other
Table 3.3: Performance on large-scale data: virtual portrait dataset (VPD) cleansing vs. B-spline smoothing

<table>
<thead>
<tr>
<th></th>
<th>5 VLDs</th>
<th>7 VLDs</th>
<th>10 VLDs</th>
<th>B-spline</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Normal</td>
<td>Gamma</td>
<td>IQR</td>
<td>Normal</td>
</tr>
<tr>
<td><strong>Accuracy</strong></td>
<td>0.9931</td>
<td>0.9950</td>
<td>0.9927</td>
<td>0.9939</td>
</tr>
<tr>
<td><strong>Precision</strong></td>
<td>0.6154</td>
<td>0.7080</td>
<td>0.7143</td>
<td>0.5820</td>
</tr>
<tr>
<td><strong>Recall</strong></td>
<td>0.5161</td>
<td>0.6452</td>
<td>0.4839</td>
<td>0.5726</td>
</tr>
<tr>
<td><strong>F-measure</strong></td>
<td>0.5614</td>
<td>0.6751</td>
<td>0.5769</td>
<td>0.5772</td>
</tr>
<tr>
<td><strong>RT (sec)</strong></td>
<td>2.74</td>
<td>2.67</td>
<td>2.31</td>
<td>3.29</td>
</tr>
<tr>
<td><strong>M.U (MB)</strong></td>
<td>0.42</td>
<td>0.42</td>
<td>0.42</td>
<td>0.42</td>
</tr>
</tbody>
</table>

Figure 3.4: Result of outlier detection from gamma distribution based virtual portrait data cleansing (7 VLDs)

two, indicating that it achieves a good balance between precision and recall and has a better overall performance; IQR-based cleansing does better at precision, indicating that this strategy performs well at exactness of outlier detection.

In contrast, outlier detection with B-spline smoothing performs poorly. With the largest degree of freedom that the computation allows\(^3\), the precision, recall and F-measure of outlier detection are all below 50%. To be worse, the overhead on running time and memory consumption is significantly higher than our method.

As shown in Fig. 3.4, our method does not identify many polluted data from time 5000 till the end. This is because those artificially polluted data are at a comparable level in value as the nearby regular load data. These “outliers” are similar to regular values and cannot be effectively detected with any method.

\(^3\)B-spline smoothing with degree of freedom larger than 2815 is beyond the capability of our desktop computers.
Discussion: Why Does B-spline Smoothing not Perform Well on Load Curve Data?

A Local View

To further investigate why B-spline smoothing does not perform well in load curve data cleansing, we first study the performance in a smaller, local scale. We analyze two situations shown in Fig. 3.5 and Fig. 3.6, where B-spline smoothing either under-fits or over-fits the load curve data.

From Fig. 3.5, we can see that the four labeled polluted data were not identified due to the under-fitted regression of B-spline. To alleviate the problem, we may increase the degree of freedom ($df$), but doing so may result in over-fitting. As shown in Fig. 3.6, in order to fit some outliers (the red dots in the figure), the fitted curve actually deviates from regular data points (the green dots in the figure), which ends up with bad performance in outlier detection. During the process from under-fitting to over-fitting, there must exist a $df$ value which results in the best performance, but finding the best $df$ value is time consuming.

The above phenomenon is caused by the inherent problem in regression method, as it treats each data point in the same way and tries to reduce the total estimation error. This may not work well because load curve data at different times follow
different statistical features. Using the *portrait* data, in contrast, we can divide data into different groups according to their attributes, and analyze each group separately. Thus “pathological” data values may infect landscape data on a large time window but has only limited impact on portrait data. This is the essential point where B-spline smoothing performs poor while virtual portrait data cleansing does better.

**A Global View**

An outcome from B-spline smoothing with $df = 100$ is shown in Fig. 3.7, in which we can have a global view of its performance.

![Figure 3.7: Results of B-spline smoothing for large-scale data ($df = 100$)](image)

From 3920h to 3975h in the load curve, the data are lost and are treated as zeros during B-spline smoothing. We can find that most missing data are not identified. This is caused by an apparent curvature trend to fit the filled data. Even if we replace the missing data with other constants, such a curvature trend is inevitable. This exposes another drawback of regression-based outlier detection methods: they cannot deal with *consecutively* polluted data. In contrast, our portrait data cleansing strategies do not have such a problem. Since all the landscape data will be separated into different portrait data, the consecutively polluted data will be evaluated and handled differently. As a result, one polluted data will not affect nearby ones.

In some special time periods such as holidays, the load values may be consecutively higher or lower in the landscape data. This scenario is similar to the above case. With regression-based outlier detection, there will be an inevitable curvature trend to fit the irregular data, while with our strategies, such data values are separated into different virtual portrait datasets and can be detected with a high possibility.
3.4 Corrupted Data Identification for Individual Residential Load

In this section, we tackle the practical problem of corrupted load curve data identification in the individual residential load. Specifically, we make the following contributions:

- First, we define a new criterion in identification of corrupted load data. The criterion is aware of domain knowledge, including the power range of appliances and the physical laws behind valid load data.

- Second, we formally formulate the Corrupted Data Identification Problem (CDIP) and establish an optimization model to solve the problem. Furthermore, we introduce a new concept, called virtual appliance, in the objective function to help record corrupted data. Our empirical study in a proof-of-concept electricity usage test environment shows that a solution to the optimization problem is capable of precisely identifying the corrupted data, even without obtaining the exact on/off states of appliances. This nice feature indicates that our method is both effective and robust.

- We develop a sequential local optimization algorithm (SLOA) to approach CDIP efficiently. SLOA focuses on solving CDIP in a smaller time window, and considers the correlation between consecutive small windows. Our SLOA method offers an efficient heuristic solution and can achieve a very high detection precision. As an extra benefit, by applying the sequential optimization algorithm, we can easily handle consecutive corrupted data.

3.4.1 CDIP Definition

In this section, we present a formal problem definition. Before that, we first describe an energy consumption model and discuss the generation rules of load data.

Energy Consumption Model

Load data is time series data that records users’ energy consumption. It is collected by (smart) meters periodically at a certain sampling frequency. Without loss of generality, we assume that the time is slotted, with each timeslot equal to the sampling
interval time. In the rest of this chapter, we thus use the terms “time”, “timeslot” and “sampling interval” interchangeably.

We denote the load data from timeslot $t = 1$ to timeslot $t = n$ in a column vector as
\[
Y \equiv [y_1, y_2, \cdots, y_n]^T,
\]  
(3.15)

where each value $y_i$ in the vector represents the aggregated energy consumption of all appliances in a property, say a house at timeslot $i$. The energy consumption at a time instant depends on the appliances’ on-off states and their individual power level.

We assume that a house includes $m$ appliances in total, and the power of the $k$-th appliance is $p_k$ (watts). At any time instant, if we record the power level of each individual appliance, we can define an $m$ dimensional column power vector to capture energy consumption of the house:
\[
P \equiv [p_1, p_2, \cdots, p_m]^T.
\]  
(3.16)

Based on the appliances power consumption pattern shown in Chapter 2.2.1, we define two $m$ dimensional column vectors, denoted as $P_l$ and $P_u$, respectively:
\[
P_l = [l_1, l_2, \cdots, l_m]^T,
\]  
(3.17)
\[
P_u = [u_1, u_2, \cdots, u_m]^T,
\]  
(3.18)

where $l_i$ and $u_i$ represent the lower and upper bounds of the power level of the $i$-th appliance, respectively. A power vector $P$ is called valid if, for each value $p_i$ in $P$, $l_i \leq p_i \leq u_i$.

At any instant, the state of an appliance may be either on or off. We use an $n \times m$ 0-1 state matrix, $S = [S_{ij}]_{n \times m}$, to record the states of the $m$ appliances from time $t = 1$ to $t = n$, where $S_{i,k} = 1$ indicates that the $k$-th appliance is on at time $i$, and 0 otherwise. In addition, we call the $i$-th row of $S$ a state vector at time $i$, denoted by:
\[
S_i \equiv [S_{i,1}, S_{i,2}, \cdots, S_{i,m}].
\]  
(3.19)

**Generation Rules of Load Data**

We have the following observations. First, a valid load data element $y_i$ (in watt-hours) should be equal to the inner product of the state vector and the power vector at $t = i$, multiplied by the sampling interval time. This is a basic physical law for
load curve data generation. Second, since the sampling interval is typically small, we assume that the probability that an appliance has more than one on-off switch events during a timeslot is negligible. In addition, the total number of on-off state switches of all appliances during a timeslot should be small. This feature is called the temporal sparsity of on-off switching events. Intuitively, this feature means that in normal operation it is unlikely that the household turns on/off many appliances in a short time. Based on the above observations, we define the generation rules of load data.

**Definition 6.** Assume that the initial state of appliances is $S_0$. We claim that each valid load data, $y_i$, must satisfy the following generation rules:

$$
\begin{align*}
S_i \cdot P_l / f &\leq y_i \leq S_i \cdot P_u / f \\
\|S_i - S_{i-1}\|_1 &\leq \delta,
\end{align*}
$$

(3.20)

where $f$ is data sampling frequency, $1 \leq i \leq n$, and $\delta$ is the upper bound on the total number of on-off state switches for $m$ appliances during a sampling interval.

Note that the energy consumption value (watt-hours) is calculated with power value (watt) multiplied by time $1/f$ (hour). To keep our discussion simple, we assume that a valid initial state vector $S_0$ is given at this moment. We will relax this assumption later and show that the impact of an inaccurate initial state vector quickly becomes negligible as long as the system runs for just a little while (Section 3.4.5).

Based on the above generation rules, corrupted data is the values that break any of the rules.

**Definition 7.** The corrupted data identification problem (CDIP) is, given load data $Y = \{y_1, y_2, \ldots, y_n\}$, power bound vectors $P_l, P_u$, and a sampling frequency $f$, find corrupted data items that violate any of the generation rules, i.e., $C \equiv \{y_i: y_i \text{ violates (Equation 3.20), for } 1 \leq i \leq n\}$.

### 3.4.2 An Important Step Towards Solving CDIP

To solve CDIP, a na"ive idea is to find all the solutions satisfying the constraints in (Equation 3.20), by brute-force, exhaustive search for all possible appliance states. This method is very time-consuming. Even for a small data set it is very costly to find the answer. Since the generation rules can be considered as constraints in
an optimization problem, we will show how the problem can be transformed to an optimization problem, which sheds light on a fast solution to an approximate problem.

**Definition 8.** *Besides the real appliances, we introduce a virtual appliance into the system. Its associated power is called virtual power, and we record the values of virtual power from time \( t = 1 \) to \( t = n \) in a virtual power vector*

\[
V \equiv [v_1, v_2, \cdots, v_n]^T,
\]  

(3.21)

*where \( v_i \in (-\infty, +\infty) \) denotes the virtual power at time \( t = i \).*

By introducing the virtual appliance, we can develop the following optimization model to solve CDIP:

\[
\begin{align*}
\min_{S_i, v_i} & \quad \|V\|_1 \\
\text{subject to} & \quad (S_i \cdot P_l + v_i) / f \leq y_i \leq (S_i \cdot P_u + v_i) / f \\
& \quad \|S_i - S_{i-1}\|_1 \leq \delta \\
& \quad S_{i,j} \in \{0, 1\} \\
& \quad 1 \leq i \leq n \\
& \quad 1 \leq j \leq m
\end{align*}
\]  

(3.22)

To understand the rationale behind the formulation of Equation (3.22), it is worthwhile to point out that \( v_i \in V \) will come into play whenever \( S_i \) cannot satisfy the generation rules, i.e., the virtual appliance is “turned on” when the load data \( y_i \) is corrupted. Thus, \( v_i \) essentially makes a record to the corrupted data. After obtaining the final solution to Equation (3.22), the \( v_i \) variables with non-zero values indicate the corrupted data, i.e.,

\[
C = \{y_i : v_i \neq 0 \text{ for } 1 \leq i \leq n\}.
\]  

(3.23)

We try to minimize \( \ell_1 \)-norm, because it is proven that for most large under-determined systems of linear equations the minimal \( \ell_1 \)-norm solution is also the sparsest solution (i.e., resulting in the minimal number of non-zero values of \( v_i \)) [66]. In addition, a larger \( v_i \) value means that a corrupted \( y_i \) is farther away from the valid range (indicated by the first generation rule in (3.20). In this sense, \( v_i \) can be also regarded as the corrupted degree of \( y_i \).
We have proved that CDIP is NP-hard by reducing the Traveling Salesperson Problem (TSP) to CDIP (refer to Appendix A for detailed proof). By investigating the special structure of the problem, however, we can develop an effective heuristic algorithm introduced in the next section.

### 3.4.3 Sequential Local Optimization

In this section, we propose a *Sequential Local Optimization Algorithm* (SLOA) and develop a quantitative strategy to estimate the minimum local window size.

**SLOA**

The temporal sparsity of corrupted load data suggests that we can perform optimization in a smaller, local time window. By considering the correlation between consecutive timeslots, we design a Sequential Local Optimization Algorithm (SLOA). Without loss of generality, we take a load data from time $t = 1$ to $t = n$ as an example to show the major steps of SLOA.

**Step 1.** Consider a small time window with size of $w$, $1 \leq w < n$, which starts from time $k$ to time $k + w - 1$. Given the state vector at time $k - 1$, i.e., $S_{k-1}$, we consider the following optimization problem:

$$\min_{S_i,v_i} \sum_{i=k}^{k+w-1} |v_i|$$

subject to

$$\frac{(S_i \cdot P_l + v_i)}{f} \leq y_i \leq \frac{(S_i \cdot P_u + v_i)}{f}$$

$$\|S_i - S_{i-1}\|_1 \leq \delta$$

$$S_{i,j} \in \{0, 1\}$$

$$k \leq i \leq k + w - 1$$

$$1 \leq j \leq m$$

(3.24)

By setting $w \ll n$, we can significantly reduce the search space. Actually, we can show that the computational complexity to solve the above problem is $O(M^w)$, where $M = \binom{m}{0} + \binom{m}{1} + \cdots + \binom{m}{\delta}$. Since $m$ is the total number of appliances and $\delta \ll m$, the problem can be solved quickly using tools such as CVX 2.0 with a Gurobi engine [67].

**Step 2.** For the $k$-th time window that starts from time $k$, we use the following strategy to handle consecutive corrupted data: if the data point at time $k$ is identified
Algorithm 2: Sequential Local Optimization Algorithm

**Input:** Load data \( \{y_1, y_2, \cdots, y_n\} \), power bounds \( P_L, P_U \), initial state \( S_0 \), sampling frequency \( f \), local time window size \( w \).

**Output:** Corrupted data set \( C \), corrupted degree \( v_i \), \( 1 \leq i \leq n \)

1: \( v_0 = 0 \)
2: \( C = \emptyset \)
3: for \( k = 1 : n \) do
4: \( \text{Solve Problem (Equation 3.24), and obtain } v_i \text{ and } S_i \text{ where } k \leq i \leq k + w - 1 \)
5: if \( v_k \neq 0 \) then
6: \( C = C \cup \{y_k\} \)
7: \( S_k = S_{k-1} \)
8: end if
9: end for
10: return \( C, \{v_1, v_2, \cdots, v_n\} \)

Estimation of Local Window Size

Clearly, one key question in SLOA is to determine a suitable size of the local window. In principle, we want the size to be as small as possible to speed up the calculation, but a size too small may result in a poor solution largely deviating from the global optimal one. For example, in the extreme case of \( w = 1 \), SLOA becomes a simple greedy search algorithm, where the final solution may not be good. On the other hand, if \( w = n \), the problem becomes the same as Equation (3.22), which is hard to solve. What is the minimum local window size that leads to a nearly global optimal solution?
Since it is hard to obtain a strict proof that the local optimal solutions together lead to the global optimal solution, we use the following heuristics to estimate the minimum local window size: within the local window, it should be possible that one state vector can be transited to any other state vectors in the vector space. In other words, within the local window, we should cover all possible state vectors in the search. This heuristics sheds light on finding the minimum local window size, as formulated below.

**Definition 9.** Consider \( m \) appliances denoted by a set \( \{ R_1, R_2, \cdots, R_m \} \), where \( R_i \equiv [l_i, u_i] \subset \mathbb{R} \) and represents the \( i \)-th appliance’s power range. An **overlap index** of the \( m \) appliances can be defined as:

\[
O \equiv \frac{\sum_{i=1}^{m} \int_{p_{\min}}^{p_{\max}} I(R_i \cap \{ x \}) dx}{\int_{p_{\min}}^{p_{\max}} I(\bigcup_{i=1}^{m} R_i) \cap \{ x \}) dx}
\]  

(3.25)

where \( p_{\max} \) and \( p_{\min} \) stand for the maximum and minimum power of all appliances, respectively, and \( I(x) \) is an indicator function

\[
I(x) = \begin{cases} 
1, & x \neq \emptyset \\
0, & x = \emptyset
\end{cases}
\]  

(3.26)

Note that the denominator \( \int_{p_{\min}}^{p_{\max}} I(\bigcup_{i=1}^{m} R_i) \cap \{ x \}) dx \) includes all valid power values, i.e., the ones that can be covered by at least one appliance’s power range. We can see that the overlap index represents the number of appliances whose power range covers a valid power value, averaged over the whole power range of all appliances. In particular, \( O = 1 \) indicates that no pair of appliances have overlapped power, and \( O = m \) means that all appliances have the same power range. Intuitively, when \( O \) is large, there is a good chance of finding multiple local optimal solutions to Equation (3.24), since there are multiple equivalent choices to turn on/off appliances in each iteration.

With the heuristics in estimating the minimum local window size, we have the following result.

**Lemma 2.** Given the overlap index \( O \) of \( m \) appliances and the upper bound \( \delta \) on the total number of on-off state switches in a timeslot, in order to get the nearly global optimal solution to Equation (3.22) via Equation (3.24), the minimum local window size \( w = \max\{ \left\lceil \frac{m}{\delta O} \right\rceil , 1 \} \).
Proof. We prove the following condition holds: within the local window, we can cover all possible state vectors in the search.

First, the value of $w$ relates to upper bound $\delta(\leq m)$ on the total number of on-off switches in one timeslot. It is obvious that from time $t = i$ to $t = i + 1$, the state vector $S_i$ can only change to another state vector $S_{i+1}$, with $\|S_{i+1} - S_i\|_1 \leq \delta$. If $\delta = m$, then within one step, a state vector is allowed to change to any other state vector. On the other hand, if $\delta = 1$, within one step, a state vector can only change one value in the vector. In other words, from one state vector, it requires at least $\left\lceil \frac{m}{\delta} \right\rceil$ timeslots to reach any other state vector in the state vector space, i.e., $w \geq \left\lfloor \frac{m}{\delta} \right\rfloor$.

Second, the overlap index $O$ can reduce the value of $w$. Based on the meaning of $O$, $m$ appliances with overlap index $O$ are equivalent to $\left\lfloor \frac{m}{O} \right\rfloor$ appliances without overlapped power. Replace $m$ with $m/O$, we can get $w \geq \left\lfloor \frac{m}{\delta \cdot O} \right\rfloor$. Considering $w \geq 1$, we conclude:

$$w \geq \max\{\left\lfloor \frac{m}{\delta \cdot O} \right\rfloor, 1\}. \quad (3.27)$$

Since we want $w$ to be as small as possible, $w = \max\{\left\lfloor \frac{m}{\delta \cdot O} \right\rfloor, 1\}$. □

Note that, although the minimum local window size obtained above is an estimation, it works effectively in our experiments with real-world data as well as with synthetic data.

Algorithm Analysis

Given $n$ load values, $m$ appliances, and the upper bound $\delta(\leq m)$ on the total number of on-off state switches in a timeslot, the computational complexity of the original problem (Equation 3.22) is $O(M^n)$, where $M = \binom{m}{0} + \binom{m}{1} + \cdots + \binom{m}{\delta}$. Using SLOA, solving the optimization problem (Equation 3.24) for $n$ times results in the time complexity of $O(n \cdot M^w)$, where $w \in Z^+$ and $w \ll n$. Please note that the appliance number $m$ is a constant value and $w$ is also a small constant.

Obviously, the larger the value of $w$, the higher the computational complexity. Fortunately, the overlap index of appliances in a house/building is usually high, as observed in our real-world testbed. This fact allows us to select a small local window size following Lemma 2. Therefore, in the application scenarios, SLOA can approach the NP-hard problem heuristically and efficiently. We will show that this algorithm indeed can provide a good solution with abundant experimental results in the following sections.
3.4.4 Experimental Evaluations

Evaluation on Real Data

Case 1: Evaluation for Laboratory Facility

We evaluate our method with real-world trace data from a real-world energy monitoring platform. We monitored the appliances’ energy consumption of a typical laboratory and a lounge room on the fifth floor of the Engineering/Computer Science Building at University of Victoria. The real-time power of laptops, desktops and some household appliances was recorded. Each appliance’s power level was measured every 10 seconds and the measurement results were transmitted with ZigBee radio to a server that stores the data. The monitored appliances and their regular power are shown in Fig. 3.8.

An appliance’s regular power is an approximate range around the rated power where this appliance works.

Figure 3.8: Energy monitoring platform and appliances’ power ranges
We collect the data over a two-month period. Fig. 3.9 demonstrates one-week and one-day load data collected by our platform. For clear illustration, we only show one-day data as an example. Note that even in a lab setting like ours, there indeed exists some apparent corrupted data, indicated by the dashed red dots in Fig. 3.9\(^5\).

\begin{figure}[!h]
\centering
\includegraphics[width=\textwidth]{figure.png}
\caption{One-week and one-day load data collected via the energy monitoring platform.}
\end{figure}

In order to introduce more corrupted data, we asked three students to distort the load data with “falsification”, i.e., they were asked to arbitrarily modify the aggregated load data within the range of \([0, \infty)\). These changed data together with the original corrupted ones were labeled and used as the ground-truth to verify the performance of our method.

