Parallel Accelerated Optimization Techniques for Large-Scale Power System Planning and Operation

by

Shengjun Huang

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Energy Systems

Department of Electrical and Computer Engineering
University of Alberta

Abstract

Electricity is ubiquitous in modern life, nevertheless, its availability and reliability are not granted. Actually, providing reliable electricity is an enormously complex technical challenge even on the most routine of days, which requires trained and skilled operators, sophisticated computers and communications, and sufficient planning and designing. A broad set of interrelated decisions should be made for scheduling and operating various types of devices for electricity generation, transmission, and distribution subject to engineering, market, and regulatory constraints.

Due to the nonlinear and non-convex power flow equality constraints, discrete control and decision variables, and large system scales, the decision-making process is extremely daunting that demands a high level of computational intelligence and speed. In addition, developments in the power industry, such as the introduction of renewable energy and smart grid, have brought new challenges on this tough issue, including uncertainty factors and real-time responses. Therefore, most conventional off-line optimization tools are required to be merged into on-line modes. On the other hand, rapid advances in digital computers, smart meters, and communication technologies, etc., provide great opportunities and powerful tools.

In the context of coexisting challenges and opportunities, possibilities for the acceleration of various optimization techniques for large-scale power system planning and operation problems are investigated in this thesis via algorithm customization, framework development, and parallel processing. Eight fundamental problems are investigated with the coverage of all three major domains of power systems. Four categories are separated for the classification of them in this thesis.

- **Planning of transmission system.** Due to the boost of loading levels and the wide utilization of distributed generators, Transmission Expansion Planning (TEP) has regained its significance for investigation. In this thesis, a Multi-Group Particle Swarm Optimization (MGPSO) algorithm is proposed to solve DC TEP based on the multi-group co-evolution strategy and Linear Equation System (LES) transformation. Su-

periority over commercial software Lingo is established with case studies. In addition, based on the disjunctive model, Security Constrained TEP (SCTEP) problem is formulated into a Mixed-Integer Linear Programming (MILP) problem. Branch-and-Cut Benders Decomposition (BCBD) method is developed with the integration of BD into a B&C framework, resulting in better performance over MILP solver Cplex.

- **Operation of transmission system.** Alternating Current Power Flow (ACPF) analysis is one of the most fundamental tasks for the transmission system operation and optimization problems, such as Real-Time Contingency Analysis (RTCA). Exploration on the single ACPF solution has been conducted on Graphics Processing Unit (GPU) with Fast Decoupled (FD) method, where both Matlab and Compute Unified Device Architecture (CUDA) are employed for programming. Due to the limited performance, RTCA which comprises multiple ACPFs is addressed with Compensation Method (CM). Based on the sensitivity analysis of similar ACPFs, the number of matrix decomposition has been greatly reduced. Good performance on accuracy, convergence, and scalability of CM running on GPU with CUDA has been validated.

- **Operation of generation system.** Optimal operation of generation system dominates the economy and security of the whole electricity supply process. In this thesis, Security Constrained Unit Commitment (SCUC) and Real-Time Optimal Power Flow (RTOPF) are solved to determine the on-off status and the amount of active power output of each thermal generator, respectively. Potential of different Robust Optimization (RO) frameworks are fully discussed within the context of parallel computing. Minimization of the prediction error with the consideration of renewable energy is investigated for RTOPF based on the GPU parallel processing.

- **Operation of distribution system.** Due to the low voltage level of distribution network, significant power losses are encountered. In order to minimize them, two problems are investigated from different aspects: Distribution Network Reconfiguration (DNRC) and Real-Time Volt/Var Optimization (RTVVO). Two major concerns of DNRC have been addressed with the proposed efficient decimal solution encoding and decoding strategy and the acceleration of distribution network power flow (DNPF) process. In terms of RTVVO, detailed formulation of transformers and other devices is integrated into the Direct Approach (DA). Well-established data structure and thread organization pattern are also provided for GPU implementation.

Preface

The material presented in this thesis is based on original work by Shengjun Huang. As detailed in the following, material from some chapters of this thesis has been published in conference proceedings and as journal articles under the supervision of Dr. Venkata Dinavahi in concept formation and by providing comments and corrections to the article manuscript.

Chapter 2 includes the results from the following papers:

- S. Huang, V. Dinavahi, "Multi-group particle swarm optimization for transmission expansion planning solution based on LU decomposition", *IET Gener. Transm. Distrib.*, vol. 11, no. 6, pp. 1434–1442, May 2017.
- S. Huang, V. Dinavahi, "Security constrained transmission expansion planning by accelerated benders decomposition", *Proc. North Amer. Power Symp.*, Denver, CO, USA, Sept. 2016, pp. 1–6.
- S. Huang, V. Dinavahi, "A branch-and-cut Benders decomposition algorithm for transmission expansion planning", *accepted in IEEE Syst. J.*, pp. 1–11, Nov. 2017.

Chapter 3 includes the results from the following papers:

- S. Huang, V. Dinavahi, "Performance analysis of GPU-accelerated fast decoupled power flow using direct linear solver", *Proc. IEEE Electr. Power Energy Conf.*, Saskatoon, Canada, Oct. 2017, pp. 1-6.
- S. Huang, V. Dinavahi, "Real-time contingency analysis on massively parallel architectures with compensation method, *Under Revision in IEEE Trans. Power Syst.*, pp. 1-8, Aug. 2017, (TPWRS-01226-2017).

Chapter 4 includes the results from the following papers:

- S. Huang, V. Dinavahi, "A comparison of implicit and explicit methods for contingency constrained unit commitment", *Proc. North Amer. Power Symp.*, Morgantown, WV, USA, Sept. 2017, pp. 1–6.
- S. Huang, V. Dinavahi, "Fast batched solution for real-time optimal power flow with penetration of renewable energy", *Accepted in IEEE Access*, pp. 1–13, Feb. 2018.

Chapter 5 includes the results from the following papers:

- S. Huang, V. Dinavahi, "Fast distribution network reconfiguration with graph theory", *Under Revision in IET Gener. Transm. Distrib.*, pp. 1-9, Sept. 2017, (GTD-2017-1488).
- S. Huang, V. Dinavahi, "GPU-based parallel real-time volt/var optimization for distribution network considering distributed generators", *Under Revision in IET Gener. Transm. Distrib.*, pp. 1-10, Nov. 2017, (GTD-SI-2017-1887).

To my wife, Shufang Wang
who is always a constant source of support and encouragement
and to my parents for their unconditional love
and to the universe.

千万人中万幸得以相逢
刹那间澈净明通
成为我所向披靡的勇气和惶恐

Acknowledgements

I would like to express my deepest appreciation to *Dr. Venkata Dinavahi*, who is the supervisor for my Ph.D. program at the University of Alberta, for his enlightening guidance, continuous support, and enthusiastic encouragement. It was my great fortune and privilege to work with *Dr. Dinavahi*. He has been and will always be the role model for me.

I thank my colleagues and friends at the RTX-Lab with whom I had a wonderful time during my Ph.D. program.

I would like to acknowledge *Dr. Bo Guo* and *Dr. Tao Zhang* from the National University of Defense Technology, who paved the way for my research at the very beginning. Special thanks should be granted to *Ms. Xiaoping Yin* and *Dr. Dong Wang*, without their solely encouragement and sacrifice, I could not achieve my dream to study abroad.

Many thanks to everyone I met, including but not limited to: *Dr. Yabing Zha*, *Dr. Yajie Liu*, *Dr. Rui Wang*, *Dr. Hongtao Lei*, *Dr. Xiaoyuan Liu*, *Dr. Yan Zhang*, *Ms. Sujuan Wang*, *Dr. Mohammed Hussain*, *Mr. Junhong Li*, *Mr. Lu Liu*, *Mr. Minghui Huang*, *Mr. Yunhe Zhou*, *Ms. Yining Fang*, *Mr. Guoxin Li*, *Mr. Chuang Xu*, *Dr. Xingcheng Hu*, *Mr. Fudong Li*, *Dr. Long Yang*, *Ms. Yue Wang*, *Mr. Zhuoxuan Shen*, *Mr. Peng Liu*, *Mr. Zhichao Shi*, *Mr. Huangke Chen*, and *Mr. Jingliang Li*, etc. Specifically, three roommates are greatly appreciated: *Mr. Min Tang*, *Mr. Ziling Wei*, and *Mr. Keyu Wu*.

I would like to thank my wife *Ms. Shufang Wang* for every single sight she gave to me. Expectations and concerns from my grandparents, parents, and the whole big family will be always remembered and cherished.

This thesis work was supported by the China Scholarship Council under Grant No. 201403170337. I greatly appreciate the financial support.

Table of Contents

List of Tables

List of Figures

List of Acronyms

AC	Alternating Current
BCBD	Branch-and-Cut Benders Decomposition algorithm
BCBV	Branch-Current to Bus-Voltage matrix
BD	Benders Decomposition
BIBC	Bus-Injection to Branch-Current matrix
BRD	Branch-based Reachability Detection
CCG	Column-and-Constraint Generation algorithm
CM	Compensation Method
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DC	Direct Current
DFS	Depth-First Search
DG	Distributed Generator
DNPF	Distribution Network Power Flow
DNRC	Distribution Network Reconfiguration
FDPF	Fast Decoupled Power Flow
GPU	Graphics Processing Unit
GS	Gauss-Seidel
HPC	High-Performance Computing
LES	Linear Equation System
MGPSO	Multi-Group Particle Swarm Optimization
MILP	Mixed-Integer Linear Programming
MINLP	Mixed-Integer Non-Linear Programming
MRD	Matrix-based Reachability Detection
MST	Minimum Spanning Tree
NR	Newton-Raphson
OLTC	On-Load Tap Changer
PDIPM	Primal-Dual Interior Point Method
PLD	Probability-based Loop Destruction
PSO	Particle Swarm Optimization
RO	Robust Optimization
RTCA	Real-Time Contingency Analysis
RTOPF	Real-Time Optimal Power Flow
RTS	Radial Tree Structure
RTVVO	Real-Time Volt/Var Optimization
SC	Switched Capacitor
SCTEP	Security Constrained Transmission Expansion Planning
SCUC	Security Constrained Unit Commitment

Nomenclature

Sets

$\mathcal{C}$	Set of candidate transmission lines.
$\mathcal{E}$	Set of existing transmission lines.
$\mathcal{G}_i$	Set of indices of the generating units at bus i.
$\mathcal{N}_b$	Set of buses.
$\mathcal{N}_g$	Set of thermal generators.
$\mathcal{N}_l$	Set of transmission lines.
$\mathcal{N}_r$	Set of renewable generators.
$\mathcal{S}$	Set of security contingencies/scenarios.
$\mathcal{T}$	Set of indices of the time periods.