Since the existing appliance-oblivious load data cleansing methods, such as B-spline smoothing, detect outliers and consider outliers as corrupted data, we use the terms “outliers” and “corrupted data” interchangeably hereafter. For outlier detection, four statistical results can be obtained: (1) true positive \((TP)\), the number of points that are identified correctly as outliers; (2) false positive \((FP)\), the number of points that are normal but are identified as outliers; (3) true negative \((TN)\), the number of points that are normal and are not identified as outliers; (4) false negative \((FN)\), the number of points that are outliers but are not identified. Using \(TP, FP, TN\) and \(FN\), we evaluate the following three broadly-used performance metrics: precision, recall, and F-measure. Precision is the ratio of the number of correctly detected

\(^5\)The corrupted data mainly comes from some incorrect power values from the laptop that occasionally reports impossible values such as hundreds of Watts.
corrupted values over the total number of detected values; recall is the ratio of the number of correctly detected values over the number of pre-labeled corrupted values; and the F-measure is a harmonic mean of precision and recall, i.e.,

\[
F\text{-measure} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.
\]  

(3.28)

For comparison, we implement and test an appliance-oblivious data cleansing method, B-spline smoothing, which is introduced in [8] to identify corrupted load data. In the B-spline smoothing method, we set the confidence coefficient \( \alpha = 0.05 \), which results in a confidence interval of 95%. We treat the degree of freedom (\( df \)) as a variable, whose value is trained when smoothing the load curve data. For our method, the overlap index is obtained as \( O \approx 2 \), and the upper bound of on-off switching events of appliances within the sampling interval is set to 2, i.e., \( \delta = 2 \). According to Equation (3.27), the local window size, i.e., the value of \( w \) in Algorithm 2, is set to 3. Since the value of local window size is an estimation, in order to obtain more comprehensive performance evaluation for our method, we also vary the local window size in a range.

Table 3.4 summarizes some of the results from the two methods. Furthermore, Fig. 3.10 and Fig. 3.11 illustrate one of the outcomes from our appliance-driven method and the B-spline smoothing method, respectively.

Table 3.4: Results of corrupted data identification on university facility data: appliance-driven approach vs. B-spline smoothing

<table>
<thead>
<tr>
<th></th>
<th>Appliance-driven approach</th>
<th>B-spline Smoothing</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w = 1 )</td>
<td>89.29% 95.83% 85.29% 84.38%</td>
<td>48.68% 50.00% 51.39% 47.44%</td>
</tr>
<tr>
<td>( w = 2 )</td>
<td>50.00% 46.00% 58.00% 54.00%</td>
<td>72.55% 74.51% 82.35% 72.55%</td>
</tr>
<tr>
<td>( w = 3 )</td>
<td>64.10% 62.16% 69.05% 65.85%</td>
<td>58.27% 59.84% 60.16% 57.36%</td>
</tr>
<tr>
<td>( w = 5 )</td>
<td></td>
<td>48.68% 50.00% 51.39% 47.44%</td>
</tr>
<tr>
<td>Precision</td>
<td></td>
<td>72.55% 74.51% 82.35% 72.55%</td>
</tr>
<tr>
<td>Recall</td>
<td></td>
<td>58.27% 59.84% 60.16% 57.36%</td>
</tr>
<tr>
<td>( F\text{-measure} )</td>
<td></td>
<td>48.68% 50.00% 51.39% 47.44%</td>
</tr>
</tbody>
</table>

From the results, we have the following interesting observations.

- Comparing to B-spline smoothing, our method performs much better in Precision, but worse in Recall. This shows that our method can identify the corrupted data more accurately, even though our output does not cover the completed set of all corrupted data. In addition, our appliance-driven method achieves a higher F-measure. F-measure reflects a balanced mean between precision and recall, indicating that our method has overall better performance.
Figure 3.10: Result of corrupted data identification on university facility data with our appliance-driven method ($w = 1, \delta = 2$); Estimated bounds denote the upper and lower power bounds based on current state vector; Corrupted degree indicates the value of the virtual appliance (Section 3.3.1.)

Figure 3.11: Result of corrupted data identification on university facility data with the B-spline smoothing method ($df = 258$)

- The performance of our method remains roughly the same when the local window size is beyond the minimum value estimated using Lemma 2. Further increase of local window size does not bring clear performance gain but takes longer running time. This suggests that our previous estimation on the minimum local window size for SLOA is appropriate.

Case 2: Evaluation for Households

In addition to the data collected from university facilities, we also evaluate our
method with real-world trace data from some typical households. Load data of four houses (each house possesses 20 ∼ 40 appliances) were collected by Belkin Inc. and made on Kaggle [68]. With the data from one of the four households, we evaluate the performance of our appliance-driven method. The experimental process is similar to that in Case 1, and the results are summarized in Table 3.5.

Table 3.5: Results of corrupted data identification on household data \((w = 1)\)

<table>
<thead>
<tr>
<th>Method</th>
<th>(\delta = 2)</th>
<th>(\delta = 3)</th>
<th>(df = 188)</th>
<th>(df = 258)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Precision</strong></td>
<td>67.86%</td>
<td>80.30%</td>
<td>36.84%</td>
<td>40.67%</td>
</tr>
<tr>
<td><strong>Recall</strong></td>
<td>73.79%</td>
<td>53.00%</td>
<td>70.00%</td>
<td>61.00%</td>
</tr>
<tr>
<td><strong>F-measure</strong></td>
<td>70.70%</td>
<td>63.86%</td>
<td>48.28%</td>
<td>48.80%</td>
</tr>
</tbody>
</table>

**Evaluation on Large-Scale Synthetic Data**

To thoroughly test our method, we evaluate its performance using large-scale synthetic data that simulates a large number of appliances and much diverse energy patterns. With different synthetic datasets, we can also test the robustness of our method.

**Load Data Generation via Monte Carlo Simulation**

There is no standard model for the load curve data of a house, since the data actually results from a complex process related to human activities. We thus use the Monte Carlo simulation to generate the load data using the following method:

- Given the lowest appliance power \((P_{min})\) and the highest appliance power \((P_{max})\), the lower bound of an appliance \((p_l)\) is a random variable uniformly distributed between \(P_{min}\) and \(P_{max}\). The upper bound of the appliance \((p_u)\) is determined by a parameter called power range ratio \((r)\) and is calculated by \(p_u = \min\{p_l + \text{random}(0, rp_l), P_{max}\}\), where \(\text{random}(0, rp_l)\) returns a random number uniformly distributed in the range \([0, rp_l]\).

- At a given sampling frequency, each appliance reports its current power value, which is a random number uniformly distributed between the appliance’s lower power bound and upper power bound. It reports 0 if its state is off.
• In a sampling interval, the number of total on-off switch events follows a Poisson distribution\(^6\) with parameter \(\lambda\).

• At the end of each sampling interval, the load data of the house is recorded as the aggregated power value of all appliances (i.e., the sum of all appliances’ load values).

To introduce some corrupted data and test the effectiveness of our method, we “corrupt” some data values by replacing them with random values uniformly distributed between [0, \(Max\)], where \(Max\) is a given large constant. The time interval of introducing corrupted data is assumed to follow an exponential distribution with the mean value of \(\mu\).

**Corrupted Data Identification on Large-Scale Appliances**

The parameters used to generate the synthetic data and the corrupted data are listed in Table 3.6.

Table 3.6: Parameter settings for load data generation and corruption

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Appliances ((m))</td>
<td>50</td>
</tr>
<tr>
<td>Sampling Frequency ((f))</td>
<td>1/6Hz</td>
</tr>
<tr>
<td>Total Time Span</td>
<td>3600s</td>
</tr>
<tr>
<td>Lowest Appliance Power ((P_{min}))</td>
<td>50w</td>
</tr>
<tr>
<td>Highest Appliance Power ((P_{max}))</td>
<td>2000w</td>
</tr>
<tr>
<td>Power Range Ratio ((r))</td>
<td>15%</td>
</tr>
<tr>
<td>Initial State ((S))</td>
<td>([0, 0, \cdots, 0]^T)</td>
</tr>
<tr>
<td>Poisson Parameter ((\lambda))</td>
<td>5</td>
</tr>
<tr>
<td>Exponential Parameter ((\mu))</td>
<td>30</td>
</tr>
<tr>
<td>Corrupted Data Range</td>
<td>([0, 50kW])</td>
</tr>
</tbody>
</table>

We treat the bound on the total number of on-off switches in a sampling interval \(\delta\) as a variable. To speed up the processing, we set the local window size to 1. The small local window size may not lead to the best performance of SLOA. However, as shown in our experimental results, even with this setting, our method already performs better than B-spline smoothing. For the B-spline smoothing method, the

---

\(^6\)Poisson distribution is a good model for situations where the total number of items is large and the probability that each individual item changes its state is small. It has been broadly adopted to simulate events related to human behavior, such as the number of telephone calls in a telephone system.
degree of freedom ($df$) is set as a variable and is trained when smoothing the synthetic data.

The performance results of our method and the B-spline smoothing method are summarized in Table 3.7. Fig. 3.12 and Fig. 3.13 illustrate one of the outcomes from our method and the B-spline smoothing method, respectively.

Table 3.7: Results of corrupted data identification on synthetic data: appliance-driven method vs. B-spline smoothing

<table>
<thead>
<tr>
<th></th>
<th>Appliance-driven Method</th>
<th>B-spline Smoothing</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\delta = 4$</td>
<td>$\delta = 5$</td>
</tr>
<tr>
<td><strong>Precision</strong></td>
<td>93.94%</td>
<td>93.94%</td>
</tr>
<tr>
<td><strong>Recall</strong></td>
<td>81.58%</td>
<td>81.58%</td>
</tr>
<tr>
<td><strong>F-measure</strong></td>
<td>87.32%</td>
<td>87.32%</td>
</tr>
</tbody>
</table>

Figure 3.12: Result of corrupted data identification on synthetic data with our appliance-driven method ($w=1, \delta = 5$)

From the results, we can see that the our method works effectively on large-scale synthetic data. In particular, the precision of our method increases with increase of $\delta$, and can even reach 100%. This result indicates that our method can provide exactly correct identification when $\delta$ is large enough. Regarding the overall performance in view of $F$-measure, our method works better with a smaller $\delta$ value and outperforms B-spline smoothing clearly.
Identification of Consecutive Corrupted Data

In practice, we often meet the situation that all data within a small time interval are corrupted or lost. Consecutive corrupted data poses a big challenge to regression-based methods, as will be illustrated in this subsection.

To introduce consecutive corrupted data, we replace the load data in a small time window as 0s, as shown in the upper part of Fig. 3.14. We then use our method and the B-spline smoothing method to test the data. Fig. 3.14 and Fig. 3.15 illustrate one outcome from our appliance-driven approach and the B-spline smoothing method, respectively.

From the results, we can see that our method does much better than B-spline smoothing for consecutive corrupted data identification. With $\delta = 5$, our method can correctly identify all the corrupted data. On the other hand, even though we regulate the parameters for B-spline smoothing, it almost failed every time to identify even half of the corrupted data.

An interesting phenomenon can be found around the consecutive corrupted data in Fig. 3.15. There is an apparent trend with B-spline smoothing to fit the corrupted data. This is mainly because the B-spline smoothing method tries to fit the curve pattern and reduce the total bias error with global optimization, indicating that it cannot deal with consecutive corrupted data well.
Figure 3.14: Identification of consecutive corrupted data with our appliance-driven method

Figure 3.15: Identification of consecutive corrupted data with B-spline smoothing method ($df = 100$)

### 3.4.5 Robustness Testing

One may question whether the performance of SLOA relies on a correct initial state vector, accurate information regarding appliances power ranges, and an accurate estimation on appliances’ on-off states. All of such information may be hard to obtain in practice. To answer this question, we test the robustness of SLOA. The synthetic dataset is created using the same parameters in Table 3.6. We first disclose the test
results and then explain the reasons.

Impact of the Initial State

For this test, we change the initial state of an appliance to a random 0-1 value, and perform multiple tests. Fig. 3.16 shows one of the outcomes. We find that, even with an incorrect initial state, our method can always recover to correct load data after a few steps. This result indicates that our SLOA method is robust against inaccurate initial power state setting.

![Figure 3.16: Fast recovery of estimated load starting from a random initial state](image)

Impact of Power Ranges

In practice, we may not precisely know the power ranges of appliances. Based on this consideration, we run extra simulations to test the robustness of our method when the power ranges of appliances is inaccurate. We carry out two kinds of tests as follow.

- Widen power range: each appliance’s power range is widened by 5% or 10%, respectively, with the center power value, i.e., \((\text{upper bound} - \text{lower bound})/2\), unchanged.

- Shift & widen power range: each appliance’s lower power bound is increased by 5% or 10%, and upper bound is increased by 5% or 10%, respectively. Note that the above operations will shift the center power value as well as widen the power range.

We do not consider the situation where appliances’ power ranges are narrowed, since intuitively a user can always widen an appliance’s power range if she/he is not
sure about the right values. The test results are summarized in Table 3.8. The results clearly indicate that, with inaccurate or even wrong power ranges of appliances, our method can still identify corrupted data with high precision.

Table 3.8: Robustness tests with incorrect power ranges of appliances

<table>
<thead>
<tr>
<th></th>
<th>Widen (5%)</th>
<th>Widen (10%)</th>
<th>Shift&amp;Widen (5%)</th>
<th>Shift&amp;Widen (10%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>δ = 3</td>
<td>δ = 4</td>
<td>δ = 3</td>
<td>δ = 4</td>
</tr>
<tr>
<td>Precision</td>
<td>93.55%</td>
<td>87.50%</td>
<td>91.30%</td>
<td>94.12%</td>
</tr>
<tr>
<td>Recall</td>
<td>76.32%</td>
<td>55.26%</td>
<td>55.26%</td>
<td>42.11%</td>
</tr>
<tr>
<td>F-measure</td>
<td>84.06%</td>
<td>67.74%</td>
<td>68.84%</td>
<td>58.18%</td>
</tr>
</tbody>
</table>

**Impact of State Vector**

We have seen that our method can give correct bounds for energy consumption most of the time. Accordingly, we might infer that the estimated states of the appliances should be the same with the real situation, or at least quite close.

In order to verify this conjecture, we calculate the difference (one-norm distance) between the estimated state $S_e$ and real state $S_r$ at each time instance. Fig. 3.17 shows the result.

![Figure 3.17: Difference between estimated state $S_e$ and real state $S_r$](image)

To our surprise, the estimated states are not close to the real states, and actually deviate remarkably from the real states. We can see that in Fig. 3.17, the mean distance between $S_e$ and $S_r$ is around 25, indicating that nearly half of the appliances are not estimated with correct states. This shows that the solution to the CDIP problem is not unique but multiple, and our method can provide right load data without need to always find the right states of appliances.
Why Is SLOA Robust?

In real life, a lot of appliances are with similar or overlapped power range. In this sense, we indeed do not need to know the exact state for similar appliances as long as we can give a good approximation for their total consumption. In addition, due to the temporal sparsity of on-off switch events in the short sampling interval and the fact that only some appliances are on at any time instant, the negative impact of inaccurate power range estimation on one appliance can be offset by the negative impact of incorrect state estimation of another appliance. The offsetting is enforced automatically by the optimization objective function that minimizes the gap between the actual load data and the estimated value.

3.4.6 Further Discussion

Our approach is based on the assumption that customers are willing to collaborate and provide their appliances’ information. This naturally causes privacy concerns. Nevertheless, this assumption is nothing special in real-world applications, since customers have to put certain trust on service providers. Commercial energy monitoring platforms, such as PlotWatt [69] and PlugWise [70], indeed require users’ appliance information.

Another concern of our solution is that users may not be able to know the power model of each appliance. Nevertheless, the information needed in our solution is simple and could be found from users’ manual, technical specification or public websites such as [38, 71]. In addition, cheap per-appliance monitoring devices [39] can be used to obtain the required information. In our above test, it can be seen that it is unnecessary to precisely know the power ranges of appliances, and our solution can tolerate up to 20% of estimation errors in appliances’ power ranges.

3.5 Conclusions

A new approach was presented to organizing and analyzing the aggregated load curve in Section 3.3. This approach was based on the inherent periodic patterns in the load curve data and re-organized the data into virtual portrait datasets that could be captured with simple models. Compared to existing regression-based analysis, portrait data-based approach significantly simplified many data analysis tasks such as outlier detection. In addition, with simple data pre-processing, our method could
effectively handle large-scale non-stationary load curve data. We tested our approach with real-world trace data, including a small-scale stationary dataset and a large-scale non-stationary dataset. The experimental results demonstrated that our approach was much more effective and efficient than existing regression-based methods over both small-scale and large-scale load curve data.

Furthermore, we developed a new approach in Section 3.4 to identify corrupted data in individual residential load. Our appliance-driven approach considers the operating ranges of appliances that are readily available from users’ manual, technical specification, or public websites. It identifies corrupted data by solving a carefully-designed optimization problem. To solve the problem efficiently, we developed a sequential local optimization algorithm (SLOA) that approaches the original NP-hard problem approximately by solving an optimization problem in polynomial time. We evaluated our method using real trace data from a university facility and four typical households, and large-scale synthetic data generated by Monte Carlo simulation. Test results indicate that our method can precisely capture corrupted data. In addition, SLOA is robust under various test scenarios, and its performance is resilient to inaccurate power range information or inaccurate power state estimation.
Chapter 4

Energy Disaggregation

4.1 Overview

Monitoring the energy consumption of major appliances in residential buildings is critical for energy management and demand response (DR) programs in smart grid. Nevertheless, prohibiting metering cost has become the main barrier for fine-grained energy monitoring. To tackle this challenge, energy disaggregation becomes a promising technique and has recently attracted heavy investment in industry. With accurate energy disaggregation, the house owner can not only learn how much energy each appliance consumes, but also take necessary actions to save energy and participate in utility demand response programs. As the smart meters have been broadly deployed all over the world, sufficiently high resolution of energy data can be collected, making it feasible to develop efficient energy disaggregation solutions.

4.2 Related Work

4.2.1 Non-Intrusive Appliance Load Monitoring

Tremendous research efforts have been devoted to solving the NIALM problem. The existing approaches can be roughly divided into two categories: signature based methods and state transition based methods.

Most approaches are based on appliances’ signatures, i.e., specific features such as the real/reactive power, current, and voltage of running appliances [18]. These methods need the support of high sampling rate and build either steady or transient signal features of appliances with labeled training datasets. The signal features are
treated as the appliances’ signatures [72, 73], based on which event detection schemes are developed to detect appliances’ on/off as well as different running states. The detected events are ascribed to certain appliances’ activities via classification [74, 37, 75]. In addition to time-domain signal features, spectral analysis has also been adopted to search for appliances’ signatures in the frequency domain [76, 77, 21]. The performance of signature based approaches depends greatly on the uniqueness of an appliance’s signature. In practice, however, the signatures of different appliances may overlap with each other, causing inaccurate event detection. Even for the same type of appliances, it may be hard to obtain the widely acceptable signature [19]. In other words, it is hard to generalize the signature learned from a particular device’s operating data. Consequently, even though a method has a good performance over a specific group of appliances, it may suffer in other datasets, caused by the over-fitting problem due to over-specific signatures. Due to these difficulties, it is not easy to use signature based methods for unambiguous appliance detection and classification.

A number of methods made use of state transition in appliances’ activities for energy disaggregation. Recently, hidden Markov model (HMM) was adopted to model the state transition patterns of appliances. The hidden states of each appliance at each time instant are predicted by inference algorithms, such as the Viterbi algorithm, with the observed emission probabilities [78, 79]. Non-negative sparse coding was proposed to solve the energy disaggregation problem in [80]. It was further discussed in [81], in which a training process was needed to obtain the basis vector related to the state transition patterns of different appliances. Although some other works, such as [40], were not to solve the energy disaggregation problem, they also utilized the appliance state transition information and their results may be helpful for energy disaggregation. The methods in this category usually need a large number of trainings, and thus are time consuming. In addition, the performance highly relies on the pattern of appliances’ activities in the training datasets, and as such the performance may vary significantly from test to test.

4.2.2 Real-Time Appliance State Monitoring

In addition to the approaches to NIALM, there are some related solutions for real-time appliance state monitoring (RTASM) from their aggregated power readings with multiple meters. In [82], the power distribution network is decomposed to multiple mono-meter trees, and the real-time states of appliances are predicted according to
their aggregated power readings by a minimum number of meters without errors. In [40], a lightweight metering and sequence decoding approach is developed to recover the real-time states of massive appliances with multiple meters, in which an HMM-based state sequence decoding model is adopted to infer the hidden states of appliances. This method is further validated in [41] with extensive experiments on simulated data and real PowerNet data. Since RTASM focuses on states prediction in real time, it only makes use of current or part of the historical aggregated data. Consequently, with limited information, the required number of meters may be too large.

Aiming at different targets, the principles in RTASM are different from those in NIALM. Timeliness is the most important requirement for RTASM methods. Therefore, only current consumption data or a small part of the historical consumption data are used for the short-time prediction. Nevertheless, accuracy instead of timeliness is the key for NIALM methods. In order to improve the accuracy, NIALM methods usually require more detailed energy consumption models and unambiguous features of appliances, and take advantage of aggregated load data during the whole time interval.

4.3 A Universal Approach to Non-Intrusive Load Monitoring

Aiming at establishing an easy-to-use, universal model for energy disaggregation, we make the following contributions in this section:

- We do not rely on appliances’ signature. Instead, we use the appliances’ rated power and power deviation, which are easy to obtain, e.g., from the user’s guide of appliances. With experimental evaluation, we show that the method is robust even if this information is not very accurate.

- Based on the simple power models and the sparsity property of appliances activities, we establish a universal Sparse Switching Event Recovering (SSER) optimization model. Unlike existing methods that minimize the aggregated residual value, our method tries to minimize the total variation of on/off switching events. The new objective function, while very effective as we will show later, has never been explored before to solve the energy disaggregation problem.
• We develop a Parallel Local Optimization Algorithm (PLOA) to solve SSER, which can significantly reduce the computational complexity of the original problem and is guaranteed to obtain the optimal solution if some weak hypotheses hold.

• We build a small-scale energy monitoring platform for a group of household appliances, and evaluate our method using the real-world trace data collected over the platform. The experimental results indicate that our approach has an overall better performance than state-of-the-art solutions, including the well-known Least Square Estimation (LSE) methods and a recently-developed machine learning method using iterative Hidden Markov Model (HMM).

4.3.1 Problem Representation and Definition

Appliance Power Model

We focus on the aggregated power readings of a number of appliances in a house, and arrange them from time $t = 1$ to $T$ as an aggregated power vector\(^1\),

$$X := [X_1, X_2, \ldots, X_T]^T.$$ (4.1)

Referring to the power consumption pattern of appliances analyzed in Chapter 2.2.1, we represent the power information of an appliance $n$ by a tuple $(I_n, P_n, \Theta_n)$, where $I_n$ is its stand-by power, $P_n$ is its rated power, and $\Theta_n$ is its power deviation.

Assume that a house is equipped with $N$ appliances. We define a stand-by power vector to represent their stand-by powers as

$$I := [I_1, I_2, \ldots, I_N]^T,$$ (4.2)

a rated power vector to represent their rated powers as

$$P := [P_1, P_2, \ldots, P_N]^T,$$ (4.3)

and a power deviation vector to represent their power deviations as

$$\Theta := [\Theta_1, \Theta_2, \ldots, \Theta_N]^T.$$ (4.4)

\(^1\)Without loss of generality, all vectors in this section are column vectors. When $T$ is used as the superscript of a vector/matrix, it means the transpose of the vector/matrix in this section.
**Definition 10.** Given a house with a certain number of appliances, we call the sum of the appliances’ stand-by power, denoted by $P_0$, as the baseline power of the house, i.e., $P_0 = \|I\|_1$.

Note that virtually all appliances’ stand-by power could be found from users’ manual, technical specification or public websites such as [38]. Theoretically, $P_0$ should be constant, which is the minimum power of the house at any time instant. In practice, however, there are small variations in $P_0$ due to inaccurate stand-by power specification, thus it is possible that the actual power could be below the baseline power.

At time instant $t$, given the state vector of all appliances $S_t$, the aggregated power reading $X_t$, $(t = 1, 2, \ldots, T)$, is bounded by:

\[
(1 - S_t)^T I + S_t^T (P - \Theta) \leq X_t, \\
(1 - S_t)^T I + S_t^T (P + \Theta) \geq X_t,
\]

where $1$ is the all-one vector. In other words, the following constraints hold:

\[
X - S^T (P + \Theta) - (I - S)^T I \leq 0, \\
S^T (P - \Theta) + (I - S)^T I - X \leq 0,
\]

where $I$ is the $N$-by-$T$ all-one matrix.

**Appliances Switching Events Sparsity**

According to the analysis in Chapter 2.2.2, we denote the on/off states of $N$ appliances from time $t = 1$ to $T$ with a state matrix, $S$, defined as

\[
S := \begin{bmatrix}
S_1^{(1)} & S_2^{(1)} & \cdots & S_T^{(1)} \\
S_1^{(2)} & S_2^{(2)} & \cdots & S_T^{(2)} \\
\vdots & \vdots & \ddots & \vdots \\
S_1^{(N)} & S_2^{(N)} & \cdots & S_T^{(N)}
\end{bmatrix},
\]

in which $S_t^{(n)}$ represents the on/off state of the $n$-th appliance at time $t$, and $S_t^{(n)} \in \{0, 1\}$ with $S_t^{(n)} = 1$ indicates the $n$-th appliance is on and 0 otherwise.

Then, the on/off switching events of the $N$ appliances from $t = 2$ to $T$ can be
indicated by an event matrix, $\Delta S$, calculated as

$$\Delta S = SD,$$  \hspace{1cm} (4.8)

where $D$ is a differential matrix with size of $N$-by-$(N - 1)$:

$$D := \begin{bmatrix}
-1 & 1 & & \\
1 & -1 & & \\
& 1 & \ddots & \\
& & \ddots & -1 \\
& & & 1 & -1 \\
& & & & 1
\end{bmatrix}$$  \hspace{1cm} (4.9)

The element of event matrix $\Delta S_i^{(n)} \in \{-1, 0, 1\}$, with $\Delta S_i^{(n)} = 1$ or $-1$ indicating a switching on or off event of the $n$-th appliance at time $t$, respectively, and 0 no switching event. Since the sampling rate of current smart meters nowadays is relatively high, we neglect the situation where an appliance has a series of switching events within a sampling interval, i.e., $|\Delta S_i^{(n)}| < 2$.

**Assertion 1.** According to our real-world observations, $\Delta S$ is a sparse matrix.

### 4.3.2 Sparse Switching Events Recovering

To compute the energy consumption of an individual appliance during a time period, we also need to know its activities (states changing) along the timeline. Based on the power model of appliances and sparsity feature of switching events, we formulate the following problem to recover the on/off states of individual appliance at each time instant.

- **Input:** Aggregated power vector $X$, power pattern $(I, P, \Theta)$.
- **Output:** State matrix $S$, i.e., the on/off states of all appliances along the timeline.
A **Sparse Switching Event Recovering (SSER)** model is established to recover the states of \( N \) appliances from time \( t = 1 \) to \( T \).

\[
\begin{align*}
\min_{\Delta S} & \quad \text{TV}(\Delta S) \\
\text{s.t.} \quad & \quad X - S^T(P + \Theta) - (I - S)^T I \leq 0, \\
& \quad S^T(P - \Theta) + (I - S)^T I - X \leq 0,
\end{align*}
\]

where \( \Delta S \) is defined by (4.8) and \( \text{TV}(\cdot) \) denotes the total variation of the event matrix calculated by

\[
\text{TV}(\Delta S) := \sum_{n=1}^{N} \sum_{t=1}^{T} |\Delta S_{t}^{(n)}|.
\]

After obtaining the on/off states of each appliance along the timeline, we can estimate its power readings with its rated power at each time instant. Therefore, we can get an approximate estimation of the power consumption of each appliance. This is equivalent to solving the original energy disaggregation problem. Note that all appliances contributing to the aggregated power vector need to take into consideration in SSER model. Otherwise, the accuracy of recovered states will decrease. This may be a limitation when applying our approach, as some appliances may be unknown or forgotten.

The total variation (TV) minimization is a classical approach to recovering a sparse matrix. It has been widely applied in signal restoration, image denoising, and compressive sensing [83]. To the best of our knowledge, however, it has not been explored in the context of energy disaggregation. Unlike other optimization methods, such as least square fitting [37], total variation minimization is a type of least absolute deviations fitting, which has been proved to be more robust in various applications.

### 4.3.3 Parallel Local Optimization

There were significant research efforts to solve the total variation minimization problem. Nevertheless, the form of total variation in our case is a discrete version and involves integer variables, which is much harder. We prove that solving SSER is NP-hard (refer to Appendix A for detailed proof), so it is hard to find the optimal solution, especially when the time interval is large. Nevertheless, the active epochs of on/off events suggest that we can perform optimization in a smaller, local time window.
Detection of Active Epochs

**Definition 11.** An active epoch of a house is defined as a time interval from the time when the aggregated power of the house jumps above the baseline power until the time when the aggregated power drops below the baseline power.

![Figure 4.1: A sketch map to illustrate the concepts of active epoch and baseline power using three appliances](image)

Fig. 4.1 is a sketch map of switching activities and power readings of three appliances with constant power, in which the concepts of baseline power and active epoch are illustrated. Algorithm 3 shows the pseudo code of detecting active epochs.

**Algorithm 3** Active Epoch Detection

**Input:** Aggregated power vector $X$, baseline power $P_0$.

**Output:** Set of Active epochs, $W$.

1: $t = 1, k = 0$
2: while $t \leq T$ do
3:     $start = end = t$
4:     while $X_{end} > P_0$ and $end < T$ do
5:         $end = end + 1$
6:     end while
7:     if $end > start$ then
8:         $k = k + 1$
9:         $W_k = [start, end]$
10:    end if
11:    $t = end + 1$
12: end while
13: return $W = \{W_1, W_2, \cdots, W_k\}$

**Parallel Local Optimization Algorithm (PLOA)**

Without loss of generality, we take aggregated load data of $N$ appliances from time $t = 1$ to $T$ as an example to show the major steps of PLOA.
Step 1: Detect all active epochs along the timeline with Algorithm 3. Denote the set of active epochs as $W = \{ W_1, W_2, \cdots, W_k \}$.

Step 2: In the active epoch starting at $t$ with the length of $\ell$, solve the following optimization problem to obtain $S_{t:t+\ell}$.

$$\min_{S_{t:t+\ell}} \text{TV}(S_{t:t+\ell}D_{t:t+\ell})$$

s.t.

$$X_{t:t+\ell} - (S_{t:t+\ell})^T(P + \Theta) - (I_{t:t+\ell} - S_{t:t+\ell})^T I \leq 0,$$

$$(S_{t:t+\ell})^T(P - \Theta) + (I_{t:t+\ell} - S_{t:t+\ell})^T I - X_{t:t+\ell} \leq 0,$$

where $S_{t:t+\ell}$ is a $N$-by-$\ell$ submatrix of $S$, $D_{t:t+\ell}$ is a $\ell$-by-$(N - 1)$ submatrix of $D$, $I_{t:t+\ell}$ is a $N$-by-$\ell$ submatrix of $I$, and $X_{t:t+\ell}$ is a vector containing the aggregated power readings of all appliances from time $t$ to time $t + \ell$.

Step 3: Perform Step 2 on the $k$ active epochs to obtain a group of $k$ solutions in parallel. Since outside of active epochs, appliances are considered as stand-by, a complete state matrix $S_{1:T}$ can thus be built.

We can show that the computational complexity to solve (4.12) is $O(2^N \cdot \ell)$. Since $\ell \ll T$ as shown in Fig 2.2, the (mixed-integer linear program) problem can be solved efficiently, using tools such as CVX 2.0 with a Gurobi engine [67].

Theorem 1. Assume that the global optimal solution to SSER in (4.10) is $S^*$, and the solution obtained from POLA is $\hat{S}$, if both solutions are unique, then $\hat{S} = S^*$.

Proof. For an arbitrary active epoch starting at $t$ with the length of $\ell$, assume that $\hat{S}_{t:t+\ell}$ is the unique local optimal solution obtained via (4.12). Assume that the global optimal solution to SSER in (4.10) is $S^*$, and the sub-matrix constructed by the $t$-th to $(t + \ell)$-th columns of $S^*$ is $S^*_{t:t+\ell}$. We prove the theorem by contradiction.

Assume that

$$\hat{S}_{t:t+\ell} \neq S^*_{t:t+\ell}. \quad (4.13)$$

Then, the following inequality must hold

$$S^*_{t:t+\ell}D_{t:t+\ell} \geq \hat{S}_{t:t+\ell}D_{t:t+\ell}. \quad (4.14)$$

Therefore, there must exist another global solution $S^{**}$, in which the $j$-th column
is
\[ S_j^{**} = \left\{ \begin{array}{ll}
\hat{S}_j, & j \in [t, t + \ell], \\
S_j^*, & j \notin [t, t + \ell],
\end{array} \right. \]  \hspace{1cm} (4.15)
such that
\[ S^{**}D \leq S^*D. \]  \hspace{1cm} (4.16)

Obviously, (4.16) is contradictory to the assumption that \( S^* \) is the uniquely global optimal solution to SSER. Therefore, the assumption (4.13) is not true. As a result, we have
\[ \hat{S}_{t:t+\ell} = S^*_{t:t+\ell}. \]  \hspace{1cm} (4.17)

Outside the active epochs, PLOA treats all appliances as stand-by, the TV value is 0 in \( \hat{S} \). Since the TV value cannot be negative, the TV value obtained with PLOA is the minimum and must be the same as that obtained with the global optional solution.

Overall, if the global optimal solution is unique, for any time instant \( t \), no matter whether \( t \) is in an active epoch or outside active epochs, the state vector \( \hat{S}_t \in \hat{S} \) must be equal to the state vector \( S_t^* \in S^* \), which means \( \hat{S} = S^* \).

Given \( T \) aggregated power readings generated by \( N \) appliances that can be broken into \( k \) active epochs with maximum size \( w \), the computational complexity of the original SSER problem (4.10) is \( O(2^N T) \). With PLOA, solving the local optimization problem in (4.12) \( k \) times results in the time complexity upper bounded by \( O(k \cdot 2^{N \cdot w}) \). Considering that the number of appliances \( N \) is constant and \( w \ll T \), PLOA significantly cuts down the computational complexity.

### 4.3.4 Experimental Evaluations

We use the real-world trace data collected from our energy monitoring platform to evaluate our method, and compare it with 1) a signature based approach, the Least Square Estimation based integer programming method [37] and 2) a state transition based approach, the iterative Hidden Markov Model [79].

#### Data Collection

Currently, there are some research based datasets available for energy disaggregation. Most of them, however, 1) are circuit-oriented rather than appliance-oriented, such as the REDD dataset [2], or 2) lack power information of appliances. To avoid these
problems, we setup an energy monitoring platform where we can gather information according to our demand.

We monitored the appliances’ energy consumption in a typical laboratory and a lounge room in the fifth floor of Engineering/Computer Science building at the University of Victoria (UVic). Using an off-the-shelf solution developed by Current Cost (http://www.currentcost.com), we recorded the real-time power of laptops, desktops and some household appliances. Each appliance’s real power was measured every 6 seconds by the device called Individual Appliance Monitor (IAM), and the measurement results were transmitted via wireless to a display server (EnviR), which can display and temporarily store the collected data. Then, the data in EnviR were sent to our data server. The platform, monitored appliances, and measuring devices are shown in Fig. 4.2.

The power consumption information of appliances are summarized in Table. 4.1\(^2\), where the rated and stand-by powers are learned from the users’ manual or according to [38], and the power deviations are estimated from the collected power data of each appliance. One may be concerned that the estimation of power deviation in practice is inaccurate. With experimental study, however, we will show that our method is resilient to inaccurate power deviation estimations in Section 4.3.4.

**Performance Evaluation**

We collected data for three months from the energy monitoring platform, and one-week data were used for performance evaluation. The evaluation metrics are defined as:

\(^2\)Considering some appliances may have multiple operating modes (rated powers), we regard each as a *virtual appliance*, such that an individual appliance with multiple modes was split into multiple virtual ones in the SSER model.
Table 4.1: Powers information of appliances

<table>
<thead>
<tr>
<th>ID</th>
<th>Appliance</th>
<th>Mode</th>
<th>Rated Power (Watts)</th>
<th>Power Deviation (Watts)</th>
<th>Stand-by Power (Watts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LCD-Dell</td>
<td>1</td>
<td>25</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>LCD-LG</td>
<td>1</td>
<td>22</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>Desktop</td>
<td>1</td>
<td>40</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>50</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Server</td>
<td>1</td>
<td>130</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>iMac</td>
<td>1</td>
<td>35</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>50</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Laptop</td>
<td>1</td>
<td>15</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>30</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>70</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Printer</td>
<td>1</td>
<td>400</td>
<td>50</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>700</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>900</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Microwave</td>
<td>1</td>
<td>1000</td>
<td>100</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1200</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>1700</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Coffee Maker</td>
<td>1</td>
<td>700</td>
<td>100</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>900</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>1100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Refrigerator</td>
<td>1</td>
<td>115</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>350</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>450</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Water Cooler</td>
<td>1</td>
<td>65</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>380</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>450</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

- **Energy Disaggregation Accuracy (EDA):** It indicates the accuracy of assigning correct power values to corresponding appliances and was also used in [2].

\[
EDA := 1 - \frac{\sum_{n=1}^{N} \| X^{(n)} - \hat{S}^{(n)} P_n \|_1}{\| X \|_1}, \quad (4.18)
\]

where \(X^{(n)}\), \(\hat{S}^{(n)}\) and \(P_n\) represent the true energy consumption vector, the estimated state vector, and the rated power of the \(n\)-th appliance, respectively, and \(X\) is the aggregated power vector.

- **State Prediction Accuracy (SPA):** It indicates the accuracy of estimating the
states of appliances.

\[ SPA := 1 - \frac{\sum_{n=1}^{N} \| S^{(n)} - \hat{S}^{(n)} \|_1}{N \cdot T} \]  

(4.19)

where \( S^{(n)} \), \( \hat{S}^{(n)} \) represent the true state vector and the estimated state vector of the \( n \)-th appliance, respectively, and \( N, T \) represent the number of appliances and the number of samples, respectively.

- Running time (\( R.T. \)) and memory usage (\( RAM \))\(^3\): They indicate the overhead on running time and memory space, respectively.

Since the performance of the iterative HMM method depends on model training, we run this method multiple times over different sizes (w.r.t. number of samples) of training datasets (denoted as \( \text{training size} \)). The average performance is calculated over all the runs, and the best and the worst performance are the best and the worst outcomes among all the runs, respectively.

Table 4.2: Accuracy and overhead of energy disaggregation, using Sparse Switching Event Recovering (SSER), Least Square Estimation (LSE) based integer programming and iterative Hidden Markov Model (HMM)

<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EDA</td>
<td>SPA</td>
</tr>
<tr>
<td>SSER</td>
<td>61.12%</td>
<td>69.62%</td>
</tr>
<tr>
<td>LSE</td>
<td>33.40%</td>
<td>45.67%</td>
</tr>
<tr>
<td>HMM (average)</td>
<td>55.27%</td>
<td>67.47%</td>
</tr>
<tr>
<td>HMM (best)</td>
<td>67.26%</td>
<td>71.29%</td>
</tr>
<tr>
<td>HMM (worst)</td>
<td>41.09%</td>
<td>61.27%</td>
</tr>
</tbody>
</table>

With the same prior knowledge in Table 4.1, the performance of the three methods are summarized in Table 4.2. In addition, as illustrated in Fig. 4.3, we also look into the overall energy disaggregation accuracy of the three methods, which indicates the energy contribution of each appliance to the total energy consumption in the whole time period. From the results, we can draw the following conclusions:

- In term of accuracy, our SSER method performs much better than the LSE based method and slightly better than the iterative HMM method in average.

\(^3\)We implemented the three methods with Matlab 8.0 and run them with 32-bit Windows OS with 3.4GHz CPU and 4GB RAM.
• In term of overhead, our SSER method and the LSE method are at a comparative level for running time and system memory usage. While the memory usage of the iterative HMM method is similar to that of the other two method, its running time is much higher.

• The performance of the iterative HMM method is subject to the training process and may have a large variation in accuracy and in running time.

Robustness Test

In practice, the rated power of an appliance can be easily learned. However, we may not precisely estimate the power deviation of an appliance working under a certain mode. As such, we test the performance of our method, assuming that the power deviations of appliances are inaccurate. We replace $\Theta$ with $\rho \cdot \Theta$ in the SSER model, so that the estimated power deviations can be changed by regulating $\rho$.

Table 4.3: Accuracy of Energy Disaggregation using SSER with inaccurate estimation on power deviation

<table>
<thead>
<tr>
<th>$\rho$</th>
<th>EDA</th>
<th>SPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.8</td>
<td>55.28%</td>
<td>70.37%</td>
</tr>
<tr>
<td>0.9</td>
<td>60.33%</td>
<td>70.27%</td>
</tr>
<tr>
<td>1.0</td>
<td>61.12%</td>
<td>69.62%</td>
</tr>
<tr>
<td>1.1</td>
<td>56.94%</td>
<td>71.15%</td>
</tr>
<tr>
<td>1.2</td>
<td>59.59%</td>
<td>72.24%</td>
</tr>
</tbody>
</table>

We changed the value of $\rho$ from 0.8 to 1.2, causing a parametric error of power
deviation up to 20%. Part of the outcomes are shown in Table 4.3. We can see that the accuracy does not change too much when the parameter error varies, indicating that our method is robust to parameter estimation.

4.4 A Scalable Approach to Semi-Intrusive Load Monitoring

Targeting at developing a solution to energy disaggregation for large-scale appliance groups, we make the following contributions in this section.

- For a large-scale appliance group, instead of using only one meter, we deploy multiple meters distributed over the power network, each measuring a subgroup of appliances. We call such an approach semi-intrusive appliance load monitoring (SIALM).

- To infer the states of different appliances, instead of relying on sophisticated appliance signatures from massive load data, we make use of the appliances’ rated power.

- For a group of appliances, we give the sufficient conditions for unambiguous state recovery. The conditions provide theoretical evidence for accurate energy disaggregation results. Accordingly, a parallel optimization algorithm is developed for accurate appliance state recovery. In addition, we quantitatively investigate the impact on energy disaggregation results when the sufficient conditions are violated.

- We perform comprehensive performance test and robustness test with real-world data and synthetic data. The experimental results show that with the help of a small number of extra meters, our SIALM approach can significantly improve the accuracy of energy disaggregation for large-scale appliances. Such improvement is resilient to inaccurate power estimation and network topology changes.

4.4.1 Problem Representation and Definition

Appliance Power Model

Without loss of generality, we consider a group of $N$ appliances in a household. Referring to the power consumption pattern of appliances analyzed in Chapter 2.2.1,
for the \( n \)-th appliance with \( m \) operating modes, we denote the corresponding rated powers with a \textit{power vector} as:

\[
p^n := [p^n_1, p^n_2, \cdots, p^n_m].
\] (4.20)

We use a \textit{power deviation vector} to represent the power deviations at corresponding operating modes as:

\[
\theta^n := [\theta^n_1, \theta^n_2, \cdots, \theta^n_m],
\] (4.21)

To denote the rated power of all the \( N \) appliances with multiple operating modes, we use the following power vector:

\[
p := [p^1, p^2, \cdots, p^N]^T.
\] (4.22)

and power deviation vector:

\[
\theta := [\theta^1, \theta^2, \cdots, \theta^N]^T.
\] (4.23)

\textbf{Definition 12.} \textit{The power model} of the \( n \)-th appliance can be defined as:

\[
\lambda^n := \{p^n, \theta^n\},
\] (4.24)

and a group of \( N \) appliances can be denoted as:

\[
\Lambda := \{\lambda^1, \lambda^2, \cdots, \lambda^N\}.
\] (4.25)

Furthermore, for the \( n \)-th appliance, we use a \textit{state vector} to denote the states of its corresponding operating modes at an arbitrary time instant, \( t \), as:

\[
s^n(t) := [s^n_1(t), s^n_2(t), \cdots, s^n_m(t)],
\] (4.26)

where \( s^n_m(t) \) represents the on/off state of the \( m \)-th operating mode of the \( n \)-th appliance at time instant \( t \). Thus, \( s^n_m(t) \in \{0, 1\} \) and \( \|s^n(t)\|_1 \leq 1 \), where \( \|\cdot\|_1 \) means \( l_1 \) norm (i.e., \( \sum_{j=1}^m |s^n_j(t)| \)) and \( s^n_m(t) = 1 \) indicating that the \( n \)-th appliance is on the \( m \)-th operating mode at time \( t \) and 0 otherwise.
4.4.2 Semi-Intrusive Appliance Load Monitoring

Definition 13. **Unambiguous State Recovery:** Given the aggregated power readings of multiple appliances in a time interval, provide the exact state of each appliance at each time instant.

Definition 14. **Predictable Energy Disaggregation:** Given the aggregated power readings of multiple appliances in a time interval, estimate the energy consumption of each appliance during the time interval with a bounded error.

With the above definitions, the semi-intrusive appliance load monitoring problem can be defined as follow.

Definition 15. **Semi-Intrusive Appliance Load Monitoring (SIALM):** Given a group of appliances and their power models, provide the minimum number of meters to ensure both unambiguous state recovery and predictable energy disaggregation.

Parallel Sparse Switching Event Recovering

In this section, we first design a sparse switching event recovering (SSER) model based on total variation minimization. Then, a parallel local optimization algorithm is proposed to solve the (NP-hard) SSER problem.

**SSER Model**

Considering a group of \( N \) appliances with states in (4.26), at an arbitrary time instant, \( t \), the **state vector** of all appliances can be denoted as:

\[
s(t) := [s^1(t), s^2(t), \cdots, s^N(t)]^T ,
\]  

(4.27)

which is an \( M \)-dimension column vector, where \( M = \sum_{n=1}^{N} \| p^n \|_0 \) and \( \| p^n \|_0 \) denoting the number of operating modes of the \( n \)-th appliance since \( \| \cdot \|_0 \) counts the nonzero elements in a vector and all values in the vector \( p^n \) are larger than 0.