Parameters

a_g, b_g, c_g	Coefficients of the quadratic cost function of generator g.
A_g, B_g	Parameters for piecewise linearizion of quadratic cost function.
c_{ij}	Construction cost for transmission line ij.
C_g^D, C_g^U	Shutdown/Startup cost of generator g.
d_i	Load demand at node i.
$\boldsymbol{d}$	Vector of load demand for each bus.
$D_i(t)$	Load demand at bus i in period t.
$\bar{f}_{ij}$	Maximum power flow on transmission line ij.
$\bar{g}_i$	Maximum amount of generation at node i.
$\bar{\boldsymbol{g}}$	Vector of maximum power generation for each generator.
G_{ij}, B_{ij}	Transfer conductance and susceptance between buses i and j.
K	Maximum number of circuits can be built for each line.
K^G, K^L	Number of generators and transmission lines under a contingency criterion.
M	Large constant number used for linearization or penalty.
m	Particle swarm optimization algorithm population size.
n_b	Total number of buses.

n_c	Total number of transmission lines.
n_g	Total number of generators.
n_t	Total number of unimproved iterations before termination.
n_{ij}^0	Initial number of circuits on corridor ij.
$n_{ij}^{0(s)}$	Initial number of circuits on corridor ij for scenario s.
P	Penalty factor for loss of load.
P_g^{G0}	Active power output of generator g in the previous subinterval.
$\bar{P}_g$	Maximum power output of generator g.
$\underline{P}_g$	Minimum power output of generator g.
$R(t)$	Spinning reserve requirement in period t.
R_g^D, R_g^U	Ramp-down/Ramp-up rate limit for generator g.
$\boldsymbol{S}$	The bus-branch incidence matrix.
T_g^D, T_g^U	Minimum down/up time for generator g.
$\bar{\theta}_i$	Maximum value of the phase angle at bus i.
γ_{ij}	Susceptance of transmission line ij.
$z_g^{(s)}$, $z_{ij}^{(s)}$	Binary parameter: 0 if generator g (transmission line ij) is under contingency at scenario s; 1 otherwise.
α	Weight parameter for the objective function.
ϵ	Small value regraded terminate condition.
γ_{ij}	Susceptance of transmission line ij.
^{down}, ^{up}	Ramp down and up limits of thermal generator.
^{min}, ^{max}	Lower and upper limits of specified variables.

Decision Variables

$c_g^d(t)$	Shutdown cost of generator g in period t.
$c_g^{p(s)}(t)$	Production cost of generator g in period t on scenario s.
$c_g^u(t)$	Startup cost of generator g in period t.
f_{ij}	Power flow on line ij.
$f_{ij}^{0(s)}$	Power flow of the original circuit ij at scenario s.
$f_{ij}^{k(s)}$	Power flow of the kth parallel circuit ij at scenario s.
$f_{ij}^{(s)}(t)$	Power flow on transmission line ij in period t on scenario s.
$\boldsymbol{f}$	Vector of power flow for each corridor.
$g_i^{(s)}$	Amount of generation at node i at scenario s.
$\boldsymbol{g}$	Vector of power generation for each generator.
n_{ij}	Number of candidate circuit to be built for corridor $i-j$.

n_{ij}^k	Binary variables indicating whether the kth parallel circuit of corridor ij is built.
P_g^G, Q_g^G	Active and reactive power output of generator g.
P_r^R, Q_r^R	Active and reactive power output of REG r.
P_i^D, Q_i^D	Active and reactive power demand of bus i.
$p_g^{(s)}(t)$	Power output of generator g in period t on scenario s.
Q^w	The worst system operation cost.
r_k	Loss of load for bus k.
$r_i^{(s)}$	Loss of load for bus i at scenario s.
$r_i^{(s)}(t)$	Loss of load for bus i in period t on scenario s.
$\boldsymbol{r}$	Vector of load curtailment for each bus.
$v_g(t)$	Binary decision variable: 1 if generator g is online in period t; 0 otherwise.
V_i, V_j	Voltages magnitude at node bus i and j.
z_g, z_{ij}	Binary decision variables have the same meaning with $z_g^{(s)}$ and $z_{ij}^{(s)}$.
θ_{ij}	Voltage angle difference between buses i and j.
θ_i	Voltage angle at node bus i.
$\theta_i^{(s)}$	Voltage angle of node i at scenario s.
$\theta_i^{(s)}(t)$	Phase angle at bus i in period t on scenario s.
$\beta, \tau, \delta, \eta,$	Dual variables corresponding to the economic dispatch constraints.
ζ, ξ, ρ	Dual variables corresponding to the economic dispatch constraints.

1 Introduction

Interdisciplinary backgrounds of power systems, operations research, and computer engineering lead to the main theme of this thesis: accelerating various optimization techniques for the fast solution of large-scale power system planning and operation problems via algorithm customization, framework development, and parallel processing.

1.1 Backgrounds

Electricity is one of the most important discoveries that is deeply involved in modern life. From the moment we open our eyes in the morning to the time we go to bed in the evening, electricity is consumed by all devices around us, including light, refrigerator, air conditioner, washing machine, computer, television, and mobile phone, etc. In addition to domestic applications, more extensive utilizations are attributed to industrial production and public service, such as food, water, clothing, transportation, communication, hospital, bank, and school, etc. We cannot imagine what the world will turn out to be without electricity.

Different from other types of energy, electricity flows at a speed close to the speed of light and cannot be stored in large amount for industrial application. Therefore, it is generated the instant that it is consumed. Failure to preserve this equilibrium will result in power shortage or even outage. In order to achieve the stable and efficient electricity supply for the whole society, the electric power system is built and maintained as a basic national infrastructure. Generally, it covers wide geographical regions and includes tens of thousands of components. Based on the functionality, a power system can be broadly divided into: the generation system that supplies the power; the transmission system that

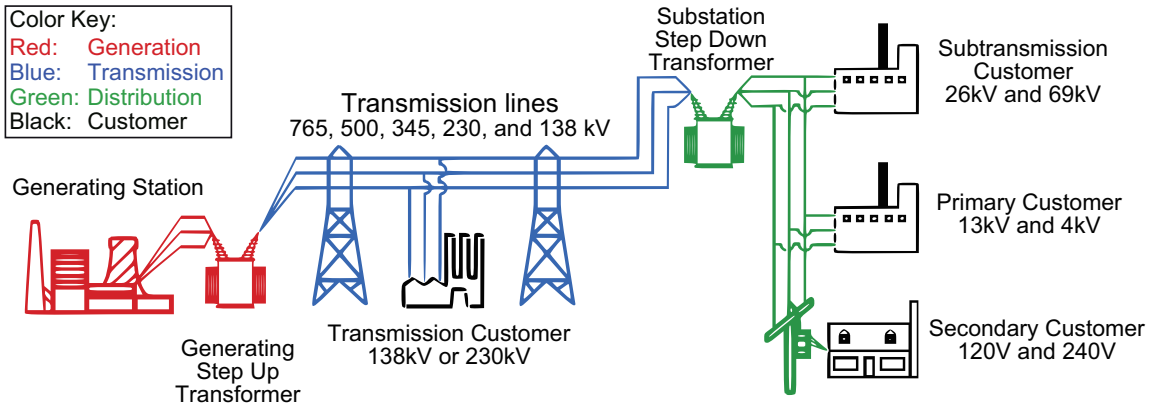


Figure 1.1: Basic structure of the electric power system [1].

carries the power from generators to the load centers; and the distribution system that feeds the power to nearby homes and industries. The basic structure is illustrated in Fig. 1.1. At the beginning, lower voltage (10 to 25 kV) electricity is produced by generators driven with various types of energy sources, such as coal, oil, natural gas, hydropower, nuclear, wind, photovoltaic, etc. Then, it is stepped up to a higher voltage (138 to 765 kV) to reduce losses for the long distance shipping over transmission lines. Finally, stepping down is performed in the distribution center for the service of industrial (4 to 69 kV) and residential (120 to 240 V) customers.

Providing reliable electricity is an enormously complex technical challenge even on the most routine of days, which requires trained and skilled operators, sophisticated computers and communications, and sufficient planning and designing [3]. A broad set of interrelated decisions should be made for planning and operating various types of devices for electricity generation, transmission, and distribution subject to engineering, market, and regulatory constraints. Due to nonlinear and non-convex power flow equality constraints, discrete control and decision variables, and large system sizes, the decision-making process is usually daunting that demands a high level of computational intelligence and speed. In addition, developments in the power industry, such as the introduction of renewable energy and smart grid, have brought new challenges on this tough issue, including uncertainty factors and real-time response. Therefore, most conventional off-line optimization tools are required to be merged into on-line operations. On the other hand, advanced progress is also achieved on digital computers, smart meters, and communication technologies, etc., which provides great opportunities and powerful weapons. Therefore, in this thesis, we intend to investigate the acceleration possibility of advanced optimization techniques based on the development of new devices and technologies, for the purpose of addressing new challenges during the solution of large-scale power system planning and operation problems.

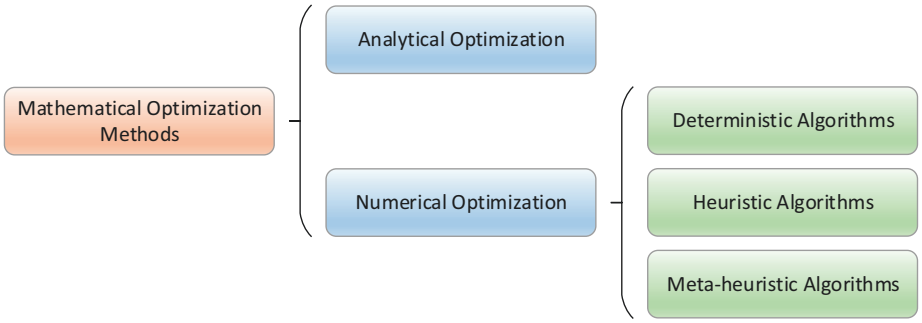


Figure 1.2: A schematic classification of mathematical optimization methods.

Fig. 1.2 illustrates a schematic classification of mathematical optimization methods utilized in the solution of engineering problems. For the majority of practical applications with strong nonlinear property and high dimensionality, the derivation of an analytical solution is convoluted or even intractable. Therefore, numerical optimization methods, which can be classified into deterministic algorithms, heuristic algorithms, and meta-heuristic algorithms, gain more popularity than analytical optimization methods.

Deterministic algorithms are based on rigorous mechanism and free of randomness, such that they will converge into global optimal solution in a finite number of iterations, and the optimality could be indicated by the lower and upper bounds at each iteration. Examples include Benders Decomposition (BD) [4], Robust Optimization (RO) [5], and Primal-Dual Interior Point Method (PDIPM) [6], etc. Simplification might be required to utilize the deterministic methods, such as linearization and continuous relaxation. Heuristic algorithms utilize problem dependent rules or the current information gathered by the algorithm to decide the next solution steps. These types of methods can achieve global optimal or suboptimal solution in a short time with limited computational effort, but the performance is problem dependent and would be poor for large-scale systems. Meta-heuristic algorithms are robust, with the capability of solving very general problems regardless of their size and whether they are convex or non-convex, continuous or discrete; however, the computation is more intensive than heuristic algorithms, and the optimality of the final solution cannot be validated by themselves. They consider the target problem as block-box without deep insight into inner structures. A series of candidates consists of the genetic algorithm, simulated annealing, and Particle Swarm Optimization (PSO), etc. Acceleration possibilities on the deterministic and meta-heuristic algorithms will be explored in this thesis due to their preferable properties, i.e., customization and acceleration strategies on the solution methodology can be easily extended to other applications.