**Lemma 3.** Given the power model of \( N \) appliances in (4.25), the aggregated power reading of the \( N \) appliances at an arbitrary time instant \( t \), denoted as \( x(t) \), should be bounded by:

\[
s^T(t) (p - \theta) \leq x(t) \leq s^T(t) (p + \theta) .
\]  

(4.28)
With (4.27), we can construct a state matrix of all appliances in a time interval, without loss of generality, from time \( t = 1 \) to \( t = K \), as:

\[
S := [s(1), s(2), \cdots, s(K)].
\]  

Since \( s(t) \) is an \( M \)-dimension column vector, \( S \) is an \( M \times K \) matrix.

Given the power model of \( N \) appliances in (4.25), based on Lemma 3, their aggregated power readings from time \( t = 1 \) to \( t = K \), denoted as:

\[
x := [x(1), x(2), \cdots, x(K)]^T,
\]  

should be bounded by:

\[
S^T (p - \theta) \leq x \leq S^T (p + \theta).
\]  

A general feature of most appliances’ state (operating mode) switching is sparsity. Based on the sparsity feature of appliances’ state switching, we establish an optimization model of sparse switching event recovering (SSER):

\[
\begin{align*}
\min_S & \quad \text{TV}(S) \\
\text{s.t.} & \quad x - S^T(p + \theta) \leq 0, \\
& \quad S^T(p - \theta) - x \leq 0, \\
& \quad HS \leq 1.
\end{align*}
\]  

In the above model, i) \( \text{TV}(S) \) denotes the total variation of the state matrix \( S \), calculated by

\[
\text{TV}(S) := \|SD\|_F = \sum_i \sum_j |(SD)_{i,j}|,
\]  

where \( D \) is a \( K \)-by-\((K - 1)\) difference matrix defined by:

\[
D := \begin{bmatrix}
-1 \\
1 & -1 \\
& 1 & \ddots
& \ddots \\
& & \ddots & -1 \\
& & & 1 & -1 \\
& & & & 1
\end{bmatrix}_{K-1}
\]  

(4.34)
ii) $H$ is an $N$-by-$M$ permutation matrix defined by:

$$H := \begin{bmatrix}
1 & \cdots & 1 \\
\|p^1\|_0 & 1 & \cdots & 1 \\
\|p^2\|_0 & \ddots & \ddots & \ddots \\
\|p^N\|_0 & & & 1 & \cdots & 1
\end{bmatrix},$$

(4.35)

and iii) $\mathbf{0}$ is a $K$ dimensional all 0 vector, and $\mathbf{1}$ is an $N$-by-$K$ all 1 matrix.

Remark 1. In the above SSER model, the objective function of total variation (TV) minimizing is applied as a classical approach to recovering a sparse matrix. It has been widely used in signal restoration, image denoising, and compressive sensing [83]. In our case, by using TV minimization, we actually minimize the total number of state changes which is regarded sparse along the timeline. For the constraints, the first two are the power reading constraints derived in equation (4.31); the third one represents the fact that an appliance can be only on one operating mode at any instant of time.

Parallel Optimization for SSER Model

To solve the SSER model, however, is proved to be NP-hard (refer to the proof in Appendix A). Thus, an approximate approach has to be proposed to solve the problem, especially when the dataset size is large.

We have observed that the aggregated power of a house increases and varies when the occupants are at home and use appliances, while it decreases to the baseline when the house is unoccupied or no appliances are used. With this observation, we can split the whole timeline into smaller time windows based on whether the house is occupied or the power consumption is close to the baseline.

As a real case shown in Fig. 4.4, the one-day aggregated power curve can be split into four kinds of smaller windows: i) periodical windows along the timeline for the activity of always-on appliances (e.g., the fridge in this example), ii) morning activities window, iii) noon/afternoon activities window, and iv) evening activities window. Note that the occupants are not always at home and even when they are at home, the appliance usage is always in burst. Thus, the length of split time windows can be quite short (at worst no more than one day).
Figure 4.4: A real example of one-day power consumption from a residential house and four types of split time windows.

For a certain time window starting from $t = k$ and with length $\ell$, we can solve a sub-problem of SSER model with the following form:

\[
\begin{align*}
\min_{S_{k:k+\ell}} \quad & \text{TV}(S_{k:k+\ell}) \\
\text{s.t.} \quad & x_{k:k+\ell} - S_{k:k+\ell}^T(p + \theta) \leq 0, \\
& S_{k:k+\ell}^T(p - \theta) - x_{k:k+\ell} \leq 0, \\
& H_{k:k+\ell} S_{k:k+\ell} \leq 1,
\end{align*}
\]  

(4.36)

in which the size of original matrices (i.e., $S$ and $H$) and vector (i.e., $x$) appearing in (4.32) is cut down to $\ell$. As $\ell$ is expected to be way shorter than the whole time length, the sub-problem can be solved efficiently using tools such as CVX with a Gurobi engine [67].

Assuming that we have split the whole time period into $Q$ windows, for the $q$-th $(1 \leq q \leq Q)$ window with length $w_q$. Thus, for each split time window we can perform a local optimization by solving a sub-problem like (4.36). Algorithm 4 shows the pseudo code of the parallel optimization procedure, in which \textit{parfor} \footnote{With support of Parallel Computing Toolbox, \textit{parfor} loop can be applied in Matlab and bring in multiple workers to execute codes in parallel.} is a notation of parallel computing instead of \textit{for}.

By applying the parallel optimization for SSER model, we can recover the state matrix $S$, i.e., the on/off states of different operating modes of each appliance along the timeline. Then, the energy consumption of individual appliances can be estimated with the rated power information. Meanwhile, based on the power deviations of different operating modes, the upper and lower bounds of the energy consumption of...
Algorithm 4 Parallel Optimization to Solve SSER Model

\textbf{Input:} Aggregated power vector $x_{1:K}$, appliances power vector $p_{{1:M}}$ and deviation vector $\theta_{1:M}$, split time windows $w_{{1:Q}}$.

\textbf{Output:} State matrix $S_{1:K}$.

1: \texttt{parfor} $q \leftarrow 1 : Q$ do
2: \hspace{1em} $t \leftarrow (w_q).start$
3: \hspace{1em} $\ell \leftarrow (w_q).end - (w_q).start$
4: \hspace{1em} Get $S_{{t:t+\ell}}$ by solving (4.36).
5: \texttt{end parfor}
6: \texttt{return} $S_{1:K}$.

individual appliances can be provided. Eventually, we achieve the initial objective of energy disaggregation.

4.4.3 Unambiguous State Recovery

With the SSER optimization model, we first provide the sufficient conditions to achieve unambiguous appliance state recovery. When the sufficient conditions are violated, we define the ambiguous degree to quantify the violations and investigate the impact on the accuracy of energy disaggregation.

\textbf{Sufficient Conditions of Unambiguous State Recovery}

\textbf{Theorem 2. Given} a group of appliances and their power models, the sufficient conditions for unambiguous state recovery with the SSER model are:

\[ \begin{align*}
\text{C-0:} & \quad \|s(t + 1) - s(t)\|_1 \leq 1, \forall t \\
\text{C-1:} & \quad [(p - \theta)_i, (p + \theta)_i] \not\subseteq [(p - \theta)_j, (p + \theta)_j], \forall i \neq j \\
\text{C-2:} & \quad 2 \cdot \|\theta\|_1 < (p - \theta)_i, \forall i
\end{align*} \]

where $(p \pm \theta)_i$ denotes the $i$-th element of vector $p \pm \theta$.

\textbf{Proof.} Assume that $\mathcal{F}$ is the set of feasible solutions satisfying the constraints in SSER model, the optimal solution from SSER model is $S_{\text{opt}} \in \mathcal{F}$, which makes $SD$ sparsest, and the ground-truth state matrix is $S_{\text{true}}$ and $S_{\text{true}} \in \mathcal{F}$. Then, $S_{\text{opt}} \neq S_{\text{true}}$, if and only if: (A) $S_{\text{opt}}$ is not a unique solution of (4.32), or (B) $S_{\text{opt}}$ is sparser than $S_{\text{true}}$, i.e.,

\[ S_{\text{opt}} \neq S_{\text{true}} \iff A \lor B. \]  

(4.37)
With constraint of C-0, no more than one appliance could change its states during a sampling interval. Thus, conditions A and B can only be caused by two kinds of power range settings, respectively:

- A is caused by the ambiguity of power range settings illustrated in Fig. 4.5, i.e., the power range of one device is covered by that of another device.

- B is caused by an over-sparse optimal solution, i.e., the switch events of a device cannot be identified since they are submerged as part of power deviation of another device, as illustrated in Fig. 4.6.

The constraint C-1 is to eliminate condition A and the constraint C-2 is to avoid condition B.

Therefore, we have

\[ C-0 \land C-1 \land C-2 \implies \neg A \land \neg B \iff S_{\text{opt}} = S_{\text{true}}, \tag{4.38} \]

which means SSER achieves unambiguous state recovery.

\[ \square \]

Figure 4.5: The power range of A_2 is covered by that of A_1. Therefore, when A_2 is actually on, there will be two optimal solutions, as the switching events of A_1 and the switching events of A_2 are indistinguishable.

The physical meanings of the three conditions in Theorem 2 can be interpreted as follow:

- **C-0**: no more than one appliance could change its states during the same sampling interval, which is also known as “one-at-a-time” constraint and has been used before [84].
Figure 4.6: The power range of $A_3$ is submerged by the power deviation of $A_1$. Therefore, when $A_1$ is on, the state switching events of $A_3$ only result in small power changes, and due to the objective function of (4.32), such small changes will be considered as part of power deviation of $A_1$. In other words, the switching events of $A_3$ will not be identified, resulting in an optimal solution sparser than the ground-truth.

- **C-1**: no power range of any appliance is allowed to be totally submerged by others (to eliminate the situation appearing in Fig. 4.5).

- **C-2**: the power deviation of any appliance should be smaller than the power range of any other appliance (to eliminate the situation appearing in Fig. 4.6).

**Remark 2.** The constraints stated in the theorem are sufficient conditions for achieving unambiguous state recovery. The theorem does not necessarily mean in practice all the conditions will be met, e.g., the estimated power deviation may not be accurate. The violation of these conditions is the main reason why we cannot achieve 100% accuracy in practice. Nevertheless, the theorem provides us with good heuristics to search for high-accuracy solutions.

After recovering the state of appliances at each time instant, we estimate the real power of each appliance at the time instant as the rated power at the recovered state. With unambiguous state recovery under the SSER model, we can easily provide the error bounds for energy disaggregation as follows.

**Corollary 1.** Energy Disaggregation Error Bounds for Individual Appliance: with unambiguous appliance state recovery under the SSER model, for the $n$-th appliance with power model $\lambda^n = \{p^n, \theta^n\}$, the energy disaggregation error $\epsilon$ satisfies 1) at each time instant: $\epsilon \leq \theta^n$, and 2) from time $t = 1$ to $t = K$: $\epsilon \leq K \cdot \theta^n$. 
Violation of the Sufficient Conditions

In the sufficient conditions, \( C-0 \) can be enforced by making the sampling interval sufficiently small. Since current power meters [39] can sample data in seconds, this condition generally holds in practice. As such, we mainly focus on the violation of Conditions \( C-1 \) and \( C-2 \) in the following.

Conditions \( C-1 \) and \( C-2 \) may be violated when 1) the power range of an appliance is submerged by another’s power range or 2) the power range of an appliance is submerged by the power deviation of another appliance. To quantify the degree of such violations, we introduce the concept of ambiguous degree.

**Definition 16.** Given the power models of all appliances \( \Lambda = \{\lambda_1, \lambda_2, \cdots, \lambda_N\} \), the ambiguous degree is defined as:

\[
    d(\Lambda) := \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} I(\lambda_i, \lambda_j),
\]

where \( I(\lambda_i, \lambda_j) \) is an indicator function defined by:

\[
    I(\lambda_i, \lambda_j) = \begin{cases} 
        1, & \text{if } \lambda_i, \lambda_j \text{ violate } C-1 \text{ or } C-2 \\
        0, & \text{otherwise}
    \end{cases}
\]

The ambiguous degree \( d \in [0, \binom{N}{2}] \), with \( d = 0 \) indicating no violation of the sufficient conditions, and \( d = \delta \) indicating \( \delta \) pairs of appliances that violate the conditions. We will investigate the impact of ambiguous degree on the accuracy of state recovery and energy disaggregation in Sec. 4.4.5.

**4.4.4 Meter Deployment Optimization**

To achieve unambiguous state recovery and predictable energy disaggregation, we can partition the appliances into exclusive sub-groups and ensure that within each sub-group, the power model of appliances meets the sufficient conditions.

**Meter Deployment under Topology Constraint**

We have noticed that it would be inconvenient or even impossible to arbitrarily partition appliances, because it is desirable not to change existing circuitry in a building,
e.g., the dryer and the television may be located in different rooms. Therefore, appliances can be grouped together for measuring purpose when they are either proximal to share the same socket, or connected to the same power line so that the low-cost meter \[39\] can be used easily. We call such a requirement as *network topology constraint*. When two appliances cannot be clustered in the same sub-group, we call them *incompatible* (otherwise, they are compatible). Correspondingly, network topology constraint can be formally defined as:

**C-3:** If appliances \(i\) and \(j\) are incompatible and \(\lambda^i \in \Lambda^k\), then \(\lambda^j \notin \Lambda^k, \forall k\). \(4.41\)

**Remark 3.** While the need to consider topology constraint is obvious, optimization with network topology constraint may cause two concerns: the availability of topology information and the overhead of re-installing meters when a new appliance is introduced or when an existing appliance moves. Regarding the former concern, the topology information should be known to facility management of commercial buildings; in the worst case, a coarse-grained topology information, such as the level of floors, is easily obtained. Regarding the latter concern, it is practically unnecessary to re-deploy meters if the changes are not significant. This is due to the fact that once deployed, meters can tolerate a certain level of appliance changes, without causing obvious performance degradation in energy disaggregation results, as shown later in Sec. 4.4.5.

In order to lower the cost, it is desirable to minimize the total number of sub-groups, or equivalently, the minimum number of meters. This optimization problem can be formulated as follow:

**Input:** A group of \(N\) appliances with power model \(\Lambda\), and network topology constraints **C-3**.

**Output:** The minimum number of sub-groups, denoted by \(\{\Lambda^1, \Lambda^2, \cdots, \Lambda^n\}\).

\[
\begin{align*}
\min_{\{\Lambda^1, \Lambda^2, \cdots, \Lambda^n\}} & \quad n \\
\text{s.t.} & \quad \bigcup_{i=1}^{n} \Lambda^i = \Lambda, \\
& \quad \Lambda^i \cap \Lambda^j = \emptyset, i \neq j, \\
& \quad \Lambda^i = \{\text{appliances fit C-1, C-2 & C-3}\}.
\end{align*}
\]
Note that we exclude $C-0$ from the constraints since it may depend on human behaviour and thus cannot be controlled. To solve the above problem, a graph $G = (V, E)$ is constructed, where each vertex $v \in V$ represents the power model of an appliance and an edge is built between two vertices if the power models fit constraints $C-1$ and $C-2$, and the two appliances are compatible. It is easy to see that the problem is equivalent to the clique-covering problem, which has been proven to be NP-hard [59]. Hence, a greedy clique-covering algorithm is adopted to obtain an approximate solution.

The basic idea of the algorithm is to find cliques that cover more vertices that have not been clustered. Heuristically, the vertices with larger degrees may have a better chance of resulting in a smaller number of cliques. Thus, the search starts from the vertex with the largest degree, until all vertices are covered. Obviously, a resulted cluster is a clique in the graph. Since each vertex represents an appliance, a clique represents a sub-group, within which unambiguous state recovery is ensured (if $C-0$ holds in the dataset). We omit the pseudocode of the greedy algorithm for brevity.

**Complexity of Deployment Optimization**

**Lemma 4.** The computational complexity of meter deployment optimization is lower bounded by $O(N^2)$ and upper bounded by $O(N^3)$, where $N$ is the number of appliances.

**Proof.** For a sub-group of $n$ appliances, $1 \leq n \leq N$, to check the sufficient conditions of unambiguous state recovery and network topology constraints, i.e., $C-1$, $C-2$ and $C-3$, the number of comparisons is up to:

$$\sum_{\text{for } C-1} \frac{n^2}{2} + \sum_{\text{for } C-2} \frac{n}{2} + \sum_{\text{for } C-3} \frac{n}{2} = \frac{3n^2}{2} + \frac{n}{2}. \quad (4.43)$$

In the greedy clique-covering algorithm, a clique represents a sub-group of appliances. As a clique gets smaller, the number of appliances in the sub-group gets smaller and the number of comparisons decreases. Hence, two extreme situations yield the minimum and maximum numbers of comparisons, respectively: 1) all the appliances can be cluster into one group, and 2) each appliance becomes a sub-group. The first situation results in $N^2$ comparisons, and the second $O(N^3)$ comparisons,
because
\[ \sum_{n=1}^{N} \left(3n^2/2 + n/2\right) = \frac{N(N+1)^2}{2} \sim O(N^3). \] (4.44)

Therefore, the total number of comparisons in the meter deployment optimization is in \([O(N^2), O(N^3)]\).

Parallel Energy Disaggregation

After optimizing deployment of meters by solving (4.42), we can obtain \(n\) sub-groups of appliances and then perform energy disaggregation for each of them. Here, to speed up energy disaggregation among the \(n\) sub-groups, a simple yet efficient strategy is performing parallel energy disaggregation. With notations and outcomes from (4.42), the pseudo code of the straightforward parallel process is shown in Algorithm 5, in which \textit{parfor} is the same parallel \textit{for} loop in Algorithm 4.

**Algorithm 5** Parallel State Recovery among Sub-groups

**Input:** Power models of sub-groups \(\{\Lambda^1, \Lambda^2, \ldots, \Lambda^n\}\), aggregated power vectors of sub-groups \(\{x^1, x^2, \ldots, x^n\}\).

**Output:** Recovered state matrices of sub-groups \(\{S^1, S^2, \ldots, S^n\}\).

1: \textbf{parfor} \(i \leftarrow 1 : n\) \textbf{do}
2: \hspace{1em} \(x \leftarrow x^i\)
3: \hspace{1em} \(\Lambda \leftarrow \Lambda^i\)
4: \hspace{1em} Get \(S^i\) by applying Algorithm 4.
5: \textbf{end parfor}
6: \textbf{return} \(\{S^1, S^2, \ldots, S^n\}\).

After recovering the state matrix for each sub-group of appliances, the energy consumption of individual appliances can be estimated with the rated power information. Thus, we eventually achieve energy disaggregation for all the appliances. Furthermore, as we have mentioned, based on the power deviations of appliances, the upper and lower bounds of estimated energy consumption of individual appliances can also be provided.

4.4.5 Experimental Evaluation

In this section, benchmark NIALM approaches are selected and performance metrics are introduced. We first evaluate different approaches using real-world trace data.
from household appliances. Then synthetic load data from large-scale appliances are generated by Monte Carlo simulations and used to evaluate different approaches.

**Benchmark Approaches and Performance Metrics**

To compare the performance, we also implemented other NIALM approaches. Using the power information the same as in the SSER model, we implemented 1) a signature based approach, the Least Square Estimation (LSE) based integer programming method [37], and 2) a state transition based approach, the iterative Hidden Markov Model (HMM) [79].

To evaluate the error of energy disaggregation, the *Disaggregation Error* is usually used [2, 79, 81]. Furthermore, to validate our state recovery strategies, we also evaluate the accuracy of recovered appliances’ states via *Hamming Loss* [85]. Accordingly, we use $1 - \text{Disaggregation Error}$ and $1 - \text{Hamming Loss}$ to get the *accuracy* of energy disaggregation and state recovery, respectively. The performance metrics are defined as follows.

**Energy Disaggregation Accuracy (EDA):** It indicates the accuracy of assigning correct power values to corresponding appliances.

$$\text{EDA} := 1 - \frac{\sum_{n=1}^{N} \| p_n^r - s^n p^n \|_1}{2 \| x \|_1}, \quad (4.45)$$

in which $N$ is the total number of appliances, $p_n^r$, $s^n$ and $p^n$ represent the real power consumption vector, the recovered state vector, and the rated power vector of the $n$-th appliance, respectively, and $x$ is the aggregated power vector.

**State Recovery Accuracy (SRA):** It indicates the accuracy of recovering the states of appliances.

$$\text{SRA} := 1 - \frac{\sum_{n=1}^{N} \| s_n^r - s^n \|_1}{N \cdot T}, \quad (4.46)$$

in which $s_n^r$ and $s^n$ represent the real state vector and the recovered state vector of the $n$-th appliance, respectively, and $N, T$ represent the number of appliances and the number of samples, respectively.

**Real-World Evaluations**

As shown in Fig. 4.7, we construct a real-world distributed appliances power monitoring platform using the off-the-shelf solution provided by *Current Cost* (www.
currentcost.com). Individual power consumption data (real power measurements) of twelve household appliances (including microwave, refrigerator, desktop, and so on) are collected using the individual appliance monitors (IAMs) and pour into our database via the EnviR display (a sink node) every 10 seconds.

One-week load data was collected and used for performance evaluation. The prior knowledge of appliances’ rated power is learned from the users’ manual or from the public website [38]. The power deviations are estimated from the collected power readings of each appliance. To address the concern on the accuracy of estimated power deviation, we will perform robustness test in Sec. 4.4.5.

The deployment and performance results with our SIALM approach are summarized in Table 4.4, and the performance comparison between SIALM and NIALM approaches is summarized in Table 4.5. In addition, as illustrated in Fig. 4.8, we can check the overall energy disaggregation accuracy of the three methods, which indicates the energy contribution of each appliance to the total energy consumption in the whole time period. From the results, we can see that with only three extra meters, the performance of SIALM approach is much better than that of the two NIALM approaches. Specifically, the accuracy in both energy disaggregation and appliance state recovery with our SIALM approach is over 90%.

**Scalability Test with Synthetic Data**

Note that currently public datasets for energy disaggregation (such as [2]) are from small-scale appliances, and most of them are circuit-oriented and do not contain detailed appliances’ information. Therefore, in this section, synthetic data of large-scale appliances are generated and numerical evaluations are performed to validate the efficiency of our SIALM approach.
Table 4.4: Deployment and performance results from the SIALM approach with real data

<table>
<thead>
<tr>
<th>Sub-group</th>
<th>Appliance</th>
<th>EDA</th>
<th>SRA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Water Cooler</td>
<td>94.66%</td>
<td>96.47%</td>
</tr>
<tr>
<td></td>
<td>Microwave-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Refrigerator</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Microwave-2</td>
<td>91.19%</td>
<td>90.01%</td>
</tr>
<tr>
<td></td>
<td>Monitor-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Laptop</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Desktop-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Monitor-2</td>
<td>93.40%</td>
<td>95.17%</td>
</tr>
<tr>
<td></td>
<td>Desktop-2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Printer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Desktop-3</td>
<td>93.55%</td>
<td>96.79%</td>
</tr>
<tr>
<td></td>
<td>Kettle</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: Appliances in the first sub-group are located in the lounge room, while others are in a laboratory.

Table 4.5: Performance results from experiments with real data

<table>
<thead>
<tr>
<th>#Meters</th>
<th>EDA</th>
<th>SRA</th>
</tr>
</thead>
<tbody>
<tr>
<td>NIALM</td>
<td>LSE</td>
<td>33.40%</td>
</tr>
<tr>
<td></td>
<td>HMM</td>
<td>55.27%</td>
</tr>
<tr>
<td>SIALM</td>
<td>SSER</td>
<td>93.20%</td>
</tr>
</tbody>
</table>

Figure 4.8: Real and estimated energy contributions of each appliance to the total consumption for one week.

There is no standard model to generate load data of appliances, since the energy consumption actually results from a complex process related to human activities. We thus apply the Monte Carlo simulation to generate the load data for large-scale appliances:

- Given the number of appliances, $N$, the number of operating modes of each
appliance is uniformly assigned between 1 and \( \Delta (\Delta \geq 1) \), and \( K (K < N/2) \) pairs of incompatible appliances are manually chosen.

- Given the lowest power \( p_{\text{min}} \) and the highest power \( p_{\text{max}} \) of all appliances, the lower bound of one operating mode of an appliance \( p_l \) is a random variable uniformly distributed between \( p_{\text{min}} \) and \( p_{\text{max}} \). The upper bound of the operating mode \( p_u \) is determined by a parameter called \textit{power range ratio} \( r \) and is calculated by \( p_u = \min\{p_l + \operatorname{random}([0, r \cdot p_l]), p_{\text{max}}\} \), where \( \operatorname{random}([0, r \cdot p_l]) \) returns a random number uniformly distributed in the range \( [0, r \cdot p_l] \).

- Validate the unambiguous state recovery necessary constraints with the appliance’s power model. Referring to the sufficient conditions, partition the \( N \) appliances into multiple sub-groups using the greedy algorithm introduced in Sec. 4.4.4.

- For each sub-group of appliances: every appliance reports its current operating mode (a random number uniformly distributed between 1 and the maximum mode number of the appliance) and power reading (a random number uniformly distributed between the appliance’s power bounds). It also reports 0 if its state is \textit{off}. Then the aggregated power reading of all appliances (i.e., the sum of appliances’ power readings in the sub-group) is recorded.

- The occurrence of state switching event of an appliance follows a Poisson distribution with parameter \( \tau \).

The values of parameters used to generate the synthetic data are listed in Table 4.6.

Table 4.6: Parameter settings for power model and load data generation

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Appliances ( (N) )</td>
<td>30, 50, 100, or 200</td>
</tr>
<tr>
<td>Number of Operating Modes ( (\Delta) )</td>
<td>3</td>
</tr>
<tr>
<td>Number of Incompatible Pairs ( (K) )</td>
<td>2, 4, 8, or 10</td>
</tr>
<tr>
<td>Sampling Rate of Meters</td>
<td>0.1 Hz</td>
</tr>
<tr>
<td>Total Simulation length</td>
<td>24 hours</td>
</tr>
<tr>
<td>Lowest Appliance Power ( (p_{\text{min}}) )</td>
<td>20w</td>
</tr>
<tr>
<td>Highest Appliance Power ( (p_{\text{max}}) )</td>
<td>2000w</td>
</tr>
<tr>
<td>Power Range Ratio ( (r) )</td>
<td>[0.05, 0.15]</td>
</tr>
<tr>
<td>Poisson Parameter ( (\tau) )</td>
<td>180</td>
</tr>
</tbody>
</table>
Effectiveness of SIALM Approach: The synthetic data is used to test the performance of NIALM and SIALM approaches. The accuracy of energy disaggregation and state recovery, and the number of meters are summarized in Table 4.7. We can see that, with the help of a few extra meters, the accuracy of energy disaggregation can be significantly improved. From the first case where the number of appliances is 30, we can see that when the sufficient conditions for unambiguous state recovery are hold, the accuracy of recovered appliance states with SSER model can reach as high as 100%.

Table 4.7: Performance results of appliances deployment optimization and energy disaggregation: NIALM (N) vs. SIALM (S)

<table>
<thead>
<tr>
<th></th>
<th>30 Appliances</th>
<th>50 Appliances</th>
<th>100 Appliances</th>
<th>200 Appliances</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#* EDA SRA</td>
<td>#* EDA SRA</td>
<td>#* EDA SRA</td>
<td>#* EDA SRA</td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSE</td>
<td>1 52.85% 40.91%</td>
<td>1 47.61% 38.43%</td>
<td>1 41.31% 37.30%</td>
<td>1 40.35% 39.47%</td>
</tr>
<tr>
<td>IHMM</td>
<td>1 62.02% 56.23%</td>
<td>1 59.77% 58.40%</td>
<td>1 59.03% 58.33%</td>
<td>1 52.95% 52.81%</td>
</tr>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rand</td>
<td>7 *90.91% 85.02%</td>
<td>10 77.08% 66.26%</td>
<td>18 75.97% 70.94%</td>
<td>35 75.46% 68.74%</td>
</tr>
<tr>
<td>Opt.</td>
<td>7 *98.60% 100%</td>
<td>10 89.70% 89.94%</td>
<td>18 87.48% 85.45%</td>
<td>35 85.66% 84.30%</td>
</tr>
</tbody>
</table>

a # represents the number of meters used in corresponding situations.

b For EDA & SRA resulted from SSER model, * is labeled for the optimal value and others are approximate ones from the approximate parallel optimizations.

Effectiveness of Sufficient Conditions: By looking into the accuracy of state recovery from each sub-groups, we can validate the effectiveness of sufficient conditions for unambiguous state recovery. To fairly compare our SSER model with others, we calculate the performance metrics of different models (LSE, IHMM and SSER) within each sub-group of appliances using only one meter. The performance results for the first five sub-groups of 30 appliances are shown in Table 4.8. From the comparison, we can see that using the same number of meter for the same group of appliances, the accuracy of state recovery and energy disaggregation from our SSER model is higher than that from the other two disaggregation models. Furthermore, as shown in the table, since the power models of each sub-group of appliances follow the sufficient conditions, the accuracy of state recovery can reach one hundred percent, while for the other two models, it is not.

Effectiveness of Deployment Optimization: To validate the effectiveness of SIALM with respect to meter deployment, we first compare the optimal deployment with the random deployment using the same number of meters. The performance of two deployment strategies is shown in the last two rows of Table 4.7. Compared with the random meter deployment, the optimal deployment strategy is considerably better than the random deployment.
Table 4.8: Performance results of energy disaggregation for sub-groups of 30 appliances using different disaggregation models

<table>
<thead>
<tr>
<th>Subgroup</th>
<th>EDA</th>
<th>SRA</th>
<th>EDA</th>
<th>SRA</th>
<th>EDA</th>
<th>SRA</th>
<th>EDA</th>
<th>SRA</th>
<th>EDA</th>
<th>SRA</th>
<th>EDA</th>
<th>SRA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subgroup-1</td>
<td>LSE</td>
<td>83.69%</td>
<td>82.70%</td>
<td>84.97%</td>
<td>83.39%</td>
<td>84.20%</td>
<td>82.71%</td>
<td>80.63%</td>
<td>79.20%</td>
<td>84.11%</td>
<td>82.09%</td>
<td></td>
</tr>
<tr>
<td>Subgroup-2</td>
<td>IHMM</td>
<td>91.43%</td>
<td>93.39%</td>
<td>93.40%</td>
<td>96.01%</td>
<td>93.55%</td>
<td>95.10%</td>
<td>91.90%</td>
<td>92.82%</td>
<td>95.47%</td>
<td>96.88%</td>
<td></td>
</tr>
<tr>
<td>Subgroup-3</td>
<td>SSER</td>
<td>97.91%</td>
<td>100%</td>
<td>98.55%</td>
<td>100%</td>
<td>98.70%</td>
<td>100%</td>
<td>96.63%</td>
<td>100%</td>
<td>99.28%</td>
<td>100%</td>
<td></td>
</tr>
</tbody>
</table>

Note: As shown in Table 4.7, there are 7 sub-groups resulted from the deployment optimization of 30 appliances. Here we only show the first five ones for space limitation, and the performance values from the other two are quite similar to those from the first five.

To further validate the effectiveness of the deployment strategies under sufficient conditions, we compare the number of meters required in the SIALM approach with that from a real-time appliance state monitoring (RTASM) method in [82] that can also provide unambiguous state recovery. The numbers of meters needed for unambiguous state recovery with SIALM and RTASM are shown in Fig. 4.9. We can see that, for the same number of appliances, SIALM needs much fewer meters than RTASM. Since RTASM only uses current information (real-time load data) to achieve unambiguous state monitoring, it does not allow any power overlap between any pair of appliances in a sub-group. In contrast, by utilizing all historical load data, SIALM can tolerate power overlaps of appliances to some degree (referring to \( C-1 \) and \( C-2 \)), and thus can reduce the number of meters while still ensuring unambiguous state recovery.

Figure 4.9: Number of meters needed for unambiguous state recovery in SIALM and RTASM, respectively.
Robustness Test

In practice, our approach might be subject to the following concerns: 1) the estimated power deviation of an appliance may not be accurate, 2) the network topology may change, e.g., some new appliances may be introduced or appliances may be moved from one place to another. It is also interesting to explore the performance of our approach when the sufficient conditions do not hold.

Impact of Inaccurate Power Deviation

We replace $\theta$ with $\rho \cdot \theta$ in model (4.32) and model (4.42), so that the estimated power deviations can be narrowed down or widened up by regulating $\rho$. The value of $\rho$ is changed from 1.0 to 1.5, causing parametric errors of power deviation up to 50%. We do not consider the situation where appliances’ power ranges are narrowed, since intuitively a user can always widen an appliance’s power range if she is not sure about the real values.

From the results shown in Table 4.9, we can see that the accuracy does not change much when the parameter error varies, indicating that our method is robust to parameter estimation.

Table 4.9: Performance of SIALM approach with inaccurate estimation of power deviation

<table>
<thead>
<tr>
<th>$\rho$</th>
<th>30 Appliances</th>
<th>100 Appliances</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#meters</td>
<td>EDA</td>
</tr>
<tr>
<td>1.0</td>
<td>7</td>
<td>98.60%</td>
</tr>
<tr>
<td>1.1</td>
<td>8</td>
<td>94.49%</td>
</tr>
<tr>
<td>1.2</td>
<td>9</td>
<td>90.82%</td>
</tr>
<tr>
<td>1.3</td>
<td>9</td>
<td>89.20%</td>
</tr>
<tr>
<td>1.4</td>
<td>10</td>
<td>86.03%</td>
</tr>
<tr>
<td>1.5</td>
<td>10</td>
<td>84.61%</td>
</tr>
</tbody>
</table>

Impact of Network Topology Changes

For the situation where new appliances are added into the network topology, if the power meters are plug-and-play as the ones used in our case, we can easily re-deploy them according to the newly computed deployment solution. In cases where the power meters are hard to change, we further test the performance of our approach when new appliances are added while the meters remain the same. Specifically, with the optimal meter deployment, we add a small number of new appliances into the existing network
without changing the meters. In particular, one to five new appliances generated with
parameters in Table 4.6 are randomly added into existing appliance sub-groups. We
then test the accuracy of energy disaggregation with the existing meters. Note that
we do not consider the situation where some existing appliances are removed, because
under such situation the sufficient conditions for unambiguous state recovery still hold
and it does not bring any impact on the accuracy.

Part of the results with our SIALM approach are shown in Table 4.10. We can
see that the accuracy of energy disaggregation and the accuracy of state recovery
only slightly deteriorate. Similarly, we also test the case when a small number of
appliances move from one place to another and did not observe a large performance
degradation.

Table 4.10: Performance of SIALM approach with newly added appliances

<table>
<thead>
<tr>
<th>Num. of New Appliances</th>
<th>30 Appliances</th>
<th>100 Appliances</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EDA</td>
<td>SRA</td>
</tr>
<tr>
<td>0</td>
<td>98.60%</td>
<td>100%</td>
</tr>
<tr>
<td>1</td>
<td>97.77%</td>
<td>98.20%</td>
</tr>
<tr>
<td>2</td>
<td>95.15%</td>
<td>93.96%</td>
</tr>
<tr>
<td>3</td>
<td>95.45%</td>
<td>94.13%</td>
</tr>
<tr>
<td>4</td>
<td>93.70%</td>
<td>93.87%</td>
</tr>
<tr>
<td>5</td>
<td>92.12%</td>
<td>93.30%</td>
</tr>
</tbody>
</table>

Impact of Violating Sufficient Conditions

To explore the performance of our approach when the sufficient conditions do
not hold, we further test its performance with various ambiguous degrees. In the
tests, after obtaining the optimal deployment solution, we arbitrarily choose 3 ∼ 10
sub-groups of appliances and adjust their ambiguous degree to specified values by
manually changing appliances’ power models.

The results of accuracy vs. average ambiguous degree ($\bar{d}$) from the tests are
summarized in Table 4.11. According to the results, our approach can tolerate the
violation of sufficient conditions to a certain degree. For example, when $\bar{d} = 1, 2, 5$,
the accuracy of energy disaggregation and state recovery does not decrease too much.
Nevertheless, our approach suffers and cannot return a result when the ambiguous
degree reaches 10 in a group of 30 appliances. Note that this extreme case is unlikely
in practice, and it implies that the appliances are nearly indistinguishable in our
model, making energy disaggregation extremely hard.
Table 4.11: Performance of SIALM approach with sufficient condition violations

<table>
<thead>
<tr>
<th>$d$</th>
<th>30 Appliances</th>
<th>100 Appliances</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$EDA$</td>
<td>$SRA$</td>
</tr>
<tr>
<td>0</td>
<td>98.60%</td>
<td>100%</td>
</tr>
<tr>
<td>1</td>
<td>97.30%</td>
<td>97.69%</td>
</tr>
<tr>
<td>2</td>
<td>96.06%</td>
<td>95.71%</td>
</tr>
<tr>
<td>5</td>
<td>85.45%</td>
<td>86.13%</td>
</tr>
<tr>
<td>10</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

In summary, our SIALM approach is robust in the presence of inaccurate estimation of power deviation and small number of appliance changes. When the sufficient conditions do not hold, our approach can tolerate the ambiguous degree to a modest level.

4.5 SmartSaver: a Consumer-Oriented Web Service for Energy Disaggregation

4.5.1 Overview

More and more consumers can get access to their aggregated load data with the help of smart meters. There are, however, still stumbling blocks on the road to performing energy disaggregation by ordinary consumers. Even though a broad spectrum of solutions to energy disaggregation have been proposed since 1980s [18, 19], most are based on appliances’ energy usage patterns, also called signatures of appliances. Those signatures are hard to obtain without particular machine learning techniques or auxiliary measurements. For example, in [21], extra equipments are needed to detect the activities of appliances based on high frequency electromagnetic interference (EMI). Other solutions like [78, 80, 84, 79] need a lot of training or inference work, which is too complex to handle by ordinary consumers.

Recently, a rapidly growing number of start-ups, such as Bidgely, PlotWatt, Nave- tas, and Energy Aware, are established to provide commercial services for energy disaggregation in consumers’ houses. Those private services, however, are usually not cheap and without guarantee for accuracy. Furthermore, special measuring instruments need to install in consumers’ houses. So far, no free/open service/application has been found to provide public energy disaggregation for common households with-
out installing any extra equipments.

We have observed that 1) the power information of an appliance, such as rated and stand-by power, is normally available in practice, from users’ manual, technical specification or public web sites such as [38], and 2) low-cost plug-and-play power meters are popular on current market and in some consumers’ houses [39]. We challenge the traditional energy disaggregation solutions and try to 1) establish a simple and universal model for energy disaggregation only referring to readily-available information of appliances, 2) provide open web service for energy disaggregation to ordinary consumers. Our service does not require any expertise on data mining or machine learning, and customers can easily obtain energy disaggregation results using either their own historical load data from smart meters deployed by the utility or their own plug-and-play power meters to transmit load data to our web service in real-time.

4.5.2 Architecture and Functions

System Architecture

The logical architecture of SmartSaver is illustrated in Fig. 4.10. As shown in the figure, SmartSaver is a hybrid system combining the Client/Server (C/S) and Browser/Server (B/S) models. The core of the system is composed of a Data Storage, a Computing Node and a Web Server, which play roles of data storage and backup, load data analysis and energy disaggregation, and task submission and request response, respectively. The three components can be separated or combined, and can be virtualized on the Cloud as well (as most commercial energy disaggregation providers have done). On the consumer side, they can either upload real-time load data from their own plug-and-play meters via specific gateways, or submit historical power readings obtained from power suppliers in a point-and-click way via the browser. The load data, service requests and responses are transmitted over the Internet.

Functional Design

Under the above system architecture, SmartSaver includes three basic functions.  

Load Data Collection: The users can connect their own power meters embedded with communication components to our database and upload their real-time load data to the Data Storage over the Internet. For such users, they can not only be served
with energy disaggregation but also monitor the real-time states of their appliances. Alternatively, the users can also submit historical load data (i.e., the time series of meter readings) obtained from power suppliers to the Data Storage via the Web Server.

**Load Data Analysis and Mining:** With customers’ requests of energy disaggregation or appliance state monitoring submitted via the Web Server, the Computing Node can run specified energy disaggregation algorithms over the aggregated load data to mine the individual appliance’s energy consumption or states. The disaggregated energy consumption or recovered appliances’ states are returned to the Web Server.

**Load Data Visualization:** Via the browser on the PC or mobile terminals, cus-
tomers can either read their total energy consumption of multiple appliances directly from the Data Storage, or perform energy disaggregation for individual appliance by submitting disaggregation requests to the Computing Node. Both services use graphical representations to return user-friendly results.

Overall, SmartSaver can provide both online and offline energy disaggregation services. In the former, the real-time load data from customers are transmitted to the Data Storage, and results of energy disaggregation as well as appliances’ states are returned. In the latter, the historical load data from customers are uploaded to the Data Storage via the Web Server, and results of energy disaggregation within the considered time duration are returned. These two kinds of services are expected to fulfill the demands of energy disaggregation under different scenarios for most customers.

4.5.3 Implementation and User Cases

Online Load Data Collection

As mentioned before, the customers should be able to connect their power meters to our Data Storage and upload the load data in real-time. In our implementation, we choose the off-the-shelf products developed by Current Cost [39]. As shown in Fig. 4.11, the IAMs measure the power consumption of (individual or multiple) running appliances; the EnviR receives the data from IAMs, displays the power readings on the screen, and retransmits the load data to the gateway (the laptop in the figure); the laptop runs our client application (named as CC Data Collector) and forwards load messages to our remote Data Storage in real-time. On the Data Storage side, the received load messages (in XML format) are parsed and inserted into our predefined database.

In such a way, the real-time load data of consumers can be transmitted to the Data Storage, based on which we can support online energy disaggregation and appliance state monitoring. For the users who have no power meters or do not want to connect their meters to our Data Storage, they can upload their historical data to use our offline service, which will be introduced later.
Energy Disaggregation Algorithm

A parallel local optimization algorithm (PLOA) was proposed to solve the SSER model introduced in Chapter 4.3, and the theoretical analysis and practical validation of the algorithm can be found in [22]. We implement PLOA on our Computing Node that responds to the disaggregation requests submitted via the Web Server. With appliances’ power information (provided by the user along with the energy disaggregation requests) and their aggregated power readings, the PLOA can recover the states of individual appliances along the timeline and predict the energy consumption of individual appliances within the specified time period. The resulted states matrix and disaggregated values are returned to the customers via the Web Server.

Note that, besides PLOA, other tasks, such as data cleansing, re-sampling, and smoothing, are performed on the Computing Node. Furthermore, to make the system extendable, we provide an interface so that any other algorithms using only appliances’ aggregated data and similar power model can be added into the library on the Computing Node.
User Interface

A website is established on the Web Server, which enables consumers to use SmartSaver and perform energy disaggregation in a point-and-click way. Here we show some screenshots of our website and illustrate the major functions and services provided by SmartSaver. The operation and performance of SmartSaver have been captured in a video available at Youtube (https://youtu.be/_copD6Gkx2E) or Youku (http://v.youku.com/v_show/id_XOTM1Njg2NzYw.html).

Energy Consumption Display: For the users who have connected their power meters to our Data Storage, they can view the real-time energy consumption from each sensor (like IAM). As illustrated in Fig. 4.12, the energy consumption of appliances from different sensors is shown.

Online and Offline Energy Disaggregation Request: For the users who have connected their power meters to our Data Storage, as shown in Fig. 4.13-(a), they can submit requests for online energy disaggregation, by providing the sensor id (channel id and collector id) and time interval. For the users using offline energy disaggregation, as shown in Fig. 4.13-(b), they can submit requests for offline analysis, by completing appliances’ information and uploading a historical data file containing records of aggregated energy consumption.

Energy Disaggregation Results: Once the online/offline energy disaggregation is completed by the Computing Node, the results are returned to the Web Server. The
Figure 4.13: Website screenshot: request forms for online and offline analysis

Figure 4.14: Website screenshot: energy disaggregation results

energy consumption of individual appliances and their percentage in the total consumption are shown on the web page of customers, as illustrated in Fig. 4.14. Furthermore, for online energy disaggregation, the real-time on/off states of individual appliances will also be shown on the web page.
4.6 Conclusions

In Section 4.3, a simple, universal model for energy disaggregation was proposed. By making use of readily available information of appliances, we built a sparse switching event recovering model based on the sparsity of appliances’ switching events. Furthermore, we used the active epochs of switching events to develop a parallel local optimization algorithm to solve our model efficiently. In addition to analyzing the complexity and correctness of our algorithm, we tested our method with the real-world trace data from an energy monitoring platform. The test results demonstrated that our method can achieve better performance than the state-of-the-art solutions, including the Least Square Estimation (LSE) method and the machine learning method using iterative Hidden Markov Model (HMM).