Enhancement on the solution efficiency to fulfill on-line and real-time optimization can be achieved through algorithm refining and parallel processing. The former relates to solid

backgrounds on specified algorithm derivation, thus the detailed discussion is devoted to subsequent chapters. In terms of parallel processing, exacting full computing power from modern hardware architectures is not trivial, which requires the knowledge and efforts from both the application and the computer architecture domain [7], including but not limited to algorithm-level optimization, data structure optimization, special hardware instructions, and parallel programming, etc. Since most engineers have limited access to supercomputers and large-scale Central Processing Unit (CPU) clusters, the multi-core CPU and many-core Graphics Processing Unit (GPU) received more attention. In order to address various general tasks, the cores in CPU are powerful and highly optimized for complex operations. Nevertheless, the number is limited and the latency for launching is high. In addition, the performance is directly dependent on the clock speed, which is away from the expectation of Moore's law and experiencing a reducing upward trend due to thermodynamic concerns, i.e., the limitation is reaching. On the other hand, thousands of smaller and lighter cores are integrated into GPU to handle millions of threads simultaneously. Initially designed for graphics and video rendering, GPU has been introduced in the High-Performance Computing (HPC) community with the release of NVIDIA® Compute Unified Device Architecture (CUDA) [8], a C-based Application Programming Interface (API) for GPU parallel programming. Based on CUDA, the extremely high floating point performance of GPU can be harnessed for general scientific computation on inexpensive accessible desktop computers.

1.2 Problem Definition and Scope

In this section, the target problems addressed in this thesis are defined to clearly identify the scope of this thesis. In order to cover all three parts of the power system, i.e., generation, transmission, and distribution, several problems are determined for investigation as follows.

1.2.1 Security Constrained Transmission Expansion Planning

Continued increase in loading levels and generation capabilities have underscored the necessity of building new transmission lines to relieve the critically burdened power system, which comprises the Transmission Expansion Planning (TEP) problem. Proposed in 1970 [9], emphasis is originally put on the economic profits, i.e., minimizing the total investigation and operation cost. Reliability and environmental concerns are added subsequently during the radial evolution of power systems, evolving TEP into Security Constrained TEP (SCTEP) [10, 11]. SCTEP determines how to expand and reinforce the transmission network in order to supply electricity to consumers in a secure and economic

fashion, where the forecast load growth over a specific time span and the available generation assets are known as inputs for decision-making. SCTEP will continue to assume great importance in the current context of smart grid [11], where large amounts of distributed generators collecting renewable energy need to be integrated, bringing in a lot of variation, dynamics, and uncertainty.

Security, in a deterministic sense (which is the current common industry practice), is the capability of a power system to withstand a specified set of credible contingencies [10, 12]. Modeling security drastically increases the complexity of the resulting problem since the unavailability of system components needs to be characterized. One of the most extensively adopted criteria in the literature on SCTEP is the $N-1$ security criterion, which states that the system should be expanded in such a way that if a line is withdrawn the resulting system should still operate adequately [13, 14].

Two main types of models have been widely used to represent the transmission network in TEP studies: the DC and AC power flow models. The classical DC model is, in general, a mixed-integer, nonlinear, non-convex, and NP-hard problem, which presents a major challenge to all known optimization approaches, ranging from analytical method to heuristic and meta-heuristic algorithms [15, 16]. Another widely utilized DC model is the disjunctive model, which is derived from the classical DC model by representing the integer decision variables with binary decision variables, and eliminating the nonlinear property by the introduction of linearizing constant (big-M), resulting in a Mixed-Integer Linear Programming (MILP) problem. Since the MILP can be addressed with various optimized off-the-shelf solvers, the disjunctive model is mainly employed for the solution of SCTEP [10]. Full Alternating Current (AC) model is considered only at a later stage of the planning process when the most attractive topologies have been determined [17].

1.2.2 Real-Time Contingency Analysis

In order to achieve the steady-state security of power system under the outage of components, a lot of contingency scenarios are generated based on the state estimator. Contingency analysis (CA) is developed to investigate these scenarios with the solution of the corresponding AC power flows within a specified time span. CA is paramount for modern power systems as it forms the basis for important operator actions that help to improve system reliability, optimize generator dispatch, manage disparate resources, prevent cascading outages, and enhance market operations [18].

The number of possible contingency scenarios for $N-k$ security criteria is $\sum_{n=1}^{k} \frac{N!}{n!(N-n)!}$, which holds an exponential relationship with N and k. During the last few decades, the

system scale has expanded and more stringent criteria have been proposed, i.e., both N and k are increased, which introduces great challenges for CA, especially in the context of smart grid, where decisions for fast reaction should be made in real-time. In order to address the Real-Time CA (RTCA), both algorithm and hardware can be resorted since the parameters for different scenarios are similar with only a few variations.

1.2.3 Security Constrained Unit Commitment

The electricity supply and demand equilibrium is dynamically maintained within milliseconds, but the generators cannot produce power immediately when it is started, thus they should be planned in advance to deal with component outages and load variations. The Unit Commitment (UC) problem is proposed to determine the start-up and shutdown schedules of thermal units to satisfy the forecast demand over certain time periods (24h to 1 week) [19]. It is a combinatorial optimization problem with the objective of minimizing operation cost and subjects to a lot of constraints.

Reliability and security are crucial for modern power systems. Unexpected outages of power grid components can result in dramatic electricity shortages or even large-scale blackouts, therefore, Security Constrained UC (SCUC) is developed. For the sake of computational tractability, the set of credible contingencies is of reduced size, being typically associated with the well-known $N-1$ and $N-2$ security criteria. However, the more generalized $N-K^G-K^L$ contingency criterion is always desirable for practical applications [20], which furnishes the system (N components) with the capability of surviving from the sudden unavailability of K^G generation units and K^L transmission lines.

1.2.4 Real-Time Optimal Power Flow

Introduced in 1962 [21], the Optimal Power Flow (OPF) problem intends to achieve various objectives, such as total generation cost, system loss, bus voltage deviation, emission of generating units, number of control actions, and load shedding, etc., via the optimization of operation parameters of components in the power system [19]. Accordingly, constraints related to power flow equations, system security, and equipment operating limits should be satisfied.

In the context of smart grid, the high penetration of intermittent renewable energy generators is remarkable, resulting in the conventional power grid monitoring and analysis tools no longer sufficient on the solution capability and efficiency. Therefore, Real-Time OPF (RTOPF) is proposed to merge the off-line decision-making to on-line operation [22, 23]. Given the large uncertainty, nonlinear AC power flow, limited reaction time,

and strict security criterion, the demand for an efficient and generally applicable computational analysis framework is of critical importance.

1.2.5 Distribution Network Reconfiguration

As the most extensive part of a power system, the distribution network is the last step to deliver the electric power from generator to consumer. At the planning stage, it is designed as interconnected with switches, while during operation it is arranged as radial tree configuration [19]. Due to the low voltage levels, significant power loss is encountered. Therefore, the Distribution Network Reconfiguration (DNRC) problem is developed to determine the open/close status of switches such that the active power loss is minimized. Other goals are also achievable in DNRC, such as high reliability [24] and smooth voltage profile [25].

During the solution of DNRC, two main concerns need to be addressed: Radial Tree Structure (RTS) and Distribution Network Power Flow (DNPF). In order to maintain the RTS, the number of lines must be smaller than the number of buses by one unit according to the graph theory [26]. On the other hand, due to the wide range of resistance R and reactance X in the distribution network, a big ratio of R/X may appear, resulting the efficient Fast Decoupled Power Flow (FDPF) algorithm fail to converge since the major assumption $R \ll X$ is violated [19]. In addition, the Gauss-Seidel (GS) or Newton-Raphson (NR) methods are too complicated and time-consuming to tackle the DNPF due to the RTS [27].

1.2.6 Real-Time Volt/Var Optimization

One of the major responsibilities of the distribution network is the voltage and reactive power (var) management, i.e., achieving high efficiency, reliability, and quality on the power supply. A lot of control devices are available to fulfill that goal, such as On-Load Tap Changer (OLTC) transformer, Voltage Regulator (VR), and Switched Capacitor (SC), etc. In terms of how to determine/adjust the operating parameters of these facilities, the Volt/Var Optimization (VVO) is proposed [28–30]: minimizing system active power losses while satisfying equality constraints to node active and reactive power balances, as well as lower/upper bounds of node voltages.

Similar to the unit commitment problem in the transmission network, VVO is usually performed on a day-ahead time span based on the forecast demand [28], which is marked as DAVVO. Although DAVVO has been thoroughly investigated [28–39], the capability of this type of decision process in responding to fast variations is limited. In the last decades, Distributed Generators (DGs), including fuel cell, photovoltaic cell, wind turbine, and

micro-turbine, etc., have been widely integrated into distribution networks, which is of great significance for the reduction of transmission loss as well as carbon emission. On the other hand, the high penetration of DG also brings large fluctuations, resulting in the fact that the DAVVO solution is non-optimal or even infeasible. Therefore, Real-Time VVO (RTVVO) has been proposed [40–43] to address these issues.

The popularity of the utilization of new technologies, such as Advanced Metering Infrastructure (AMI), keeps increasing in the context of smart grid. Two-way communication can be provided by the AMI system to collect the information/data from smart meters and distribute the instruction/command to devices at the same time, i.e., RTVVO is technically possible from the perspective of communication. Nevertheless, in terms of fast computation for real-time decision-making, the technique is far from mature.

1.3 Literature Review

Various target problems, as well as the related optimization methodologies and solution frameworks, are reviewed as follows.

1.3.1 SCTEP with Benders Decomposition and Particle Swarm Optimization

In the literature, SCTEP is usually formulated as a MILP problem based on the disjunctive model. To tackle this tough MILP problem, a lot of proposals are developed, such as the direct MILP solution method [44] and the adjustable robust optimization approach [10]. However, both methods formulate all the scenarios/contingencies into one whole MILP problem, leading to the number of decision variables and constraints increases to a scale of thousands or even millions, which presents a major challenge for several popular commercial MILP solvers, such as Cplex and Lingo. Therefore, two types of strategies are commonly utilized:

- **Meta-heuristic Methods:** Due to the straightforward implementation scheme and global convergence capability, meta-heuristic methods are widely utilized in the solution of SCTEP. However, the execution time remains a bottleneck for meta-heuristic algorithms when dealing with large-scale TEP problems due to the long time consumed by thousands of times of LP solution (which has been reported by [45] that it may take up to 90% of the total elapsed time). Therefore, for the purpose of improving the performance, two alternatives have been emphasized in the literature: (i) an efficient LP solver [46], and (ii) a better metaheuristic algorithm [47]. To fill this

gap, an efficient LP solution process was proposed by [45], where the LP subproblem was transformed into another equivalent LP problem with reduced numbers of constraints and variables, resulting in less solution time. On the other hand, as a popular meta-heuristic method, PSO gained widespread application in the solution of TEP. Many successful results have been reported in the solution of TEP with classical PSO and its variants, of which the most popular enhancement strategies include high diversity initialization [48], replication and selection [49], mutation [49–51], and evolutionary adaptation strategy [52]. In addition, four types of PSO are discussed in [53] as a survey, and a multi-objective PSO has also been proposed in [54].