In Section 4.4, we proposed a semi-intrusive appliance load monitoring (SIALM) approach to energy disaggregation for large-scale appliances. Instead of using only one meter, multiple meters were distributed in the power network to collect the aggregated load data from sub-groups of appliances. Based on a simple power model, we established a sparse switching event recovering (SSER) model and proposed a parallel optimization algorithm to recovery appliance states from the aggregated load data. We also provided the sufficient conditions for unambiguous state recovery of multiple appliances. Furthermore, under the sufficient conditions and network topology constraints, a minimum number of meters was searched for via a greedy clique-covering algorithm. To evaluate the performance of our SIALM approach, we not only used the real-world trace data from household appliances, but also applied Monte Carlo simulation to generate load data from large-scale appliances. The energy disaggregation accuracy and state recovery accuracy were compared with those from two benchmark NIALM approaches. The results showed that the SIALM approach can provide high-precision accuracy for appliance state recovery and improve the accuracy of energy disaggregation with a small number of extra meters.

In Section 4.5, we show SmartSaver, a consumer-oriented, user-friendly web service for energy disaggregation. By establishing sparse switching event recovery (SSER) model, we provide energy disaggregation service by only considering the power information of appliances. Using a Data Storage, a Computing Node and a Web Server as the core, we design the architecture and major functions of the system. Based on the logical design, we implement Smart Saver which offers both online and offline energy disaggregation services to consumers.
Chapter 5

Fine-Grained Power Monitoring in Datacenters

5.1 Overview

The energy expense has become one of the most significant operating costs in today’s datacenters. Companies like Amazon, Google, IBM, Microsoft, and Facebook, pay tens of millions of dollars every year for electricity [86]. A recent report [87] states that in US alone “datacenter electricity consumption is projected to increase to roughly 140 billion kilowatt-hours annually by 2020, the equivalent annual output of 50 power plants, costing American businesses $13 billion annually in electricity bills and emitting nearly 1004 million metric tons of carbon pollution per year.”

Fine-grained power monitoring, which refers to power monitoring at the server level (refer to Fig. 5.1), is critical to the efficient operation and energy saving of datacenters. Nevertheless, it is an extremely challenging task in datacenters that house diverse legacy servers as well as high-density blade servers and enclosures. While high-density computer systems greatly reduce the space of IT infrastructure and simplify the cabling process, many widely-used blade servers, such as DELL PowerEdge M100e, HPE ProLiant DL380 and some of IBM BladeCenter H series, are not equipped with power sensors, and power meters are typically installed at power distribution units (PDU) or at the rack level (refer to Fig. 5.1). In a legacy datacenter consisting of tens of racks, each with hundreds of servers not equipped with power sensors, how can we precisely capture the real-time power consumption of each server?
Figure 5.1: Power distribution hierarchy of the IT facilities in a typical datacenter.

Table 5.1: Comparison of current fine-grained power monitoring solutions vs. our NIPD solution

<table>
<thead>
<tr>
<th>Software/ Hardware</th>
<th>Monitoring Levels</th>
<th>Deployed Scale (# Server)</th>
<th>Homogeneous/Heterogeneous</th>
<th>Error Rate</th>
<th>Extra Cost ($/Server)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schneider [88]</td>
<td>hardware</td>
<td>intrusive</td>
<td>rack</td>
<td>any scale</td>
<td>both</td>
</tr>
<tr>
<td>SynapSense [31]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PowerExecutive [32]</td>
<td>hardware</td>
<td>intrusive</td>
<td>rack server</td>
<td>any scale</td>
<td>both</td>
</tr>
<tr>
<td>PowerPack [89, 90]</td>
<td>software</td>
<td>intrusive</td>
<td>rack server component</td>
<td>9</td>
<td>homogeneous</td>
</tr>
<tr>
<td>Mantis [91]</td>
<td>software</td>
<td>intrusive</td>
<td>rack server</td>
<td>1</td>
<td>–</td>
</tr>
<tr>
<td>ISCA’07 [92]</td>
<td>software</td>
<td>intrusive</td>
<td>rack server</td>
<td>≥100</td>
<td>homogeneous</td>
</tr>
<tr>
<td>TC’12 [93]</td>
<td>software</td>
<td>intrusive</td>
<td>rack server component</td>
<td>1</td>
<td>–</td>
</tr>
<tr>
<td>NIPD</td>
<td>software</td>
<td>non-intrusive</td>
<td>rack server</td>
<td>326</td>
<td>both</td>
</tr>
</tbody>
</table>

1. The *error rates* shown in brackets are the measurement errors of corresponding power meters or sensors.
2. The *hardware cost* of each solution is estimated by the current prices of measuring devices (or prices of similar products) on the market.
5.2 Related Work

In this section, we first review existing solutions for fine-grained power monitoring in the datacenter, including hardware-based power measuring and software-based power modeling [94] [95]. Then, focusing on server-level power monitoring, we briefly summarize the key points that need to be considered.

5.2.1 Fine-Grained Power Monitoring

Hardware-Based Power Measuring

Power monitoring with dedicated hardware is regarded as the most accurate yet expensive approach to obtain the fine-grained power information. For rack-level power monitoring, Schneider Electric [88] provided the metered Rack Power Distribution Units (RPDU). In addition, SynapSense [31] developed power monitoring solutions using devices like power clamps and intelligent power strips. Regarding the power monitoring at the server level, IBM developed its own power management system, PowerExecutive [32], which utilized the embedded sensors to measure the power usage and allowed users to monitor power consumption at the server level. For a large-scale datacenter not adopting the above equipments, the upgrade of power infrastructures and IT facilities would be prohibitively expensive.

Software-Based Power Modeling

This type of solution establishes power models to estimate the power consumption of a server, using information collected from the level of servers, components, or applications. Power models could be built for server, hardware components, or applications. Here, we only review the power models built for the servers, since they are mostly related to our work.

Server-level power models are usually trained based on the correlation between the state (or resource utilization) of individual hardware component and power consumption of corresponding component. In [96], using hardware performance monitoring counters (PMCs), a surrogate linear regression model was applied to build the power model and predict the power consumption of computer system. The notion of “ensemble-level” power management was proposed in [97], which leveraged usage patterns of concurrent resource across the individual server blades for power monitoring and saving. Power modeling with microprocessor PMCs was proposed in
Fine-Grained Datacenter Power Monitoring

Hardware-Based Power Measuring

Rack-Level
(e.g., Schneider PDU, SynapSense clamps)

Server-Level
(e.g., IBM PowerExecutive)

Software-Based Power Modeling

Intrusive Training
(e.g., PowerPack, Mantis, ISCA’07, TC’12)

Non-Intrusive Training
(Our NIPD Solution)

Figure 5.2: Classification of fine-grained power monitoring for datacenters.

[98]. Five different sever-level power models, which correlate AC power measurements with software utilization metrics, were investigated in [99], in which the accuracy and portability of the power models were also compared over different workloads and servers.

Various power monitoring platforms were developed utilizing software-based power modeling. PowerPack was initially established in [89] and further improved in [90]. It was implemented on a small-scale (9 nodes) cluster and supports power measuring of individual servers as well as power estimation of parallel applications. In [91], a hybrid hardware-software infrastructure, Mantis, was introduced. It first collected the individual server power consumption using an AC power meter. By correlating the power consumption data with system utilization metrics, it trained a linear regression model to predict server-level power consumption based on system utilization data. In [92], a power model was learned from the aggregated power of a few hundreds of homogeneous servers along with their corresponding CPU utilization.

5.2.2 Key Points of Server-Level Power Monitoring

Linear/Non-Linear Power Models

Among the developed power models for the server-level power estimation, the linear regression model is the most widely applied. It is simple and has been shown to yield accurate results [91, 96]. Nevertheless, there is evidence showing that nonlinear power models can also be good alternatives in some cases [100, 99]. Therefore, both linear
and nonlinear power models may need to be considered before we choose one of them for the power estimation at server level.

**Intrusive/Non-Intrusive Power Model Training**

To obtain the regression coefficients in the power model, a model training process (e.g., least square estimation) is needed. In existing power model training, either server-level or component-level power information is required. As such, we call these methods intrusive since power measuring at the server or lower levels is needed during the initial model training phase, even if afterwards no hardware-based power measuring is needed. As the difficulty of training data acquisition because of the integration of servers in enclosures, non-intrusive training can be a more practical approach.

We are thus motivated to develop a non-intrusive, purely software-based, fine-grained power monitoring solution. Our method is non-intrusive in the sense that it does not need power monitoring at the server or lower levels for initial model training. We use a novel technique, called non-intrusive power disaggregation (NIPD), to achieve fine-grained power monitoring in datacenters. NIPD establishes power mapping functions (PMFs) between the states of a server and its power consumption and uses PMFs to infer the power consumption of each server with the aggregated power of the entire datacenter. Compared with existing methods, our solution is unique, as illustrated in Fig. 5.2.

**5.3 NIPD: Overview and Rationale**

In this section, we clarify why we need non-intrusive power disaggregation (NIPD) in datacenters. Then, inspired by non-intrusive load monitoring for residential houses, we propose a new way to achieve NIPD in datacenters. We also overview the steps of NIPD.

**5.3.1 Why NIPD?**

To reduce the metering cost to zero, a software-based solution has to be adopted. As we have introduced in Sec. 5.2.1, a power model training process is needed for the software-based solution. The traditional intrusive model training is undesirable for legacy datacenters, since it is hard and tedious to obtain node-level power consump-
tion data from highly integrated rack that is needed for model training. Therefore, a non-intrusive power model training scheme needs to be developed.

As shown in Fig. 5.1, electric power for datacenters is usually supplied via the uninterruptible power supply (UPS) and power distribution units (PDUs). With the help of UPS/PDUs, we can easily get access to the aggregated power consumption of the datacenter\(^1\), from either the embedded meter [101] or certain interfaces (e.g., RS485 or RS232 serial interface) provided by the vendors. Such readily-available power readings, however, are aggregated power consumption from multiple racks of servers. To extract the fine-grained power information, we need to infer the power consumption of servers from the aggregated power readings, which is termed as power _disaggregation_.

In summary, we need non-intrusive training for building power model and power disaggregation to obtain the fine-grained power information. Hence NIPD comes naturally.

### 5.3.2 How to Develop NIPD?

The idea of separating aggregated power into individual units can be found in non-intrusive load monitoring (NILM) in residential houses [19, 20]. The NILM technology was initially proposed to separate the aggregated electricity consumption of a household into that of individual appliances. As it does not need any intrusive measuring in the house but only refers to the measurements from one meter outside the house, this technology has drawn much attention in energy conservation and demand response programs.

When applying existing NILM approaches in the datacenter environment, however, most assumptions in these approaches do not hold anymore. For example, servers do not turn on/off so often like household appliances. In addition, household appliances typically have their own power features, so-called appliances’ signatures, which are used in many NILM solutions. This property is not obvious in datacenters for multiple servers can have exactly the same power ratings. To the best of our knowledge, no NILM solution developed for household power monitoring can be applied directly in datacenters.

Is there any way to revamp the NILM technology for datacenter environment?

---

\(^1\)In the rest of the thesis, the power consumption of the datacenter refers to that of the IT facilities in particular. The power supply of others (e.g. cooling facilities) is out of the focus of this thesis.
With embedded firmwares in the server and easy-to-access interfaces, we can get the **component states**\(^2\) information of a server. At a particular instant, the component states from different servers are most likely different, even when they are fed with the same type of workloads. This observation motivates us to utilize the distinguishability of component states from individual servers to achieve power disaggregation.

Our procedure of NIPD for a datacenter is shown in Fig. 5.3. As illustrated in the figure, we first collect aggregated power information and component states information of all servers through the collection module. Then, the two types of information are processed in non-intrusive power model training (the NIPD model in the figure). Finally, with the trained power model, we estimate the power consumption of each individual server.

While the above procedure is clear, the details on modeling, training and using the power model need further explanation. To start with, we formally formulate the problem and present solutions accordingly in Section 5.4.

---

\(^2\)A component state of a server in this thesis refers to the instant index/value of one component’s utilization or working speed, such as the CPU or memory utilization, I/O speeds, or hardware performance monitoring counters (PMCs).
5.4 Model Design for NIPD

In this section, we formally define the problem of NIPD for fine-grained power monitoring in datacenters and develop solutions for training and updating power models used in NIPD.

5.4.1 Formal Definition

Without loss of generality, we consider a datacenter consisting of $m$ servers. We denote the aggregated power consumption of the $m$ servers sampled in time interval $[1, t]$ by an aggregated power vector as:

$$y := [y_1, y_2, \cdots, y_t]^\top,$$

and we denote the power consumption of the $i$-th ($1 \leq i \leq m$) server in the same time interval, which is unknown, by an individual power vector as:

$$y^{(i)} := [y_1^{(i)}, y_2^{(i)}, \cdots, y_t^{(i)}]^\top.$$

For the purpose of NIPD, we use the state information of components collected from each server, which is recorded in a state vector containing the $n$ scalars ($n$ is the number of components whose information is available):

$$s := [\mu_1, \mu_2, \cdots, \mu_n]^\top.$$

Accordingly, the state vector of the $i$-th server at time $j$ ($1 \leq j \leq t$) can be represented as:

$$s_j^{(i)} := [\mu_{1,j}^{(i)}, \mu_{2,j}^{(i)}, \cdots, \mu_{n,j}^{(i)}]^\top$$

in which $\mu_{\kappa,j}^{(i)}$ represents the value of the $\kappa$-th ($1 \leq \kappa \leq n$) component state in the $i$-th server at time instant $j$.

**Definition 17.** During a time interval $[1, t]$, given the aggregated power vector $y$ of $m$ servers and each server’s state vector $s_j^{(i)} \in \mathbb{R}^n$, $1 \leq i \leq m$, $1 \leq j \leq t$, the problem of non-intrusive power disaggregation (NIPD) is to estimate the power consumption of each individual server at each time instant, i.e., $y_j^{(i)}$, $1 \leq i \leq m$, $1 \leq j \leq t$. 
5.4.2 PMFs Modeling

To solve the NIPD problem, we first logically divide the servers in the datacenter into multiple virtual homogeneous clusters (VHCs), so that in each VHC the major hardware components (e.g., CPU, memory, disk and NIC) of servers are the same or similar (i.e., the same brand and similar capacity). Thus, if a datacenter is composed by \( r (r \geq 1) \) types of servers, we can divide the servers into \( r \) VHCs.

**Definition 18.** We define a state feature transformation (SFT) \( \phi : \mathbb{R}^n \rightarrow \mathbb{R}^{\tilde{n}} \), such that the original state vector \( s \in \mathbb{R}^n \) can be transformed to a dilated state vector \( x \in \mathbb{R}^{\tilde{n}} \), i.e.,

\[
x = \phi(s)
\]

and the elements in \( x \) (named state features) can be nonlinear-form of individual component states and/or their combinations.

**Example 1.** Given a state vector consisting of two elements, \( s = [\mu_1, \mu_2]^\top \), one of the possible dilated state vectors can be constructed as \( x = [1, \mu_1, \mu_2, \mu_1^2, \mu_1\mu_2]^\top \), where the SFT is defined as \( \phi : \mathbb{R}^2 \rightarrow \mathbb{R}^5 \).

With SFT, the original feature space of component states will be extended. Accordingly, the dilated state vector of the \( i \)-th server at time \( j \) (\( 1 \leq j \leq t \)) can be represented as:

\[
x_{(i)}^{(j)} := [x_{1,j}^{(i)}, x_{2,j}^{(i)}, \ldots, x_{\tilde{n},j}^{(i)}]^\top
\]

in which \( x_{\kappa,j}^{(i)} \) represents the value of the \( \kappa \)-th (\( 1 \leq \kappa \leq \tilde{n} \)) state feature for the \( i \)-th server at time instant \( j \).

We set the first state feature as a unit constant 1, i.e., \( x_1 = 1 \) or \( x_{1,j}^{(i)} = 1 \), which is convenient for the following power model representation. Note that using a different constant leads to different coefficient values but has no impact on the final NIPD results.

**Definition 19.** For servers in the same VHC, we define a power mapping function (PMF) \( f : \mathbb{R}^{\tilde{n}} \rightarrow \mathbb{R} \), such that the input of a server’s dilated state vector \( x \) at any time instant can yield its power consumption at corresponding time instant, i.e., for the \( i \)-th server’s dilated state vector at time \( j \), \( x_j, f(x_j^{(i)}) \) approximates \( y_j^{(i)} \).

According to the related works, both linear and nonlinear models have been explored to depict the relation between energy consumption of a server and its component states. As we have mentioned in Sec. 5.2.1, the linear power models are widely
used and have shown good power estimations. Nevertheless, there are also evidences showing that nonlinear power models may perform better in some situations.

As illustrated in Fig. 5.4, a quadratic relationship between the CPU frequency and system power consumption can be observed under condition of high frequency or overclocking. Furthermore, the correlation among the server components’ power consumption is usually ignored, as taking it into consideration can bring in nonlinear terms in the power model (e.g., the term of $\mu_1\mu_2$ in Example 1).

Figure 5.4: Total system power consumption with respect to three categories of CPU at high-end working frequency [4]. As a power model to depict the relationship between CPU frequency and system power consumption, the quadratic function is fitting more tightly than the linear one.

Therefore, we explore and compare the potentials of both linear and nonlinear power models by applying SFT in the PMFs modeling.

For servers in the same VHC, with the $\tilde{n}$-dimension dilated state vector $x$ from a user-defined SFT in (5.5), we formulate their PMF as follows:

$$f(x) = w^\top x$$

(5.7)

where $w$ is the coefficient vector denoted by:

$$w = [w_1, w_2, w_3, \ldots, w_{\tilde{n}}]^\top.$$  

(5.8)

Example 2. Given the state vector and SFT shown in Example 1, i.e., $x = [1, \mu_1, \mu_2, \mu_1^2, \mu_1\mu_2]^\top$, the PMF of the server can be formulated as:

$$f(x) = w_1 + w_2 \cdot \mu_1 + w_3 \cdot \mu_2 + w_4 \cdot \mu_1^2 + w_5 \cdot \mu_1\mu_2$$

(5.9)
where the coefficient vector \( w = [w_1, w_2, \cdots, w_5]^T \).

Note that as SFT can be either linear or nonlinear transformation of the state vector, the PMF in (5.7) thus can in an either linear or nonlinear form w.r.t. component states of each server in a VHC.

**Remark 4.** Conventional methods try to build a power model for each major component in a server, which is used to estimate the power consumption of each component. The server’s power consumption is approximated by the aggregate of the estimated power consumption of its major components. Our PMF can be regarded as a special type of power model, but it is different from conventional ones in that our PMF just indicates a way of mapping the dilated state features to the server’s overall power consumption. The power of uncovered components, e.g., fans within the server enclosure, will be properly “absorbed” (in the sense that \( f(x_j^{(i)}) \) best approximates \( y_j^{(i)} \)) by the modeled terms in PMF. Hence, the value of each term in PMF is not necessarily the true power value.

### 5.4.3 PMFs Training

For the overall power consumption of a server \( f(x) \), it can be broken down into two parts: idle power (or static power) and dynamic power [102]. The former is considered as the baseline power supplied to maintain the server system in an idle state, and the latter is the additional power consumption for running specific workloads.

As we have set the first state feature in the dilated state vector as a unit constant (i.e., \( x_1 = 1 \)), the first term in the PMF model (e.g., \( w_1 \) in Example 2) represents a constant and reflects the idle power. The left coefficients \( w_2, w_3, \cdots, w_\tilde{n} \) are thus associated with the dynamic power of a server system.

**Estimation of Coefficients**

We first estimate the coefficients of a server’s PMF. Assume that a datacenter consists of \( r \) VHCs, and \( m_\kappa \) servers are in the \( \kappa \)-th (\( 1 \leq \kappa \leq r \)) VHC. Moreover, each server of the \( \kappa \)-th VHC reports a \( n_\kappa \)-dimensional state vector, and with a user-defined SFT, the state vector is transformed to a \( \tilde{n}_\kappa \)-dimensional dilated state vector \( x \). Then, with the dilated state vector, the PMF for the VHC can be expressed as:

\[
f_\kappa(x) = (w^{(\kappa)})^T x
\] (5.10)
where \( w^{(\kappa)} \) is the coefficient vector of PMF for the \( \kappa \)-th VHC denoted as:

\[
w^{(\kappa)} = \left[ w^{(\kappa)}_1, w^{(\kappa)}_2, \ldots, w^{(\kappa)}_{\tilde{n}_\kappa} \right]^\top.
\] (5.11)

At an arbitrary time instant \( j \), the aggregated power consumption of the \( \kappa \)-th VHC can be expressed as:

\[
\hat{y}_j = (w^{(\kappa)})^\top \hat{x}^{(\kappa)}_j,
\]
where

\[
\hat{x}^{(\kappa)}_j = \left[ \sum_{i=1}^{m_\kappa} x^{(i)}_{1,j}, \sum_{i=1}^{m_\kappa} x^{(i)}_{2,j}, \sum_{i=1}^{m_\kappa} x^{(i)}_{3,j}, \ldots, \sum_{i=1}^{m_\kappa} x^{(i)}_{\tilde{n}_\kappa,j} \right]^\top.
\] (5.12)

Meanwhile, the aggregated power consumption of the whole datacenter (or \( r \) VHCs) can be expressed as:

\[
y_j = \tilde{w}^\top \tilde{x}_j,
\]
where

\[
\tilde{x}_j = \left[ \hat{x}^{(1)}_j, \hat{x}^{(2)}_j, \ldots, \hat{x}^{(r)}_j \right]^\top,
\] (5.13)
and

\[
\tilde{w} = \left[ w^{(1)}, w^{(2)}, \ldots, w^{(r)} \right]^\top,
\] (5.14)
in which \( \hat{x}^{(\kappa)}_j \) and \( w^{(\kappa)} \) are defined by (5.12) and (5.11), respectively. Refer to Appx. B.1.1 and Appx. B.1.2 for the detailed transform.

With the measured aggregated power vector of the whole datacenter \( y \) (in form of (5.1)), the following least square estimation (LSE) problem is formulated as the training model for the \( r \) PMFs of the datacenter:

\[
\min_{\tilde{w}} \sum_{j=1}^{t} (\tilde{w}^\top \tilde{x}_j - y_j)^2.
\] (5.15)

By solving the above problem, we can obtain the optimal coefficients for the \( r \) PMFs appearing in \( \tilde{w} \), with which we can estimate the power consumption of individual servers in different VHCs by providing corresponding dilated state vectors.

**Remark 5.** The LSE problem represented in (5.15) belongs to linear regression, and to solve the problem is trivial (with closed form given in Appx. B.3). Therefore, rather than the way to solve problem (5.15), it is the way to represent and formulate the NIPD into a solvable form that is the major innovation of this work.

Nevertheless, the above LSE training model cannot capture multiple but only one constant term appearing in the coefficient vector [99]. Consequently, if there are
more than one VHC in the datacenter \((r > 1)\), the resulted constant terms (i.e., \(w^{(1)}_1, w^{(2)}_1, \cdots, w^{(r)}_1\)) from (5.15) are not accurate. In other words, the idle power of servers in each VHC cannot be estimated by this model. Therefore, further approaches need to be developed to estimate the constant terms in PMFs.

**Estimation of Constant Terms**

A widely used energy saving strategy in many datacenters is to shutdown idle servers. They will be turned on again when the working servers cannot satisfy the workload \([103, 104, 105]\). Such a strategy provides us with an opportunity to estimate the constant terms in PMFs.

**Definition 20.** For a datacenter with \(r\) VHCs, at an arbitrary time instant \(j\), if \(h\) servers are turned off (or on), and meanwhile a power decrease (or increase) in the aggregated power consumption of the whole datacenter, \(\Delta y (\Delta y > 0)\), is detected, we call that an off/on event is captured. We use \(\Delta y > 0\) to indicate that only the absolute value is considered in our late problem formulation.

According to our real-world experiments illustrated in Fig. 5.5, although the aggregated power of the whole datacenter always fluctuates over time, we are still able to capture the off/on events without turning on/off a large proportion of servers.

Assume that \(t\) off/on events have been captured in the datacenter consisting of \(r\) VHCs. For the \(j\)-th \((1 \leq j \leq t)\) off/on event, a **counting vector** can be defined as:

\[
d_j := \left[ d^{(1)}_j, d^{(2)}_j, \cdots, d^{(r)}_j \right]^\top,
\]

\((5.16)\)
where \( d_j^{(\kappa)} \) stands for the number of turned-off (or turned-on) servers in the \( \kappa \)-th VHC at time \( j \), and the detected (mean) power decrease (or increase) is \( \Delta y_j \). Then the following optimization problem can be formulated to find the optimal estimation of the constant terms, i.e., \( w_1 = [w_1^{(1)}, w_1^{(2)}, \cdots, w_1^{(r)}]^\top \):

\[
\min_{w_1} \sum_{j=1}^{t} (w_1^\top d_j - \Delta y_j)^2. \tag{5.17}
\]

**Remark 6.** In the estimation of constant terms of PMFs, we can combine the optimization strategy using (5.17) and the manual setup with information from technical specification of servers. For servers that can be shut down, e.g., the computing nodes, it is easy to gather off/on events and estimate the idle power via the optimization method. For other IT units that cannot be shut down during the operation of datacenter, e.g., the admin nodes, the best way is to refer to the server’s technical specification or approximate their idle power using the information from other servers equipped with similar hardware components.

So far, we have introduced the details of building PMFs. The datacenter operators can then use PMFs to estimate the real-time power consumption of individual servers by referring to real-time component states from corresponding servers.

### 5.4.4 Adaptive PMFs Update

According to the analysis in Appx. B.3, the complexity of PMFs training relies on two metrics: the number of training data and the number of state features. Specifically, the training complexity has a linear growth with the increase of training data and a quadratic growth with the increase of state features. Therefore, it is critical to choose an appropriate number of training data as well as state features, so that the training process is lightweight while the resulted PMFs is accurate enough.

**Selective Training Data Collection**

To make PMFs as accurate as possible, we need a training dataset that contains complete states in the feature space, i.e., all possible state features of the servers in each VHC should be included in the training dataset. Nevertheless, in real-world datacenter operations, it is hard to stress each of the components in a server to work through all possible states. The training dataset collected in a time interval of several
hours or even several days may be incomplete. In other words, there is no guarantee that the training dataset covers all possible state features.

Simply collecting training data as much as possible, however, is not a good solution to the above problem due to two reasons: (1) the larger the training dataset, the higher the overhead in PMF training, and (2) more redundant data entries will be collected while they do not contribute to the improvement of PMFs. Therefore, we develop a selective data collection strategy as follows.

**Preprocessing:** we first set an update time interval for the training dataset, denoted as $\Delta t_1$. At an arbitrary time instant $j$, the dilated state features from $r$ VHCs can be expressed as $\tilde{x}_j$ in form of (5.13). Along with the measured aggregated power consumption of the datacenter at the same moment $y_j$, a data entry can be represented as $(\tilde{x}_j, y_j)$. With data entry of $(\tilde{x}_j, y_j)$, the process of selective training data collection is as below:

- **Step 1.** normalize each element in $\tilde{x}_j$ with the corresponding maximum value, i.e., rescale the values of each element to $[0, 1]$.
- **Step 2.** compare the normalized data entry with those in the training dataset. If it already exists, go to **Step 4**; otherwise, go to **Step 3**.
- **Step 3.** insert $(\tilde{x}_j, y_j)$ into the training dataset as a new entry.
- **Step 4.** backup the power value $y_j$ for the existed entry with the same component states.

Note that in the fourth step, we do not simply discard the redundant entry, but keep record of its power value. Thus, one data entry in the training dataset may have multiple power values, and we calculate their median as the final value used for PMFs training. Using the median can alleviate the affect of outliers [106] and make the PMFs’ training more robust. In addition to the gathering of state features, the same strategy can also be applied to collect the off/on events for constant terms estimation introduced in Sec. 5.4.3.

**Remark 7.** For our selective data collection strategy, the resolution of the normalized state features determines the maximum number of data entries in the training dataset. Assuming that a datacenter consists of $r$ ($r \geq 1$) VHCs, each with $\tilde{n}_\kappa$ ($1 \leq \kappa \leq r$) state features.

---

3The maximum value could be found from technical specification, e.g., maximum I/O speed, or if unknown, it could be set as a value higher than any possible values of the state feature.
features, and the preset resolution of normalized state features is $p$ ($0 < p \ll 1$), then the number of data entries in the training dataset is upper-bounded by $\sum_{n=1}^{p} \lceil \frac{1}{p^n} \rceil$. Refer to Appx. B.2 for the proof.

We can update training dataset at a regular basis, e.g., every $\Delta t_1$ interval time. Theoretically, with the above data collection strategy, the training dataset will become complete as time goes on. Meanwhile, we use the most updated dataset to perform PMFs training introduced in Sec. 5.4.3 at a regular basis, e.g., every $\Delta t_2$ interval time.

Iterative State Feature Elimination

There are multiple tools that can be used to collect system component states of a server, e.g., the hardware performance monitoring counters (PMC) and dstat tool [107]. According to [108], as many as 86 PMC states were selected from all available ones. As to the dstat tool that will be used in our experiments, it can provide up to 16 different states. Simply gathering all component states provided by these tools can result in a large state vector. Furthermore, with our state feature transformation introduced in Sec. 5.4.2, the dilated state vector for PMFs training will be even larger. To control the overhead in model training, we need to limit the number of state features, especially when the number of training data entries is already huge.

So far, it is still challenging to choose the most relevant and effective state features to construct the power model. Current approaches (like principal component analysis (PCA) [109]) usually falls into the supervised learning and thus need intrusive ground-truth measuring. Since intrusive measuring is not feasible in our context, a new state feature selection approach is needed.

To deal with this problem, we propose and perform an iterative state feature elimination process, which does not need supervised learning and can significantly reduce the irrelevant state features in PMFs. The process is shown by following steps:

- **Step 1.** Initially choose potential state features from the the raw data, and discard the irrelevant or redundant state features, e.g, the CPU idle and CPU utilization are actually equivalent and one of them can be discarded.

- **Step 2.** Apply the left state features to formulate the preliminary PMFs, and
Algorithm 6 Selective Training Data Collection

Input: Sampling interval $\Delta t_0$, training dataset update interval $\Delta t_1$
Output: Training dataset $\mathcal{D}$

1: $\mathcal{D} \leftarrow \emptyset$
2: while True do
3: \hspace{1em} $t \leftarrow \text{gettime}()$
4: \hspace{1em} if $t \mod \Delta t_1 == 0$ then
5: \hspace{2em} for $j \leftarrow 1 : \Delta t_0 : \Delta t_1$ do
6: \hspace{2em} \hspace{1em} compose the data entry $(\tilde{x}_j, y_j)$
7: \hspace{2em} \hspace{1em} normalize each element in $\tilde{x}_j$
8: \hspace{2em} \hspace{1em} if $\tilde{x}_j \notin \mathcal{D}$ then
9: \hspace{2em} \hspace{2em} $\mathcal{D} \leftarrow \mathcal{D} \cup (\tilde{x}_j, y_j)$
10: \hspace{2em} \hspace{1em} else
11: \hspace{2em} \hspace{2em} backup power value $y_j$
12: \hspace{2em} \hspace{1em} end if
13: \hspace{1em} end for
14: \hspace{1em} end if
15: end while

then conduct PMFs training following the routine introduced in Sec. 5.4.3 when selected training dataset is small.

- **Step 3.** When the number of training data entries exceeds a pre-defined threshold value $n_\delta$, we check the coefficient of each state feature: if its absolute value is approximate to zero, e.g., less than $1 \times 10^{-3}$ in our implementation, eliminate this term in the corresponding PMF.

The pseudocodes of selective training data collection and iterative state feature elimination are illustrated in Algorithm 6 and Algorithm 7, respectively. The two processes that cooperate together can achieve adaptive PMFs updating, with which we can significantly reduce the PMFs training overhead. As explained in Remark 7, the number of the training data entries using selective data collection is not large (less than $10K$ in our latest experiment). Furthermore, by applying iterative state feature elimination, we will show that a small number of state features can be resulted and sufficient to provide accurate PMFs in Sec. 5.6.

Overall, to recap our NIPD solution introduced in this section: at background the PMFs are adaptively updated by selective training dataset collection and iterative state features elimination; at foreground the real-time component state information is fed into the most updated PMFs to obtain the server-level power estimations.
Algorithm 7 Iterative State Feature Elimination

**Input:** Dilated state vector $x$, PMFs update interval $\Delta t_2$, training dataset $\mathcal{D}$, elimination threshold $n_\delta$

**Output:** Updated PMFs

1. initialize preliminary PMFs with $x$
2. while True do
3. $t \leftarrow \text{gettime}()$
4. if $t \mod \Delta t_2 == 0$ then
5. if $|\mathcal{D}| < n_\delta$ then
6. update PMFs coefficients with $\mathcal{D}$
7. else
8. reformulate PMFs: eliminate state features with coefficients approximate to zero
9. end if
10. end if
11. end while

5.5 Implementation

We implement our NIPD solution over a real-world 326-node server cluster. It consists of 12 (blade) server racks that house 306 CPU nodes, 16 disk array nodes, 2 I/O index nodes, and 2 admin nodes, each running a Linux kernel. Table 5.2 shows the detailed configuration of each type of server. Fig. 5.6 illustrates the power architecture, network topology, and data collection modules of the experimental environment.

<table>
<thead>
<tr>
<th>Type</th>
<th>Configurations</th>
<th>Num.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Node</td>
<td>2\times Intel Xeon E5-2670 8-core CPU(2.6G)</td>
<td>306</td>
</tr>
<tr>
<td></td>
<td>8\times8GB DDR3 1600MHz SDRAM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1\times300G 10000rpm SAS HDD</td>
<td></td>
</tr>
<tr>
<td>Disk Array Node</td>
<td>1\times Intel Xeon E5-2603 4-core CPU(1.8G)</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>4\times4GB DDR3 ECC SDRAM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1\times300G 10000rpm SAS HDD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>36\times900G SAS HDD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Networking Switches</td>
<td></td>
</tr>
<tr>
<td>I/O Index Node</td>
<td>2\times Intel Xeon E5-2603 4-core CPU(1.8G)</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>8\times4GB DDR3 ECC SDRAM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1\times300G 10000rpm SAS HDD</td>
<td></td>
</tr>
<tr>
<td>Admin Node</td>
<td>2\times Intel Xeon E5-2670 8-core CPU(2.6G)</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>8\times16GB DDR3 1600MHz SDRAM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1\times300G 10000rpm SAS HDD</td>
<td></td>
</tr>
</tbody>
</table>
5.5.1 Data Collection and Feature Elimination

As shown in Fig. 5.6, we collect aggregated power consumption of the IT infrastructure via the UPS interface and a power monitoring proxy (P-1 in the figure). The sampling interval is 2 seconds. Besides the UPS, our datacenter is further equipped with 6 Power Data Management Modules (PDMMs) as part of the PDUs, each of which can provide power measuring at the rack-level, also at the sampling interval of 2 seconds. To verify our power estimation at the rack-level in Sec. 5.6.2, we also collect the power consumption of each rack via corresponding PDMM using the rack proxies (P-2 in Fig. 5.6).

In addition to the collection of power consumption data, the admin node is used to collect the component state information from each node (with sampling interval of 1 second). Particularly, we use dstat tool [107], a widely-used resource statistic tool that can gather various component states of a server, as shown in Table 5.3.
Table 5.3: State metrics collected using dstat tool

<table>
<thead>
<tr>
<th>Component</th>
<th>Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>processor</td>
<td>usr</td>
<td>CPU utilization for user processes</td>
</tr>
<tr>
<td></td>
<td>sys</td>
<td>CPU utilization for system processes</td>
</tr>
<tr>
<td></td>
<td>idle</td>
<td>CPU in idle</td>
</tr>
<tr>
<td></td>
<td>wai</td>
<td>CPU utilization for I/O waiting</td>
</tr>
<tr>
<td>memory</td>
<td>used</td>
<td>memory usage for processes</td>
</tr>
<tr>
<td></td>
<td>buff</td>
<td>buffer memory</td>
</tr>
<tr>
<td></td>
<td>cach</td>
<td>cache memory</td>
</tr>
<tr>
<td></td>
<td>free</td>
<td>free memory</td>
</tr>
<tr>
<td>disk</td>
<td>read</td>
<td>disk reading amount</td>
</tr>
<tr>
<td></td>
<td>write</td>
<td>disk writing amount</td>
</tr>
<tr>
<td>network</td>
<td>recv</td>
<td>traffic amount that the system received</td>
</tr>
<tr>
<td></td>
<td>send</td>
<td>traffic amount that the system sent</td>
</tr>
<tr>
<td>paging</td>
<td>in</td>
<td># pages changed from disk to memory</td>
</tr>
<tr>
<td></td>
<td>page</td>
<td># pages changed from memory to disk</td>
</tr>
<tr>
<td>system</td>
<td>int</td>
<td>system interruption time</td>
</tr>
<tr>
<td></td>
<td>csw</td>
<td>content switch times</td>
</tr>
</tbody>
</table>

Note that other tools can also be used here, such as vmstat, iostat, mpstat and netstat. To ensure the time synchronization between the component state collection and aggregated power collection, we set their clock source the same as Time Stamp Counter (TSC).

We first applied all the states terms listed in Table 5.3 to establish the preliminary PMFs. Then, we adopted the process of state feature elimination introduced in Sec. 5.4.4 and iteratively reduced the number of state features provided by dstat to six: total CPU utilization, total memory utilization, disk reading/writing and network traffic receiving/sending. With the six component states, according to (5.3), the state vector can be represented as:

\[ s = [\mu_1, \mu_2, \ldots, \mu_6]^T. \] (5.18)

When constructing nonlinear PMFs, we first included all second-order items with states in (5.18). Then, the state feature elimination was applied to minimize the dilated feature space. At last, only the nonlinear terms of \( \mu_1^2 \) and \( \mu_1 \mu_2 \) left, which capture the quadratic trend shown in Fig. 5.4 and the correlation between CPU utilization and memory usage, respectively.

In real-world datacenter environment, there are moments of component failures,
e.g., some servers or ToR switch may be down. Under this situation, the component states of some servers may not be available. To alleviate the impact of component failures and increase the reliability, we can take the following two measurements when implementing NIPD:

- During the PMFs training, special caution should be taken to make sure that only the data entries containing the state information of all servers are inserted into the training dataset. In this way, we guarantee that the trained PMFs are not affected by component failures.

- Once PMFs are obtained, we only estimate the power of servers whose component states are available. For servers whose component states are not accessible, we should not assign them zero power. Instead, we do not estimate their power and correspondingly raise a warning message to the administrator.

5.5.2 Estimation of Idle Power

For the estimation of idle power (or constant terms in PMFs) of CPU nodes in our experiments, we identify the idle nodes and remotely turn them off and on. For purpose of remote operation, the industry-standard IPMI [110] is used to turn on/off servers. During the on/off time period, multiple off/on events and corresponding power changes are captured from the event logs and data logs, respectively (as illustrated in Fig. 5.5), which are fed into the optimization model (5.17) to estimate the constant terms (idle power) of CPU nodes. As for the estimation of idle power of I/O and admin nodes, they are not allowed to be shut down for the normal operation of a running datacenter. Since the number of these two server types is quite small (only 2 for each type) and their hardware configurations are similar with that of CPU nodes, we set their idle power as the same as that of CPU nodes. The disk array nodes also need to be kept on all the time. Thus we infer their idle power from their working power range by making use of rack power: in our experimental datacenter, some racks only contain CPU nodes and disk arrays, so we can shut down all the CPU nodes and only leave the disk arrays running to obtain the idle power.
5.6 Evaluation

In our experimental environment introduced in Sec. 5.5, we evaluate the performance of different (linear and nonlinear) PMFs for power monitoring at the rack level and the server level, respectively.

5.6.1 Experiment Configuration

Table 5.4: Parameter settings of our experiments

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of VHCs ( (r) )</td>
<td>4</td>
</tr>
<tr>
<td>Number of component states ( (n_\kappa) )</td>
<td>([6, 6, 6, 6])</td>
</tr>
<tr>
<td>Dilated state vectors ( (x) )</td>
<td>Linear</td>
</tr>
<tr>
<td></td>
<td>Nonlinear-1</td>
</tr>
<tr>
<td></td>
<td>Nonlinear-2</td>
</tr>
<tr>
<td></td>
<td>([1, s]^\top)</td>
</tr>
<tr>
<td></td>
<td>([1, \mu_1^2, s]^\top)</td>
</tr>
<tr>
<td></td>
<td>([1, \mu_1^2, \mu_1\mu_2, s]^\top)</td>
</tr>
<tr>
<td>Normalizing resolution ( (p) )</td>
<td>0.01</td>
</tr>
<tr>
<td>Training dataset update interval ( (\Delta t_1) )</td>
<td>2 seconds</td>
</tr>
<tr>
<td>PMFs update interval ( (\Delta t_2) )</td>
<td>5 minutes, 0.5 hour</td>
</tr>
<tr>
<td>Elimination threshold ( (n_\delta) )</td>
<td>8,000</td>
</tr>
</tbody>
</table>

*The \( s \) in dilated state vectors refers to the state vector defined by (5.18).

Table 5.4 summarizes the setting of parameters in our experiments. We setup the parameters based on the following considerations:

- **Number of VHCs \( (r) \):** According to Table 5.2, we can logically divide the whole datacenter into 4 VHCs, and the number of servers in each VHC is 306, 16, 2, 2, respectively.

- **Number of component states \( (n_\kappa) \):** As introduced in Sec. 5.5.1, we select 6 component states for each individual server based on the information provided by Table 5.3.

- **Dilated state vectors \( (x) \):** Based on the original state vector of (5.18), we establish one linear PMFs and two nonlinear PMFs via the SFT showing in the table. Specifically, for PMFs with one nonlinear term (i.e., \( \mu_1^2 \)), we consider the quadratic trend shown in Fig. 5.4; for the PMFs with two nonlinear terms (i.e., \( \mu_1^2 \) and \( \mu_1\mu_2 \)), besides the quadratic trend, we also consider the correlation between CPU utilization and memory usage.
• Normalizing resolution ($p$): In the update of training dataset, we set the resolution of normalized data in each entry as 0.01, which is proved to be precise enough for accurate PMFs training, as shown in Sec. 5.6.2. According to Remark 7, a higher resolution will increase the size of training dataset as well as PMFs training complexity.

• Interval for updating training dataset ($\Delta t_1$): As the sampling interval for aggregated power consumption in our case is 2 seconds, we set the update interval of training dataset to the same value in order to collect training data quickly.

• PMFs update interval ($\Delta t_2$): We first set this value as 5 minutes, which is based on the estimation of PMFs training time needed under the theoretical maximum size of training dataset. As time goes, having observed that the training dataset size tends to be constant, we then change the update interval to 0.5 hour to reduce the overhead of PMFs update.

• Elimination threshold ($n_4$): When the number of training data entries exceeds 8,000, the resulted PMFs coefficients tend to be accurate (with mean relative error less than 15%), which can be verified by Fig. 5.7a.

<table>
<thead>
<tr>
<th>Workload</th>
<th>Description</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>Background OS processes</td>
<td>Server-level validation</td>
</tr>
<tr>
<td>Peak</td>
<td>Stress CPU usage to 100% malloc memory till 100%</td>
<td></td>
</tr>
<tr>
<td>SPECint</td>
<td>gcc</td>
<td>Compiler</td>
</tr>
<tr>
<td></td>
<td>gobmk</td>
<td>Artificial Intelligence: go</td>
</tr>
<tr>
<td></td>
<td>sjeng</td>
<td>Artificial Intelligence: chess</td>
</tr>
<tr>
<td></td>
<td>omnetpp</td>
<td>Discrete Event Simulation</td>
</tr>
<tr>
<td>SPECfp</td>
<td>namd</td>
<td>Biology/Molecular Dynamics</td>
</tr>
<tr>
<td></td>
<td>wrf</td>
<td>Weather Prediction</td>
</tr>
<tr>
<td></td>
<td>tonto</td>
<td>Quantum, Chemistry</td>
</tr>
<tr>
<td>IOZone</td>
<td></td>
<td>File system benchmark tool</td>
</tr>
<tr>
<td>Our Synthetic</td>
<td>Occupy CPU randomly</td>
<td>Rack-level validation</td>
</tr>
<tr>
<td></td>
<td>Read/write memory randomly</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.5: Workloads/benchmarks for NIPD evaluations
Putting the real-time component state information of the servers into the corresponding PMFs, we can get the estimated power consumption of each server. The estimated power consumption of servers in the same rack are aggregated as the estimated power consumption of the rack. To measure the error rate of our rack-level estimation, we apply the widely used metric of \textit{mean relative error (MRE)} defined by:

\[
\text{MRE} := \frac{1}{t} \sum_{j=1}^{t} \left| \frac{y'_j - y_j}{y_j} \right|
\]  

(5.19)

where \( t \) is the number of data entries in the dataset, and \( y_j \) and \( y'_j \) are the ground truth and estimated rack power for the \( j \)-th data entry, respectively.

By running different benchmarks shown in Table 5.5, we collect training data and duly update PMFs following the strategies introduced in Sec. 5.4.4. Furthermore, after each PMFs update, we run our synthetic workloads, collect rack power consumption and server component states, and calculate MRE of the power estimation with updated PMFs.

Figure 5.7: The overview of MRE from three PMFs along with the training data size (left) and that of a zoomed-in view when MRE \( \leq 5.0\% \) (right).