- **Decomposition Methods:** This type of method intends to alleviate the computation by separating the complex full problem into several smaller subproblems, among which BD has received the most attention in the literature. One of the pioneering works that introduce BD into TEP solving is [55], where a minimum value of the disjunctive parameter is derived to eliminate the numerical difficulties when utilizing the disjunctive model; however, the security constraints are not considered. Since then, several investigations have sought to improve the performance of BD from two aspects: master problem and subproblem. Generally, the master problem of SCTEP is MILP, which is the most time-consuming part for large-scale problems since the solution techniques are not as efficient as these for LP, thus several enhancements have been investigated to relieve the complexity and solution space of MILP [56–58]. On the other hand, efforts have also been poured into generating more effective cuts from subproblem [57–59], to eliminate regions that contain suboptimal or non-improving solutions [60], thus reducing the total number of iterations required for the whole algorithm. Specifically, [56] introduced a local search procedure to solve the master problem, where considerably decreased computational time is reported when compared with classical BD; however, additional constraints are applied to reduce the search space at the time of problem formulation, i.e., the original problem has been revised. [57] introduced a lot of interesting improvements employed by a practical project using textual description, such as inexact master problem resolution mechanism, semi-relaxed cuts for discrete decision variables, and the combination of mono-cut and multi-cut, etc.; nevertheless, few technical and mathematical details were exhibited. [58] introduced two valid inequalities to reduce the search space of the master problem, as well as multiple generation cuts and strong high-density cut to boost the convergent efficiency. [59] proposed a set of appropriate Benders cuts specifically tailored for the binary decision variables, and the effectiveness of standard and modified disjunctive model has also been studied.

1.3.2 Parallel RTCA with Compensation Method

Conventionally, two directions have been exploited to alleviate the difficult compromise between a large number of contingency scenarios and the limited solution time:

- **Robust selection procedure:** Since the evaluation of all scenarios is impractical, a subset is usually generated for analysis according to appropriate selection rules, such as the performance index contingency ranking method [61]. The size of the subset has considerable impact on the solution process of RTCA: if it is very large (conservative principle), the subsequent evaluation burden would be heavy; if it is relatively small (progressive principle), the critical scenarios might be skipped.

- **Fast evaluation methodology:** The cardinality of the contingency set can still be very large for practical systems even if progressive selection strategy was utilized, thus HPC architecture is resorted for acceleration. Investigations on RTCA with HPC were implemented on the multi-core CPU architecture [62–68], such as shared memory computers, distributed systems, and CPU clusters. Although the performance reported is acceptable, their application scope is limited due to demands on specific hardware or large computer infrastructure. On the other hand, the many-core GPU architecture is accessible for the common researcher with PC or workstation.

On the GPU platform, both the Direct Current Power Flow (DCPF) and Alternating Current Power Flow (ACPF) have been implemented to formulate the RTCA by [69–71] and [72–75] respectively. The DCPF is a linear simplification of real systems, with great advantages on the computation efficiency and convergence; however, the accuracy and capability of the solution is insufficient, e.g., inability to check voltage limit violations. On the contrary, the nonlinear ACPF is more accurate but complicated. In the literature, several sophisticated methods have been implemented on GPU to address ACPF-based RTCA, such as the Newton-Raphson (NR) method [72–77] and the Fast Decoupled (FD) method [72,78].

Compared with NR, the FD is algorithmically more efficient due to smaller dimension of Jacobian matrix and fewer times of factorization [61]. Although superiority has been granted to the FD by theoretical analysis, improvement and enhancement on it for the solution of RTCA during practical application are still necessary. The process of FD for single RTCA problem can be concluded as the solution of a series of $\boldsymbol{A}\boldsymbol{x}_i = \boldsymbol{b}_i$, for which the direct method [71,72,74,75] seems to be more attractive than the iterative method [69] since $\boldsymbol{A}$ is fixed. Taking the LU decomposition as an example, $\boldsymbol{A}$ is factorized into $\boldsymbol{L}$ and

$\boldsymbol{U}$ matrices, and $\boldsymbol{y}_i$ is generated from $\boldsymbol{L}\boldsymbol{y}_i = \boldsymbol{b}_i$ by forward substitution, then $\boldsymbol{x}_i$ is deduced from $\boldsymbol{U}\boldsymbol{x}_i = \boldsymbol{y}_i$ by backward substitution. During the whole process, the factorization takes the majority of the elapsed time even though it is executed only once while Forward and Backward (F/B) substitutions are performed several times. In order to accelerate the solution, efforts have been put forward on the sparse LU decomposition, such as reducing the number of fill-ins [79].

Nevertheless, in reality, RTCA comes with large numbers of scenarios, which can be represented as $\boldsymbol{A}_k\boldsymbol{x}_i = \boldsymbol{b}_i$ (k distinguishes different scenarios). The solution methodology depicted above for single-scenario RTCA can be easily expanded into multi-scenario RTCA since each scenario is spontaneously independent. According to the classical FD method, each $\boldsymbol{A}_k$ should be factorized to perform the F/B substitutions. In order to boost the efficiency to a higher extent, the concept of Compensation Method (CM) is proposed [61, 80], which factorizes the matrix $\boldsymbol{A}$ of base case for only once, and deduces compensation factors for each scenario during the F/B substitutions, thus the time and effort corresponding to LU decomposition for all the other scenarios can be saved. In addition to the superb solution efficiency, the CM processes satisfactory accuracy. Theoretically, the CM is integrated within the framework of FD, and FD derives from the same mathematical formulation with NR, therefore, the CM, FD, and NR should generate the same accuracy of results if the same convergence criterion is adopted. Different from the FD and NR with high popularity, the CM has not been reported with GPU architecture.

1.3.3 SCUC with Robust Optimization Framework

Under the criteria of $N - K^G - K^L$, the breakdown of each component is a random probability event. Conventionally, reserve adjustment method [81] is widely utilized to deal with outage due to its easy application [82], however, it is usually criticized for economic inefficiency or even inadequacy. Stochastic programming technique is authoritative in the formulation and solution of problems with uncertainty, which assures that the optimal solution is feasible for all (or almost all) of the possible realizations. Nevertheless, how to identify an accurate probability distribution of different types of uncertainties remains a great challenge for practical application. Instead of the hard-to-obtain probability distribution function, only the bounds of uncertainty are required for the Robust Optimization (RO) [5, 20, 83–86] approach. The obtained solution is global optimal and feasible for the worst case of uncertainty set, as well as all the other possibilities.

Commonly, the RO approach might lead to a higher cost (the "price of robustness") since it protects against the most severe event regardless of its low probability. Nevertheless, it still gained close attention in the solution of SCUC since the reliability has a con-

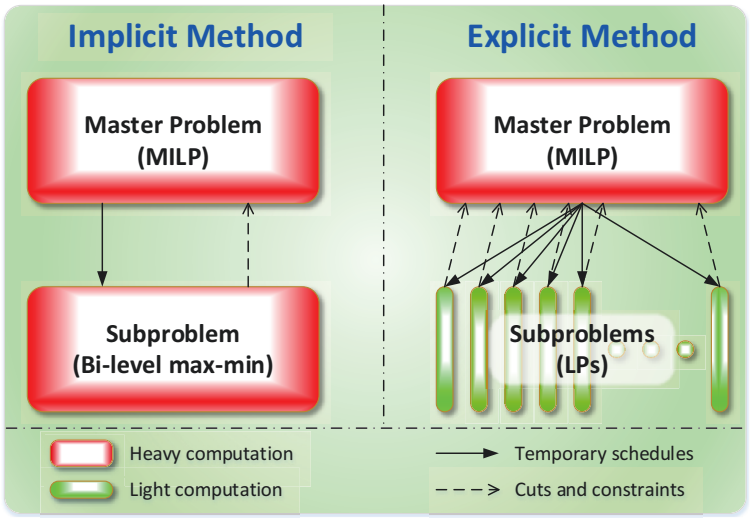


Figure 1.3: Implicit and explicit implementation frameworks for robust optimization method.

servatism nature, i.e., the final solution should withstand the worst circumstance. SCUC was first solved by [84] with RO under $n-K$ contingency criteria, however, neither the transmission capacity constraints nor the transmission line contingencies were considered. This work was extended by [20] with full consideration of transmission constraints and contingencies, but a heuristic was introduced to reduce the solution time, which made the optimality of the final solution difficult to identify. In [85], both generation unit and transmission line contingencies were investigated; nevertheless, this study was restricted on single-period SCUC (one-time decision rather than the 24-hours successive decision), i.e., time-coupling constraints were omitted. Reference [86] enhanced this work from the consideration of day-ahead schedule.

RO employs a two-stage decomposition framework for the solution of SCUC, which divides the whole problem into two stages or problems as shown in Fig. 1.3. The master problem aims to minimize the total costs while satisfying the constraints from pre-contingency and post-contingency (cuts or constraint sets). At each iteration, the master problem is solved first to generate an intermediate solution (unit commitment schedule), which is then validated by subproblems to find the most violated scenario from all realizations of uncertainty set. If there exists a violated contingency scenario, one or more cuts will be generated and included in the master problem, and it goes to the next iteration; otherwise, the solution process is terminated since all realizations are satisfied. In terms of how to identify the worst case in the second stage, two schemes are illustrated in Fig. 1.3.

- **Implicit Method:** To distinguish the most serious situation, the implicit method resorts to the bi-level $max-min$ programming. The $maxmin$ problem in the solution

approach is NP-hard, and is usually converted into a bilinear $maxmax$ problem according to the strong duality theory, which is then relaxed and linearized based on the outer approach [87] or disjunctive constraints [85], resulting into an MILP problem [88, 89].

- **Explicit Method:** As shown in Fig. 1.3, enumeration strategy is adopted by the explicit method to determine the worst scene. Since the number of subproblems presents an exponential dependence with n, K^G, and K^L, the explicit method is criticized a lot. Nevertheless, it is easier to be parallelized.

1.3.4 Parallel RTOPF with Penetration of Renewable Energy

Two types of methodologies are commonly resorted to address the RTOPF problem with the consideration of renewable energy generators:

- **Probabilistic method:** Suppose the probability distribution functions of the uncertain demands and generators are given, the stochastic optimization problem is formulated and addressed with Chance-Constrained Programming (CCP) technique, which ensures that each constraint will be satisfied in a predefined high probability. The CCP is a relatively robust approach, nevertheless, it is complex and difficult to solve and; therefore, its application is usually restricted to long-term off-line optimization [90–92].

- **Deterministic method:** Instead of performing the prediction with historical data and mathematical statistical model, different types of sensors designed for temperature, luminance, and wind speed, etc., can be widely utilized for observing. Since the obtained dataset consists of fixed real values rather than PDFs, a deterministic optimization problem can be formulated, whose solution is easier and more accurate when compared with its probabilistic counterpart.

Based on the above analysis, the deterministic method is advisable for the solution of RTOPF. As reported in [22, 23, 93], the RTOPF is carried out in every 5–15 min intervals based on the static snapshot forecast data. Fig. 1.4 illustrates the traditional implementation framework of RTOPF. At the beginning of each interval i, RTOPF$_i$ is carried out to produce the control decisions for interval $i+1$. Theoretically, the real-time observed dataset O_{i+1} should be utilized as the input for RTOPF$_i$ since the decision is made for interval $i+1$. Nevertheless, O_{i+1} cannot be gathered until t_{i+1}, i.e., utilizing O_{i+1} at t_i is not

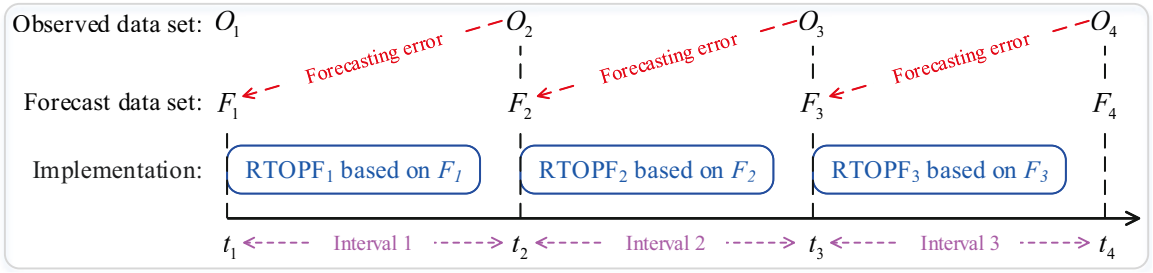


Figure 1.4: Traditional implementation framework of RTOPF.

achievable. Therefore, forecast dataset F_i is generated for the substitution of O_{i+1}. Since F_i is derived from O_i, and O_i is different from Q_{i+1} as their time-stamps are various, therefore the variation between F_i and O_{i+1} is inevitable, and it goes higher as the length of interval increases. In order to mitigate the forecasting error, a lot of research effort has been put on the development of advanced physical and statistical algorithms, such as conditional kernel density estimation [94], artificial neural network [95], and numerical weather prediction grids [96], etc.

The aforementioned methodologies provide one promising direction to improve the accuracy of deterministic RTOPF; however, the pure prediction technique is approaching its maximum capability. Therefore, further improvements from other directions have been explored. In [93], the regular RTOPF scheduling interval (10 min) has been divided into several subintervals (1 min) based on the introduction of participation factors for each generator. The time resolution has been enhanced, thus the prediction error was reduced. On the other hand, acceleration on the solution process of RTOPF is also constructive for the reduction of the length of intervals in Fig. 1.4. The projected gradient descent was employed for on-line OPF in [97], where good performance has been reported for both solution efficiency and global convergence. Ref. [98] combines the genetic algorithm and two-point estimate method to effectively tackle the uncertainties. Quasi-Newton method is resorted to developing a real-time algorithm for AC OPF in [99], where the second order information is utilized to provide suboptimal solutions on a fast timescale.

Although these methods are valid and beneficial, they share one common limitation that F_i in Fig. 1.4 is directly utilized to represent O_{i+1}, i.e., there is an implicit hypothesis that the forecast data set is 100% acceptable during each interval, which may not always be true. To address this issue, several F_i were generated to predict Q_{i+1} in [100]. By solving different RTOPFs corresponding to each F_i, a lookup table was maintained. At time t_{i+1}, the O_{i+1} is available, then the closest F_i can be determined, thus quick decisions can be made by indexing the lookup table. This solution framework is beneficial to minimize to eliminate the forecasting error between F_i and O_{i+1}; however, the intensive computational

burden due to large numbers of F_i consists one of its greatest drawbacks, thus only 2 REGs and 49 scenarios were considered in [100]. In addition, the difference between F_i and O_{i+1} might still be large after minimization. To address these two concerns, the GPU is beneficial for the acceleration of lookup table formulation since RTOPFs corresponding to each F_i are independent and can be solved in parallel.

1.3.5 Fast DNRC with Graph Theory and Direct Approach

As indicated in the problem definition, two main concerns need to be addressed for the solution of DNRC. In terms of the RTS issue, two types of heuristics are widely investigated:

- **Modification:** This kind of method begins with an RTS tree. Firstly, add a new link to the tree. Due to RTS feature and graph theory, the resulted figure does not hold RTS since a cycle is formulated. Secondly, in order to obtain RTS again, select one edge belonging to the cycle and remove it. This method is named *branch exchange* in the literature, i.e., exchanging two edges in different sets. Within this framework, various strategies are proposed [101–103] to determine the selection strategy for branches.

- **Generation:** This type of method starts from the full graph G. A successive process should be implemented to generate a candidate RTS. At each step, identify one cycle in G, and then select one edge in that cycle to delete. By doing this, that loop is broken. The process continues until there is no cycle in G. Based on how to select edge in each loop, different methods are developed [104–107].

Although these heuristics are straightforward and easy to implement, they are greedy and the final solution depends on the initial configuration [19], thus it is a local optimum rather than the global best. Therefore, they are commonly integrated into a meta-heuristic framework to achieve the global optima. Nevertheless, the encoding and decoding strategies are usually complicated or inefficient due to the identification of different loops. For example, the integer coded genetic algorithm is utilized in [108], where crossover and mutation operations are based on the matroid theory, which is highly dependent on the loops. Instead of dealing with loops, a novel decimal encoding technique is proposed in [27], where the RTS is naturally guaranteed for each candidate based on the Minimum Spanning Tree (MST) calculation. Given an initial graph G with varied weights of edges, different trees can be generated by advanced MST algorithms. This method is direct, but the solution process is time-consuming because the MST searching is computationally intensive and involved into each candidate generation.

To address the DNPF concern, techniques widely utilized for transmission network have been modified and applied, such as the FD [109–111] and NR [112–114] algorithms. One limitation of this kind of method is that the successive admittance matrices must be updated in each iteration due to the topology variation. The construction and factorization of these matrices bring heavy computation burden, thus the solution efficiency is limited. On the other hand, the Backward/Forward Sweep (BFS) method [115] and its variants [116, 117] have gained great popularity for the DNPF solution due to their good convergence and easy applicability. One major drawback of the BFS as summarized in the literature [118] is the requirement on the numbering schemes for the system buses and/or branches, which could reduce the flexibility and affect its ability to accommodate changes in topology. In addition, the solution time of DNPF with BFS is mainly determined by the number of system buses, thus the capability for large-scale systems is restrained.

Instead of the time-consuming LU decomposition and backward/forward substitution, a distinctive Direct Approach (DA) was proposed in [119], where only the matrix multiplications were involved. The essential processes related to two newly defined matrices, i.e., the Bus-Injection to Branch-Current (BIBC) matrix and the Branch-Current to Bus-Voltage (BCBV) matrix. The advantageous of DST has been revealed in [119, 120]. Although the formulation algorithm of BIBC and BCBV reported in [119] is intuitive, the efficiency is limited since only one branch is considered at each step.

1.3.6 Parallel RTVVO with Distributed Generators

Due to different nature of control devices, VVO is conventionally formulated as a Mixed-Integer Non-Linear Programming (MINLP) problem with complex numbers and constraints. Two types of methods have been widely utilized for the solution of such a Nondeterministic Polynomial hard (NP-hard) problem: 1) mathematical programming methods, including robust optimization [29, 33], mixed-integer programming [28, 32], BD [34], model predictive control [35], etc.; 2) meta-heuristic methods, such as genetic algorithm [36, 37], PSO [38, 39], simulated annealing [40], etc. A common feature for many of the proposed mathematical programming methods is the relaxation of original variables and constraints [36]. For example, linear power flow formulation is utilized in [28], and integer variables are relaxed into continuous during computation and discretized subsequently in [31]. Although relaxation can facilitate computation and mathematical programming methods are deterministic, the accuracy is sacrificed. On the other hand, the meta-heuristic methods are capable to address various types of problems of their original form, such as non-linear, non-convex, mixed-integer, NP-hard, and combinatorial, etc. Therefore, the ACPF calculation for distribution network can be fully integrated into the VVO framework. Nevertheless, they are usually computational intensive.

In order to alleviate the computational burden and accelerate the solution process of VVO, parallel computing technique has been introduced in [38]. The parallel realization of PSO was fulfilled with OpenMP [121] on CPU, resulting speedups of 1.95×, 3.42×, and 3.72× with 2, 4, and 8 threads, respectively. It is observable that the parallel efficiency faced with the bottleneck on the 8 threads. In addition, the investigated system is only 14-bus, which is partially due to the contradiction between real-time requirement and solution efficiency. Actually, for other references related to VVO, whether the real-time and ACPF are considered or not, the target system scales are also limited without the introduction of HPC platform and parallel computing technique.

1.4 Motivation and Objective

As discussed in the literature review, the advantageous endeavor has been made to improve the solution efficiency and accuracy of optimization problems for large-scale power systems planning and operation. Although most of them are remarkable with magnificent results, the exploration is still far away from the destination since new challenges and opportunities continue to emerge. Therefore, taking new opportunities to address new challenges consists the major motivation of this work. The derivation of countermeasures is mainly based on algorithm refining and computation architecture updating according to the characteristics of target problems and solution methodologies. Since the challenges have been illustrated in the above, the following is devoted to opportunities extraction, which also comprises the main objective of this work.

- **Meta-Heuristic Algorithms for TEP**
  When utilizing meta-heuristic algorithm for the solution of TEP, the execution time is still a bottleneck, which is mainly due to premature of meta-heuristics and large numbers of LP solution. The population of classical PSO and most of its variants are led by a single global best, which usually results into premature and local optimal solution; therefore, their applications are commonly limited to small- or medium-scale systems. In this study, acceleration strategies will be proposed with the combination of multi-group co-evolution strategy and LU decomposition method for TEP.

- **Deterministic Algorithms for SCTEP**
  As a deterministic algorithm, BD has been widely utilized for the solution of SCTEP, however, most of them are implemented in the classical framework, i.e., iteratively solving MILP master problem and LP subproblem until the convergence criteria are met. It can be very expensive from a computational viewpoint since the solution of MILP (which is much harder to be tackled than LP) is heavily involved. Therefore,

in this thesis, instead of solving the MILP master problem, the BD process will be integrated into a Branch-and-Cut (B&C) framework to save computational effort and execution time, where only the relaxed LP master problem needs to be solved at each node. In addition, acceleration strategies will also be investigated.