### 5.6.2 Power Monitoring at Rack Level

Putting the real-time component state information of the servers into the corresponding PMFs, we can get the estimated power consumption of each server. The estimated power consumption of servers in the same rack are aggregated as the estimated power consumption of the rack. To measure the error rate of our rack-level estimation, we apply the widely used metric of \textit{mean relative error (MRE)} defined by:

\[
\text{MRE} := \frac{1}{t} \sum_{j=1}^{t} \left| \frac{y'_j - y_j}{y_j} \right|
\]  

(5.19)

where \( t \) is the number of data entries in the dataset, and \( y_j \) and \( y'_j \) are the ground truth and estimated rack power for the \( j \)-th data entry, respectively.

By running different benchmarks shown in Table 5.5, we collect training data and duly update PMFs following the strategies introduced in Sec. 5.4.4. Furthermore, after each PMFs update, we run our synthetic workloads, collect rack power consumption and server component states, and calculate MRE of the power estimation with updated PMFs.
Figure 5.8: The estimated power of one rack with corresponding ground truth values: a global view (left) and a local view (right).

The results are summarized in Table 5.6 (column 2-4) and illustrated in Fig. 5.7, respectively. From the rack level results in Table 5.6, we can see that the two nonlinear PMFs slightly outperform the linear PMF. According to Fig. 5.7a, the MRE of power estimation from the three PMFs monotonically decreases with the increase of training dataset, and tends to be stable at the value strictly smaller than 5%. As a zoomed-in view, Fig. 5.7b illustrates the MRE of three PMFs when MRE $\leq 5.0\%$, in which the MRE from nonlinear PMFs decreases faster than that from the linear PMFs.

To illustrate the performance of three PMFs more clearly, their power estimations for a random rack along with the ground truth values (in 0.5 hour) are shown in Fig. 5.8. A global view in Fig. 5.8a and a local view in Fig. 5.8b are shown, from which we can see that the nonlinear PMFs provide more tight fittings for the ground truth than the linear PMFs.

Note that the power estimation shown in Fig. 5.8 is slightly lower than the ground truth values. This may cause a risk for peak power monitoring applications like rack-level power capping. To reduce the risk of the power underestimation, we suggest that the datacenter operators set the redline threshold, i.e., pre-defined peak power that triggers management actions, slightly lower than the expected one, when applying
NIPD for rack-level power monitoring. As the power estimation at rack level is relatively accurate (with MRE < 3% in our case), a small magnitude (e.g., 5%) drop of the redline threshold can much reduce the risk of power overflow.

Table 5.6: Performance of linear and nonlinear PMFs for power estimations at rack level and server level

<table>
<thead>
<tr>
<th>PMFs Type</th>
<th>Monitoring Level</th>
<th>Monitoring Level</th>
<th>Monitoring Level</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rack Level</td>
<td>Server Level</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Linear</td>
<td>Nonlinear-1</td>
<td>Nonlinear-2</td>
</tr>
<tr>
<td>Mean Relative Error (NIPD for Datacenter)</td>
<td>2.63%</td>
<td>2.29%</td>
<td>2.18%</td>
</tr>
<tr>
<td>Mean Relative Error (NIPD for Racks)</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

*The two values of MRE at the server level are corresponding to the idle power and peak power, respectively.

Figure 5.9: Disaggregating datacenter power: estimated power consumption vs. referred idle/peak power.

5.6.3 Power Monitoring at Server Level

It is hard to fully validate the accuracy of our estimation at server level, because the power consumption of individual servers in our experimental environment is hard to
be obtained. As the (blade) servers are highly integrated in the rack, e.g., fourteen 4U CPU nodes are tightly packaged in one row, it is difficult to assemble sensors/meters inside individual servers. In addition, multiple servers may share the same power supply, e.g., the fourteen 4U CPU nodes share only four power suppliers, so it is also hard to obtain server-level power outside the servers.

Even though we cannot record ground truth power consumption for individual nodes, we do have a knowledge about the idle power, peak power or working power range of each server type. We focus on the power estimation of the CPU nodes, which are dominated in our datacenter (306 out of 326 servers). Their idle power and peak power can be either estimated via the process introduced in Sec. 5.4.3 or learned by referring to the nameplate power provided by the server vendor. Then, we use the measured idle and peak power values as references to evaluate our server-level power estimation.
Power Disaggregation of Datacenter

Using the PMFs trained from the aggregated power readings of IT facilities, we estimate the real-time power consumption of individual servers. To illustrate the performance, we choose four CPU nodes (Node-25 to Node-28) among our datacenter as test nodes. Moreover, we make two of them run our peak workload (listed in Table 5.5), and the other two firstly keep idle for 15 minutes and then run peak workload for another 15 minutes.

Along with corresponding referred power bounds, the resulted power estimations for two of the CPU test nodes are demonstrated in Fig. 5.9 (the situations of the other two nodes are similar). From the results we can see that, both the estimated idle/peak power\(^4\) from the three PMFs are close to the referred power bounds, with the nonlinear PMFs slightly outperform the linear PMFs. The overall performance is summarized in Table 5.6 (the last three columns). By checking the performance under these two extreme cases, we can validate the effectiveness of our solution.

We have also observed that the estimated power values are slightly larger than the referred ones, as shown in Fig. 5.9. This is because when disaggregating the datacenter power, the power loss during the transmission (e.g., by wire and PDUs) as well as power consumed by interconnection network (e.g., network switches, line cards, and datacenter accessories) are assigned to individual servers, as discussed in Remark 4.

Power Disaggregation of Racks

When a datacenter is capable of monitoring power consumption of each rack, our NIPD can be used to disaggregate the rack-level power consumption into server-level power consumption. As the servers in a rack are usually homogeneous, we can set the number of VHCs as one, and in this case the computational complexity for training PMFs will be much lower than that in a heterogeneous environment (refer to Appx. B.3).

In our datacenter, we choose a test rack which contains 28 CPU nodes (Node-1 to Node-28) and 2 I/O index nodes. Since the number of CPU nodes is much larger than that of the I/O index nodes and the CPU nodes’ working power ranges are very similar, this rack is approximately homogeneous. The collected historical data from this rack are used for PMF training, and the updated PMF is used to make estimation

\(^4\)Peak power values refer to the power readings when the CPU utilization is 100%.
under idle/peak workloads for individual servers in this rack. The resulted idle/peak power estimation of the same test nodes in Sec. 5.6.3 is illustrated in Fig. 5.10.

According to the server level results shown in Table 5.6 and the comparison between Fig. 5.10 and Fig. 5.9, we can find that the server-level power estimation by disaggregating the rack power is better than that from disaggregating the entire datacenter power. This is because the impact of hardware components not modeled in PMFs is smaller at the rack level than at the whole datacenter level, as per the discussion in Remark 4.

As a concluding suggestion, if the datacenter operators can obtain rack-level power information, it would improve the accuracy of NIPD estimation to directly disaggregate the rack-level power than to disaggregate the power of the entire datacenter.

5.7 Conclusions

We defined the problem of non-intrusive power disaggregation (NIPD) for legacy datacenters, and developed a software-based approach for power estimation of individual servers. A non-intrusive training procedure was proposed to find the power mapping functions (PMFs) between the states of servers and their power consumption. With linear or nonlinear state feature transformation, the PMFs can represent linear or nonlinear power models. To effectively improve the precision of PMFs as well as lower the training overhead, we adopted adaptive PMFs update strategies of selective training data collection and iterative state feature elimination. Based on the updated PMFs with servers’ running state information, the power consumption of individual servers can be estimated in real-time by only referring to the aggregated power of the entire datacenter. Our solution introduced no hardware cost and incurred no interruption to the running servers. The experimental results over a 326-node datacenter showed that our solution can provide precise power estimation at both the rack level and the server level. For example, with the nonlinear PMFs including two nonlinear terms, the mean relative error of our power estimations can reach 2.18% at rack level, and 9.61% and 7.53% at server level with respect to the idle power and peak power.
Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this dissertation, by taking advantage of load curve modeling, we have investigated three critical load monitoring problems in smart power systems.

First, to deal with the load curve data cleansing problem, we developed novel model and approach, i.e., portrait data model and appliance-driven approach, for the energy supply side and demand side, respectively. The results from real-world data tests demonstrated that i) our outlier detection methods by applying portrait data model are much more effective and efficient than existing regression-based methods, and ii) our appliance-drive approach can precisely capture corrupted data in the load curve of customers.

Second, to separate the aggregated energy consumption to that of individual appliances, we proposed practical solutions for both non-intrusive and semi-intrusive load monitoring. By referring to the load curve data from one meter outside the residential house, our non-intrusive method provided a simple and universal approach to obtain individual appliances’ energy consumption; by deploying multiple low-cost energy meters, we validated the benefits for energy disaggregation under large-scale appliances situation and provided corresponding strategies to pursue the minimum meter cost.

Last but not least, we extended the load curve modeling to fine-grained power monitoring in datacenters. A purely software-based approach was developed for power estimation at server level, which introduces no hardware cost and incurs no interruption to the running servers. The experimental results over a real-world 326-node
datacenter show that our solution can provide precise power estimation at both the rack level and the server level.

6.2 Future Work

Load Curve Modeling for Commercial Buildings

Based on the building types, e.g., residential buildings, commercial buildings, industrial buildings, and transport buildings, the targets of load curve modeling can be different. In this dissertation, we focus on the load curve modeling and load monitoring of residential buildings. Nevertheless, according to [42], the commercial buildings (including office and educational buildings) consumed 35% of the total electricity consumption of U.S. in 2010, which held the second largest share of all types of buildings. Therefore, the analysis of load profile from commercial building is also important. In the future work, the load curve modeling for the commercial buildings, as well as corresponding load monitoring problems like load curve data cleansing and energy disaggregation, need further investigation.

Energy Disaggregation and Re-aggregation

Energy disaggregation is to decompose the aggregated load down to the individual (appliance) energy consumption, which can in turn give us the basic building blocks to re-aggregate different views according to various grouping criteria. For example, spatial re-aggregation recomposes devices based on where they are, and functional re-aggregation recomposes appliances based on what they are. The concept of energy re-aggregation was initially proposed and preliminarily investigated in [111, 112]. We regard it as another future research perspective as it has shown great effectiveness for energy consumption visualization and information systems for the end users.

Privacy Issue in Smart Metering

Referring to our current work and an extended research in [113], we have found that the house occupancy states (whether there is someone at home or not) can be inferred based on decoded appliances’ states. Thus, the privacy of residential customers is a big concern, which has also drawn attention in recent research [114, 115]. One of the possible solutions might be to make use of the access control of residents’ energy data. According to [116], the cloud hosting services with simple access control can
be provided, which can not only eliminate most types of privacy violations but also enable fine-grained user control over the energy consumption data. As occupancy detection is based on the analysis of energy consumption data, similar protecting approaches can be applied. Overall, as our future work, it is interesting to pursue the balance between human privacy and smart metering.
Appendix A

NP-Hardness Proof of CDIP and SSER

A.1 NP-Hardness of CDIP

A.1.1 Preparation

First, we introduce a tree structure $T = (M, N)$, which is a complete $M$-ary tree with height of $N$, i.e., every internal node has exactly $M$ children and all leaves have the same depth of $N$. By default, the height of the root is 0. Furthermore, each edge $(i, j)$ of $T$ has a non-negative cost $c(i, j)$, which will be defined later.

Assume that load data $\{y_1, y_2, \cdots, y_n\}$ is generated by $m$ appliances with the initial state vector $S_0$. Also assume that the upper bound on the total number of on-off switches within a sampling interval is $\delta(< m)$. We can build the following tree:

Step 1. Set $S_0$ as the root of the tree.

Step 2. Set the children of root as all possible states that can be transited from $S_0$, with the constraint that the total number of on-off switches is no larger than $\delta$.

Therefore, we can add $M$ children to the root, where $M = C_m^0 + C_m^1 + \cdots + C_m^\delta$;

Step 3. For each node of the tree, set its children as all possible states ($M$ states) that can be transited from it;

Step 4. Repeat Step 3) from $t = 1$ to $n$. At the end, we obtain $T = (M, N)$, where $N = n$;
Step 5. Set the cost of edge \((i, j)\) as \(c(i, j) = |v|\), where \(v\) is obtained by solving the optimization problem:

\[
\begin{align*}
\text{minimize } |v|, & \quad \text{subject to } (P_i^T S_j - v) / f \leq y_i \leq (P_u^T S_j + v) / f.
\end{align*}
\]

Thus, we can translate CDIP into the problem of finding the minimum-cost path in \(T(M, N)\) from the root to a leaf. We reduce the optimization problem to its decision version.

**Definition 21. Decision version of CDIP (d-CDIP)**: Given a constant \(k\), find out whether or not there exists a full path in \(T(M, N)\) with total cost no larger than a constant \(k\).

The d-CDIP can be re-formulated as

\[
d-\text{CDIP} = \{(T, c, k) : T = (M, N),
\]

\[
c \text{ is the cost function ,}
\]

\[
k \in \mathbb{R}^+, \text{ and}
\]

\[
T \text{ has a full path with cost } \leq k\}.
\]

We next reduce a well-known NP-complete problem, the Traveling Salesperson Problem (TSP) to d-CDIP. TSP can be formulated as

\[
TSP = \{(G, c', k) : G = (V, E) \text{ is a complete graph },
\]

\[
c' \text{ is the cost function },
\]

\[
k \in \mathbb{R}^+, \text{ and}
\]

\[
G \text{ has a Hamiltonian cycle with cost } \leq k\}.
\]

We complete the proof in two steps: firstly we show that d-CDIP is NP; then, we prove that d-CDIP is NP-hard by showing \(TSP \leq_P \text{d-CDIP, i.e., there exists a reduction from TSP to d-CDIP.}\)

**A.1.2 d-CDIP is NP**

- **Certificate**: A path of \(T\).
- **Algorithm**:
  - Check that the path is full, i.e., the path starts from the root and ends at a leaf.
– Sum up the edge costs along the path and check if it is no larger than $k$.

• *Polynomial Time:* We need $N$ steps to check the fullness of path and obtain the total cost.

### A.1.3 d-CDIP is NP-Hard

• Firstly, we develop an algorithm $F : (G, c', k) \rightarrow (T, c, k)$, i.e., $G$ and $c'$ in TSP can be transferred to $T$ and $c$ in d-CDIP as follow:

**Step 1.** Choose any node of $G$ as the root of $T$;

**Step 2.** For each leaf node of the current tree, add its children as all the other nodes of $G$. Since $G$ is a complete graph, we can add $|V| - 1$ children to each leaf node, where $|V|$ is the number of nodes in $G$.

**Step 3.** Repeat Step 2 for $|V|$ times. At the end, we build $T = (M, N)$, where $M = |V| - 1$ and $N = |V|$;

**Step 4.** Set the cost of edge $(i, j)$ in $T$, $c(i, j)$, as follows:

(a) Initialization: $c(i, j) = c'(i, j)$, where $c'(i, j)$ is the edge cost in $G$.

(b) For each edge $(i, j)$ of $T$ where $j$ is a non-leaf node, if $j$ has appeared in the path from the root (including the root) to $i$, i.e., $j$ is an ancestor of $i$ in the tree already, replace $c(i, j) = \infty$.

(c) For each edge $(i, j)$ of $T$ where $j$ is a leaf node, if $j$ is not the same as the root node, replace $c(i, j) = \infty$.

To help understand the construction of $T$ with $G$, Fig. A.1 show an example with three nodes in $G$. 

![Figure A.1: An example showing the construction of $T$ with $G$](image-url)
• Secondly, it is easy to see that $F$ takes $O(N^2)$ running time.

• Thirdly, we show that

\[ (G, c', k) \in TSP \iff (T, c, k) \in CDIP. \]

- ($\Rightarrow$)

$G$ has a Hamiltonian cycle with cost $\leq k$.

$\Rightarrow$ there exists a full path in tree $T$ with cost $\leq k$.

(Note that there will be no internal node along the path occurring more than once, otherwise the cost will be infinite with operation in Step 4.)

- ($\Leftarrow$)

$T$ has a full path with cost $\leq k$.

$\Rightarrow$ there exists a traverse instance in its corresponding graph $G$ with cost $\leq k$.

(Note that based on the tree construction procedure, only the full paths starting and ending at the same node can have a cost no larger than $k$, because other paths have a cost of infinity.)

$\Rightarrow$ so $G$ has a Hamiltonian cycle with cost $\leq k$.

With Step A.1.2 and Step A.1.3, we prove that d-CDIP is NP-complete. Since CDIP problem is no easier than d-SSER, the former is NP-hard.

### A.2 NP-Hardness of SSER

Referring to the optimization models formulated in (3.22) and (4.10), we can derive that the SSER problem is essentially similar to CDIP. Therefore, following the same routine shown in Section A.1, we can prove that SSER problem is NP-hard.
Appendix B

Equation Transformation, Complexity Analysis and Proof in NIPD

B.1 Equation Transformation

B.1.1 Transformation of Equation (5.12)

For a VHC consisting of \( m \) servers, each with \( n \) component states and \( \tilde{n} \) dilated state features, given its PMF in form of (5.7) and state vector in form of (5.4), the aggregated power consumption at time \( j \) can be expressed as:

\[
\hat{y}_j = \sum_{i=1}^{m} f(x_j^{(i)}) 
\]

\[
= f(x_j^{(1)}) + f(x_j^{(2)}) + \cdots + f(x_j^{(m)}) 
\]

\[
= w^\top (x_j^{(1)} + x_j^{(2)} + \cdots + x_j^{(m)}) 
\]

\[
= w^\top \hat{x}_j 
\]

where

\[
\hat{x}_j = x_j^{(1)} + x_j^{(2)} + \cdots + x_j^{(m)} 
\]

\[
= \left[ \sum_{i=1}^{m} x_{1j}^{(i)}, \sum_{i=1}^{m} x_{2j}^{(i)}, \cdots, \sum_{i=1}^{m} x_{\tilde{n}j}^{(i)} \right]^\top. 
\]
B.1.2 Transformation of Equation (5.13)

Assume that a datacenter consists of \( r \) VHCs and the PMF of the \( \kappa \)-th (\( 1 \leq \kappa \leq r \)) VHC is denoted in form of (5.10). Then, at an arbitrary time instant \( j \), the aggregated power consumption generated by \( r \) VHCs can be expressed as:

\[
y_j = \sum_{\kappa=1}^{r} \sum_{i=1}^{m_\kappa} f_\kappa(x_j^{(i)})
\]

(B.3a)

\[
y_j = \sum_{\kappa=1}^{r} \left\{ f_\kappa(x_j^{(1)}) + f_\kappa(x_j^{(2)}) + \cdots + f_\kappa(x_j^{(m_\kappa)}) \right\}
\]

(B.3b)

\[
y_j = \sum_{\kappa=1}^{r} \left\{ (w^{(\kappa)})^T \left( x_j^{(1)} + x_j^{(2)} + \cdots + x_j^{(m_\kappa)} \right) \right\}
\]

(B.3c)

\[
y_j = \tilde{w}^T \tilde{x}_j
\]

(B.3d)

where

\[
\tilde{x}_j = \left[ \hat{x}_j^{(1)}, \hat{x}_j^{(2)}, \cdots, \hat{x}_j^{(r)} \right]^T
\]

(B.4)

and

\[
\tilde{w} = \left[ w^{(1)}, w^{(2)}, \cdots, w^{(r)} \right]^T,
\]

(B.5)

in which \( \hat{x}_j^{(\kappa)} \) and \( w^{(\kappa)} \) are defined by (5.12) and (5.11), respectively.

B.2 Proof of Remark 7

Given that a datacenter is consist of \( r \) (\( r \geq 1 \)) VHCs, each with \( n_\kappa \) component states and \( \bar{n}_\kappa \) (\( 1 \leq \kappa \leq r \)) dilated state features, for each data entry in the training dataset in form of \( (\tilde{x}, y) \), the number of non-constant elements of \( \tilde{x} \) is \( \sum_{\kappa=1}^{r} \bar{n}_\kappa \) (referring to (5.12)). Then, for each of the elements, as the normalizing resolution is set as \( p \) and the normalized range is \([0, 1]\), the number of its possible values is \( \left\lceil \frac{1}{p} \right\rceil \). Therefore, the total number of possible combinations, i.e., the values of \( \tilde{x} \), is \( \left\lceil \frac{1}{p^1} \right\rceil + \left\lceil \frac{1}{p^2} \right\rceil + \cdots + \left\lceil \frac{1}{p^{r}} \right\rceil \), i.e., \( \sum_{\kappa=1}^{r} \left\lceil \frac{1}{p^{n_\kappa}} \right\rceil \).

B.3 PMFs Training Complexity

For PMFs training, the optimization model established in (5.15) is used to find the optimal PMFs coefficients, which essentially falls into the form of lease square linear
regression. With \( t \) data entries in the training dataset, the closed-form solution to the least square regression problem (5.15), i.e., the PMFs coefficients \( \tilde{w} \), can be expressed as:

\[
\tilde{w} = (X^T X)^{-1} X^T \hat{y},
\]

(B.6)

where \( X = [\tilde{x}_1, \tilde{x}_2, \cdots, \tilde{x}_t]^T \) and \( \hat{y} = [y_1, y_2, \cdots, y_t]^T \).

Assuming that the total number of dilated state features for all VHCs is \( \tilde{n} \), \( \tilde{n} = \sum_{\kappa=1}^{r} m_\kappa \) where \( m_\kappa \) denotes the number of component states for the \( \kappa \)-th VHC, the time complexity to get \( \tilde{w} \) from formulation (B.6) is \( O(\tilde{n}^2 \cdot t) \).

For the datacenter hosting extremely large number of servers, the complexity of PMFs training can be much reduced under the following two cases:

- If the number of servers in a virtual homogeneous cluster (VHC) is small, the PMF modeling and training process developed for the whole datacenter can be directly applied within the VHC, as long as the aggregated power of the VHC is accessible.

- If the number of servers in a VHC is large, the (dilated) state features from part of the servers (e.g., a few hundreds as in our case) should be enough to yield a relatively accurate PMF, as long as the aggregated power of these servers is accessible.
Bibliography


[66] D. L. Donoho, “For most large underdetermined systems of linear equations the minimal $l_1$ norm solution is also the sparsest solution,” *Communications on pure and applied mathematics*, vol. 59, no. 6, pp. 797–829, 2006.


composable and responsive power models for multicore processors using perfor-
mance counters,” in Proceedings of ACM International Conference on Super-


[96] M. Y. Lim, A. Porterfield, and R. Fowler, “Softpower: fine-grain power estima-

[97] P. Ranganathan, P. Leech, D. Irwin, and J. Chase, “Ensemble-level power man-

[98] W. L. Bircher and L. K. John, “Complete system power estimation: A trickle-
down approach based on performance events,” in Proceedings of Interna-


[100] J. C. McCullough, Y. Agarwal, J. Chandrashekar, S. Kuppuswamy, A. C. Sno-
eren, and R. K. Gupta, “Evaluating the effectiveness of model-based power character-