- **Fast Decoupled Method and Direct Linear Solver for ACPF with GPU**
  ACPF analysis is one of the most fundamental tasks for the power system operation and optimization, which dominates the essential steps of many practical problems. FDPF received much attention for the solution of ACPF due to its lighter computation burden. In the process of FDPF, a lot of Linear Equation Systems (LESs) can be solved with both iterative and direct solvers. Although it is reported that the iterative solver is more desirable for the solution of large-scale LESs in the context of parallel computing, this work intends to explore the potential of direct solver for DNPF. The procedure of the direct solver usually consists of factorization and substitution, which matches the property of FDPF, i.e., factorization results are reusable. In addition, the direct solver is more robust for ill-conditioned problems. Implementation will be conducted on GPU with both Matlab and CUDA.

- **Compensation Method for RTCA with GPU**
  A series of ACPFs need to be efficiently solved in the RTCA problem. Although FDPF with direct linear solver and GPU has gained good performance for the solution of single ACPF, improvement is still possible since all the ACPFs are similar. Based on the factorization result of a specific matrix, the CM can derive factorization results for other similar matrices with limited operations based on the sensitivity analysis. Nevertheless, the implementation of CM on GPU has never been reported to the best of our knowledge. Thus, this work intends to fill this gap with the presentation of detailed implementation schemes and performance validations.

- **Robust Optimization Frameworks for SCUC**
  Two implementation frameworks have been developed for RO, where implicit method gained a good reputation and explicit method was criticized a lot. However, with the popularity of parallel processing, some promising features of explicit method should not be overlooked: 1) each subproblem of characteristic is LP, which is much easier to solve than MILP; 2) the subproblems are independent with each other, thus they can be solved simultaneously, i.e., suitable for parallel computing; 3) not only the most violated scenario can be found, other violated contingencies can also be identified, therefore, more cuts can be generated to enhance the convergence; 4) more information on the violated scenarios is beneficial for the releasing of the conservatism.

Therefore, in this thesis, both frameworks will be implemented and compared to evaluate their potential for SCUC, especially in the context of parallel computing.

- **Primal-Dual Interior Point Method for RTOPF with Batched GPU**
  With the introduction of renewable energy generators and loads, the forecasting error is inevitable. One of the possible alternatives to minimize the mismatch is generating more scenarios for estimation. Based on the results from different scenarios, a lookup table can be maintained, which can be quickly indexed when the uncertainty is realized. However, the implementation is not trivial since large numbers of scenarios should be addressed within limited time. In order to address this concern, all scenarios will be solved simultaneously with batched GPU mode, where PDIPM is employed for optimization. By tuning matrices into the same sparsity pattern, the batched mode can do the factorization for a lot of matrices at the same time. Not only the execution time but also the storage space can be saved.

- **Direct Approach and Graph Theory for DNRC**
  During the solution of DNRC, two main concerns are RTS and DNPF. Although the MST method performs better than other algorithms for the pursuing of RTS, it is still computationally expensive. In this work, a new RTS formulation strategy based on the loop detection and destruction will be proposed. Since a lot of radial trees need to be generated from the graph (original distribution network) during the solution process, the graph loop information is reusable, thus computation effort can be saved. On the other hand, the DA is one of the most efficient methods for DNPF. Improvements on the $[BIBC]$ and $[BCBD]$ matrices formulation will be generated in this study based on the adjacency and path matrices from graph theory. Instead of the classical method that considering one switch at a time, the proposed method handles all branches concurrently with matrix operations.

- **PSO and GPU for RTVVO**
  The popularity AMI is driving the DAVVO into RTVVO from the perspective of communication and controlling. Nevertheless, the solution efficiency of RTVVO is rarely discussed in the literature. This thesis intends to fill that gap. Firstly, the DNPF solution method within RTVVO will be changed from NR and FDPF into DA due to its regulated matrix operations. Full mathematical model of VVO control components is formulated and integrated into the solution process of DA, including SC, OLTC, and DG. Then, a whole solution scheme developed with PSO will be implemented on GPU, with data structure design and thread organization are fully revealed.

1.5 Thesis Outline

In this thesis, two computation platforms, five programming languages, three parallel architectures, four solver packages, and twelve algorithms/methods are utilized and consulted to achieve the performance enhancement for various large-scale power system planning and operation problems. Fig. 1.5 illustrates the mapping relationship between these items and main chapters. Supportive descriptions are given as follows:

- **Chapter 1: Introduction** - In the context of coexisting challenges and opportunities, various power systems planning and operation optimization problems are defined and clarified to highlight the scope of the thesis in this chapter. Advantageous work related to each problem is reviewed to sketch out the achievements and limitations. Finally, the motivation, objective, and outline are summarized.

- **Chapter 2: Transmission System Optimal Expansion Planning: TEP and SCTEP** - Due to the boost of loading levels and the wide utilization of distributed generators, TEP has regained its significance for investigation. Without generation redispatch, the LP subproblem of DC TEP is transformed into LES, which is much easier to be solved than LP with LU decomposition. Based on the discrete PSO framework, MGPSO is proposed with the introduction of beneficial enhancements, such as Sobol sequence initialization method, multi-group co-evolution strategy, and mutation mechanism. Superiority over commercial software Lingo is established with case studies. On the other hand, the disjunctive model is utilized to formulate the SCTEP into MILP problem, where integer numbers are replaced with binary numbers and nonlinear constraints are relaxed into linear ones with the introduction of linearizing constant (big-M). Due to large numbers of decision variables, BD is employed to divide and conquer. In order to achieve better solution efficiency, BCBD algorithm is proposed by the integration of BD into a B&C framework. Different from BD with MILP master problem, the master problem in BCBD is LP, which is much less computationally intensive than MILP. Four acceleration strategies are developed to further improve the efficiency of BCBD. Better performance has been observed from the comparison with MILP solver Cplex, reducing the execution time from days to hours.

- **Chapter 3: Transmission System Optimal Operation: FDPF and RTCA** - ACPF analysis is one of the most fundamental tasks for the transmission system operation and optimization problems, therefore achieving high solution efficiency for ACPF analysis from HPC architecture is a leading and important challenge in power system analytics and computation. This chapter starts the solution of single ACPF with

Matlab and CUDA based on FD method and GPU platform. Data storage formats and fill-in reduction algorithms are compared and discussed. With this experience, single ACPF is evolved into multiple ACPF, i.e., the RTCA problem. Instead of solving ACPF individually with FD method, the CM is employed to solve all ACPFs simultaneously. Based on the sensitivity analysis of similar ACPFs, the number of matrix decomposition has been greatly reduced. Strategies and principles on the data structure, kernel function, and memory management are designed for GPU implementation. Good performance on accuracy, convergence, and scalability of CM running on GPU with CUDA has been validated with five benchmark systems ranging from 300- to 13,659-bus. It is concluded that the parallel CM is promising for industrial application since it is capable to finish the whole $N-1$ RTCA analysis for the 13,659-bus system within one minute.

- **Chapter 4: Generation System Optimal Operation: SCUC and RTOPF** - All power in the grid is provided by the generator, thus the optimal operation of generation system is of great significance for the economy and security. In this chapter, SCUC is solved to determine the on-off status of each thermal generator. In order to reveal the capability of different methods on the solution of CCUC, both explicit and implicit decomposition frameworks have been investigated, as well as their inner feedback strategies, such as BD and CCG algorithm. In addition, sensitivity analysis, multi-cut strategy, and parallel implementation have also been analyzed and discussed. Results indicate that explicit method is competitive with the implicit method in the context of parallel computing. On the other hand, RTOPF is introduced to optimize the active power output of thermal generators. Nevertheless, the prediction error is inevitable if renewable energy is considered. To improve the solution efficiency and accuracy of RTOPF, a three-stage framework for parallel processing is developed. In Stage 1, uncertainties from renewable generators and demand loads are characterized with scenarios. Large numbers of RTOPFs corresponding to each scenario are formulated and addressed in Stage 2, where the linear systems are regulated into the same sparsity pattern and then tackled in a batched style with the GPU. Results from Stage 2 are utilized in Stage 3 to perform a hot-start RTOPF, where the forecasting error can be minimized. Case studies are implemented on various systems to validate the effectiveness of the proposed implementation framework with GPU batched mode.

- **Chapter 5: Distribution System Optimal Operation: DNRC and RTVVO** - Due to the low voltage level of distribution network, significant power losses are encountered. In order to minimize them, DNRC and RTVVO are investigated from two different aspects, i.e., determining the open/close status of switches and controlling

a set of devices, respectively. In order to accelerate the solution process of DNRC, two essential components of meta-heuristic algorithms are investigated: solution representation and fitness evaluation. An efficient decimal solution encoding and decoding strategy is proposed based on the graph theory. DNPF solution process is also accelerated based on the new generation process of $[BIBC]$ matrix in DA. Superiority of the two proposals over advanced counterparts reported in the literature is established with case studies. GPU is introduced in this chapter for the first time to evolve the day-ahead VVO into RTVVO. PSO is employed as the solution framework, where DNPF subproblem is addressed with DA. Mathematical formulation of various components is integrated into DA to achieve higher accuracy. Well-established data structure and thread organization pattern are also provided for GPU implementation. Based on the comparison with CPU implementations, the promise of the proposed GPU parallel implementation scheme for practical application is established.

- **Chapter 6: Conclusions and Future Works** - The contribution of this research and the future works are summarized in this chapter.

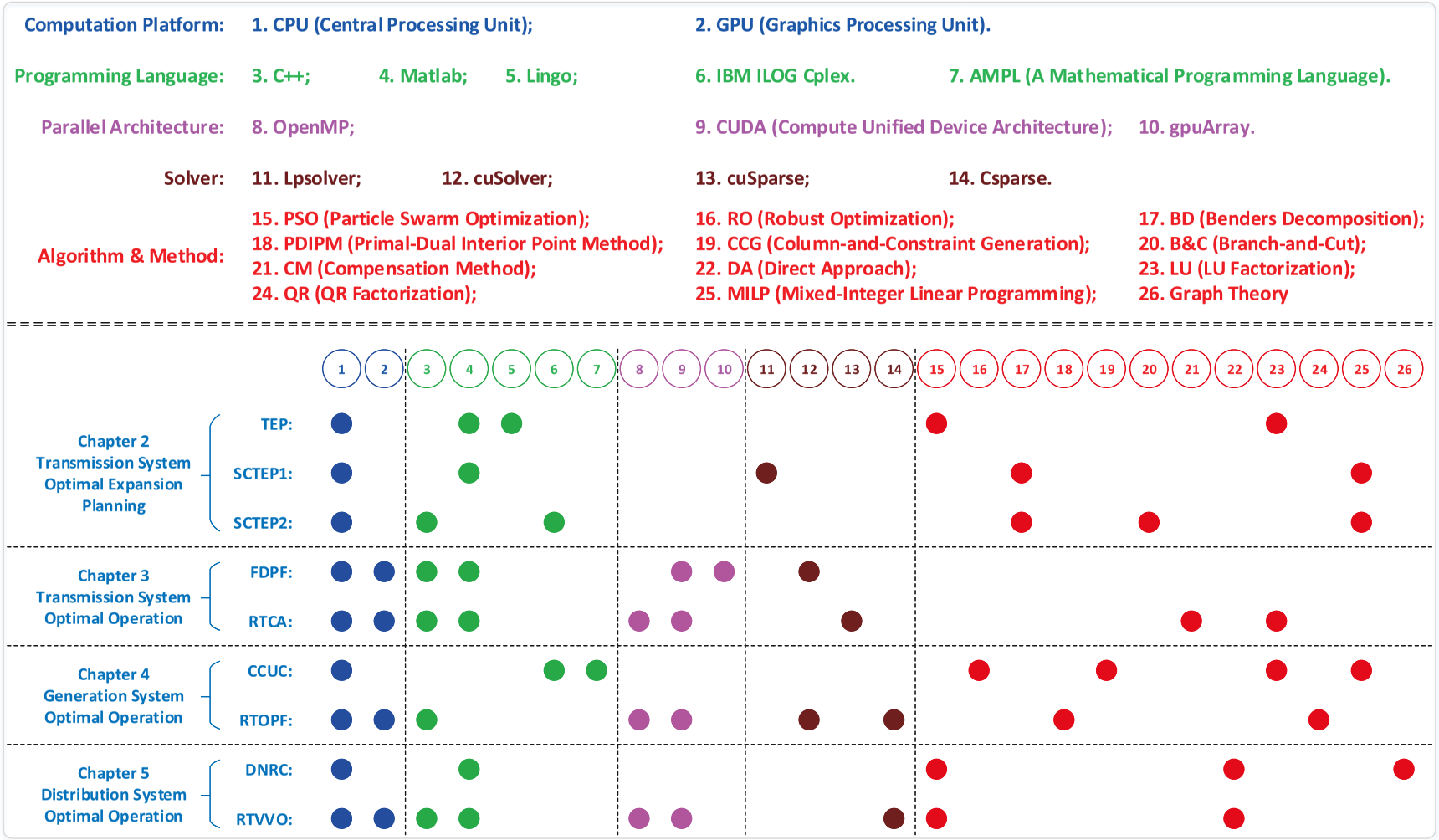


Figure 1.5: Outline of this thesis along with technical details.

2

Transmission System Optimal Expansion Planning: TEP and SCTEP

2.1 Introduction

The emergence of a great number of regional planning projects worldwide has considerably increased the complexity and relevance of TEP, prompting intensive research and investigation on the formulation and solution [48, 59]. In addition, reliability and security concerns of electricity supply during this radical evolution of modern power systems is causing operators and engineers to find improved and efficient solutions to SCTEP [10, 122]. In this chapter, both problems are addressed with meta-heuristic and deterministic methods, respectively.

At the beginning, the widely utilized DC power flow model is employed for TEP, resulting in a mixed-integer, nonlinear, non-convex, and NP-hard problem [123]. Due to its great complexity, the meta-heuristic method PSO is determined since it is robust for various types of problems. In order to further improve the solution efficiency, the evolutionary mechanism is enhanced, including Sobal sequence initialization, multi-group co-evolution, and mutation. In addition, the fitness evaluation process is also accelerated by replacing the LP with LES, which is much easier to be solved with LU decomposition.

Based on the experience for TEP, the SCTEP is investigated subsequently. The consideration of security criteria greatly increases the complexity, thus the disjunctive power flow model is utilized, leading to a MILP problem. Although there are various off-the-shelf solvers available for MILP problems, major concerns are still emerged for large-scale systems due to their thousands or millions of decision variables and constraints. Therefore,

BD is introduced to divide and conquer. In the literature, there are few fields where BD can work very well without much additional implementation work, i.e., enhancements to BD are almost always necessary. Instead of the classical acceleration strategies to generate tighter cuts, a new implementation framework is proposed by integrating the BD into a B&C skeleton, resulting in the BCBD algorithm. Furthermore, four acceleration strategies are also tailored for the solution of SCTEP.

2.2 Transmission Expansion Planning

This section intends to address the TEP with MGPSO, where formulation and implementation details are presented in subsection 2.2.1 and 2.2.2 respectively; performance validation is demonstrated in subsection 2.2.3.

2.2.1 TEP Problem Formulation

2.2.1.1 DC Power Flow Model

DC power flow model is the basic network model for TEP study, which can be formulated as follows:

$$\min_{n_{ij},\, r_k,\, f_{ij},\, \theta_i,\, g_i} \quad \sum_{(i,j)\in\mathcal{C}} c_{ij} n_{ij} + P \sum_{k\in\mathcal{N}_b} r_k, \tag{2.1}$$

$$s.t. \quad \boldsymbol{S}\boldsymbol{f} + \boldsymbol{g} + \boldsymbol{r} = \boldsymbol{d}, \tag{2.2}$$

$$f_{ij} - \gamma_{ij}\left(n_{ij}^0 + n_{ij}\right)(\theta_i - \theta_j) = 0, \tag{2.3}$$

$$|f_{ij}| \le \left(n_{ij}^0 + n_{ij}\right)\bar{f}_{ij}, \tag{2.4}$$

$$\mathbf{0} \le \boldsymbol{g} \le \bar{\boldsymbol{g}}, \tag{2.5}$$

$$\mathbf{0} \le \boldsymbol{r} \le \boldsymbol{d}, \tag{2.6}$$

$$0 \le n_{ij} \le \bar{n}_{ij},\ n_{ij} \text{ integer},\ \theta_i \text{ and } \theta_j \text{ unbounded}. \tag{2.7}$$

where

$$\boldsymbol{S}_{ij} = \begin{cases} -1, & \text{if branch } i-j \text{ originates from bus } i, \\ 1, & \text{if branch } i-j \text{ terminates at bus } j, \\ 0, & \text{otherwise.} \end{cases} \tag{2.8}$$

The objective function (2.1) comprises of the construction cost $\sum_{(i,j)\in\mathcal{C}} c_{ij} n_{ij}$ and the load curtailment penalty $P\sum_{k\in\mathcal{N}_b} r_k$. Equation (2.2) is the bus balance constraint. Equations (2.3) – (2.4) are DC power flow constraints. Equations (2.5) – (2.7) are limit constraints for the generation, load curtailment, and the number of new lines to be built.

Algorithm 2.1 Pseudo code of meta-heuristic algorithms for TEP solution

1: Propose a candidate solution vector $\boldsymbol{n}$.
2: **while** Terminate conditions are not met **do**
3: Evaluate the quality of $\boldsymbol{n}$ by LP (2.9) – (2.12).
4: Evolve to a new solution $\boldsymbol{n}$ by several operation processes, such as crossover, mutation, velocity adjusting, etc.
5: **end while**
6: **return** The final solution.

2.2.1.2 Linear Programming Subproblem

The TEP (2.1) is a MINLP problem, which has been proved to be a hard combinatorial problem [45,46]. However, if a candidate solution $\boldsymbol{n}$ is given, i.e., $\hat{n}_{ij}$ is fixed, problem (2.1) – (2.7) can be converted into an LP problem and rewritten as follows:

$$\min_{r_k,\, g_i,\, \theta_i} \quad \sum_{(i,j)\in\mathcal{C}} c_{ij}\hat{n}_{ij} + P\sum_{k\in\mathcal{N}_b} r_k, \tag{2.9}$$

$$s.t. \quad \boldsymbol{B\theta} + \boldsymbol{g} + \boldsymbol{r} = \boldsymbol{d}, \tag{2.10}$$

$$\gamma_{ij}\left(n_{ij}^0 + \hat{n}_{ij}\right)|\theta_i - \theta_j| \le \left(n_{ij}^0 + \hat{n}_{ij}\right)\bar{f}_{ij}, \tag{2.11}$$

$$\text{Constraints } (2.5) - (2.7). \tag{2.12}$$

where constraints (2.10) and (2.11) are derived from (2.2) and (2.4) respectively, with f_{ij} replaced by $\gamma_{ij}\left(n_{ij}^0 + \hat{n}_{ij}\right)(\theta_i - \theta_j)$ according to (2.3). Accordingly, $\boldsymbol{Sf}$ is rewritten into $\boldsymbol{B\theta}$, where $\boldsymbol{\theta}$ represents the vector of voltage phase angles, and $\boldsymbol{B}$ is a symmetric singular matrix generated by:

$$\boldsymbol{B} = -\boldsymbol{S}\left(\left(\left(\boldsymbol{\gamma}.*\left(\boldsymbol{n}^0 + \boldsymbol{n}\right)\right)\mathbf{1}_{1\times n_b}\right).*\boldsymbol{S}^T\right), \tag{2.13}$$

where $\mathbf{1}_{1\times n_b}$ is a vector of ones by the scale of $1 \times n_b$ and "$.*$" means element-wise multiplication operator in Matlab, $\boldsymbol{\gamma}$, $\boldsymbol{n}^0$, and $\boldsymbol{n}$ are vectors of γ_{ij}, n_{ij}^0, and $\hat{n}_{ij}$ respectively, n_b is the total number of buses.

Since MINLP (2.1) – (2.7) contains large numbers of constraints, which provides great obstacles for meta-heuristic algorithm with conventional constraint handling methods, such as penalty function method, therefore, another more efficient strategy (shown in **Algorithm 2.1**) has been proposed and employed by the majority of meta-heuristic algorithms when tackling TEP problems.

2.2.1.3 Linear Programming Transformation

In order to gain higher solution efficiency, the LP (2.9) – (2.12) was transformed into a new form by [45] via a series of processes, resulting in the number of decision variables reduced

from $2n_b$ to n_b, and the number of constraints decreased from $2n_b + 2n_c$ to $n_b + 2n_c + 1$, where n_c is the number of candidate circuits. Due to the reduction in problem scale, the revised LP formulation gained a better performance than Minos 5.4 software in [45]. However, it was pointed by the authors themselves that the proposed transformation presented a disadvantage of requiring the inversion of matrix $\boldsymbol{B}$ in an explicit way, which is relatively computationally intensive.

Instead of transforming the original LP into another LP with reduced scales, an LES is extracted in this section since it is much easier to be addressed. Different from the LP solution process with simplex method or interior point method, where large numbers of iterations are required, the LES can be efficiently solved by matrix inverse or LU decomposition. In order to fulfill the transformation from LP to LES, the following two assumptions are proposed:

- Assumption 1: $r_k = 0$ for all k. It is required that all load buses are satisfied by generation buses for the optimal solution, i.e., for each bus the loss of load is 0, which means $r_k = 0$.
- Assumption 2: $\boldsymbol{g} = \bar{\boldsymbol{g}}$. This assumption holds for all the TEP without the consideration of generation redispatch, such as the real Brazilian 46-bus system [13,17,48,124].

Based on Assumptions 1 and 2, LP problem (2.9) – (2.12) can be modified into:

$$\min_{s_{ij},\,\theta_i} \quad \sum_{(i,j)\in\mathcal{C}} c_{ij}\hat{n}_{ij} + P\sum_{(i,j)\in\mathcal{C}} s_{ij}, \tag{2.14}$$

$$s.t. \quad \boldsymbol{B\theta} + \bar{\boldsymbol{g}} = \boldsymbol{d}, \tag{2.15}$$

$$\gamma_{ij}\left(n_{ij}^0 + \hat{n}_{ij}\right)|\theta_i - \theta_j| \le \left(n_{ij}^0 + \hat{n}_{ij}\right)\bar{f}_{ij} + s_{ij}, \tag{2.16}$$

$$s_{ij} \ge 0. \tag{2.17}$$

where s_{ij} is the relaxation coefficient. Compared with (2.9) – (2.12), decision variables $\boldsymbol{g}$ and $\boldsymbol{r}$ are eliminated, as well as the constraints of limitation for them.

The key point for solving LP (2.14) – (2.17) is deriving $\boldsymbol{\theta}$ from LES (2.15), after which, another decision variable s_{ij} is very easy to be extracted from (2.16) with simple substitution process. There are several methods to tackle with LES (2.15), such as LU decomposition. The details to solve (2.14) – (2.17) will be explicated by **Algorithm 2.2** in the next section.

2.2.2 Multi-Group Particle Swarm Optimization

PSO was first proposed by Kennedy and Eberhart [125] in 1995 with the core idea of sharing information between particles, local best, and global best, which can be expressed as:

$$v_i^{k+1} = w_0 v_i^k + c_1 w_1 (p_i^k - x_i^k) + c_2 w_2 (g^k - x_i^k), \tag{2.18}$$

$$x_i^{k+1} = x_i^k + v_i^{k+1}, \tag{2.19}$$

where v_i^{k+1} and v_i^k are the velocities of particle i at the $(k+1)$th and (k)th iteration; x_i^{k+1} and x_i^k are the positions of particle i at the $(k+1)$th and (k)th iteration; $w_0 \in [0, 1]$ is the inertia weight; $c_1, c_2 \in [0, 2]$ are the self-knowledge and social-learning factors; $w_1, w_2 \in [0, 1]$ are random numbers; p_i^k is the local best of particle i till (k)th iteration; g^k is the global best till (k)th iteration. Based on the classical PSO, a MGPSO has been proposed for the solution of TEP problem as follows.

2.2.2.1 Problem Codification

The design of any iterative meta-heuristic algorithm requires an encoding of the solution, which plays a crucial role in the efficiency and effectiveness. Several alternative codification approaches were proposed in the literature, including binary codification, independent bits, and decimal codification, etc. In this work, the decimal codification is employed based on the following two considerations:

- Both binary and independent bits have Hamming cliffs, introducing great obstacles for convergence. For example, two successive numbers 3 and 4 in decimal will be expressed totally inverse by binary bits as "011" and "100" respectively.
- The individual length is longer if the solution is represented by binary bits and the transform process from binary to decimal will be executed for thousands of times, which requires more computational resources.

2.2.2.2 Population Initialization

In order to gain higher diversity, Sobol sequence [126] is adopted in this work. Different from the classical pseudo-random sequences, the Sobol sequence is a quasi-random sequence. It can be observed from Fig. 2.1 that the Sobol sequence is more uniformly sampled than pseudo-random sequence, where a total number of $N = 500$ points is sampled for each method in a 1×1 square area. In order to do a numerical analysis, both axes are divided into $W = 1...10$ segments, resulting in W^2 square subareas. If the points are ideally distributed in even, each subarea should have N/W^2 points. Suppose subarea ij has y_{ij} points ($i = 1...W$, $j = 1...W$), then the abstract biases for each subarea from its ideal

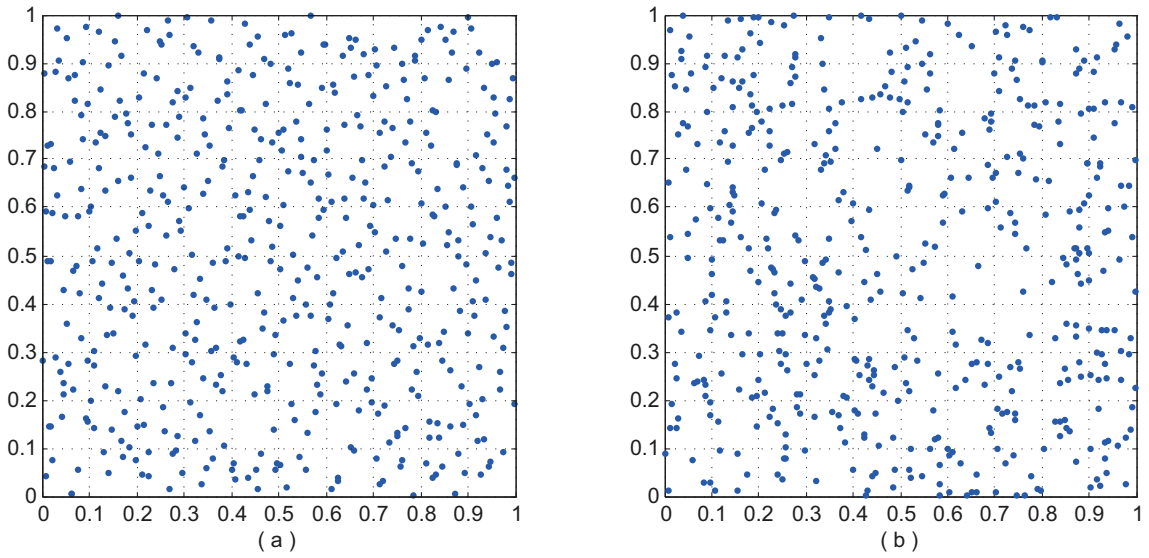


Figure 2.1: Sample points of different random sequences: (a) Sobol sequence; (b) pseudo-random sequence.

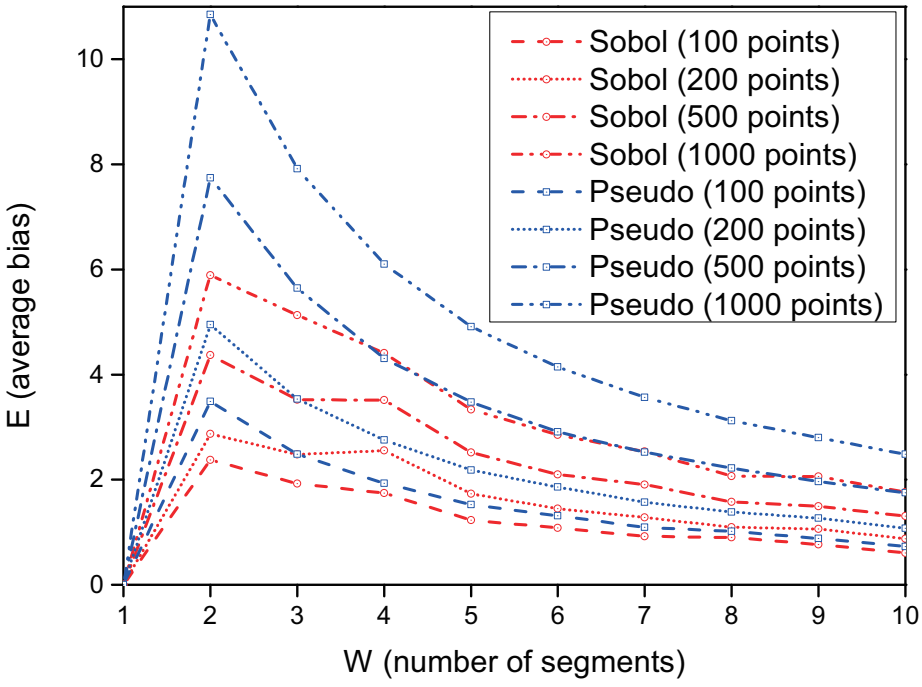


Figure 2.2: The relationship between E and W for different N.

value can be derived, and their average value E can be used to illustrate the uniformity of each sequence.

$$E = \frac{\sum_{i=1}^{W} \sum_{j=1}^{W} |y_{ij} - N/W^2|}{W^2}. \tag{2.20}$$

Fig. 2.2 shows the average bias E across different W and N. It can be seen that for the fixed N, E is getting smaller as W increases, which is due to the reduction of the ideal value N/W^2. Another important feature is that the average bias of Sobol sequence is always smaller than pseudo sequence for any fixed N and W, indicating that the Sobol sequence is evenly distributed, i.e., diversity is guaranteed.

2.2.2.3 Particle Evolution

Particle evolution is the kernel of all types of PSO ranging from single-group to multi-group. For the discrete version of PSO, the most commonly utilized method is regulating velocity and position to be integer using functions such as $fix()$, $round()$, and $ceil()$. With the continuous of convergent process, the current solution x_i^k is getting close to its local best p_i^k and global best g^k, thus the real value of velocity is usually within an interval of $(-1, 1)$ according to (2.18). At this circumstance, $fix()$ and $ceil()$ may have huge error, such as $fix(0.9) = 0$ and $ceil(0.1) = 1$, therefore $round()$ is utilized by [48] and (2.18) is also modified into (2.21) to increase diversity, especially for the final stages.

$$v_i^{k+1} = round\left(w_0 W_0 v_i^k + c_1 W_1 (p_i^k - x_i^k) + c_2 W_2 (g^k - x_i^k)\right), \tag{2.21}$$

where W_0 is a random number taking discrete values of 0, 1, or -1; W_1 and W_2 are random discrete numbers of either 0 or 1; all the other variables and parameters share the same definition as (2.18) and (2.19). Different from (2.18), equation (2.21) provides more possibilities by the introduction of W_0, W_1, and W_2, such as temperately eliminating the influence of velocity, local best, and global best by setting W_0, W_1, and W_2 into 0 respectively. The search direction can even be turned totally inverse if $W_0 = -1$. These possibilities bring more diversity to the searching process. The performance of this strategy has been verified by [48], and it will be employed as the evolution strategy for each particle of MGPSO. If the velocity is too high, particles might fly past good solutions; if it is too small, particles may not capable to explore beyond local regions [127]. Therefore, the velocity is restricted into $[-2, 2]$ in this work.

2.2.2.4 Multi-Group Co-evolution

Although particle evolution strategy shown above has a good performance reported in [48], beneficial improvements are still in demand. For example, the current global best g^k involves in the evolution process of all particles in (2.21), which forces the whole population to converge to a small space dominated by g^k, leaving large areas unexplored, where the real global optimum might exist, i.e., the final solution is a local best. Therefore, in order to reduce the strong influence of single global best, the whole population was divided

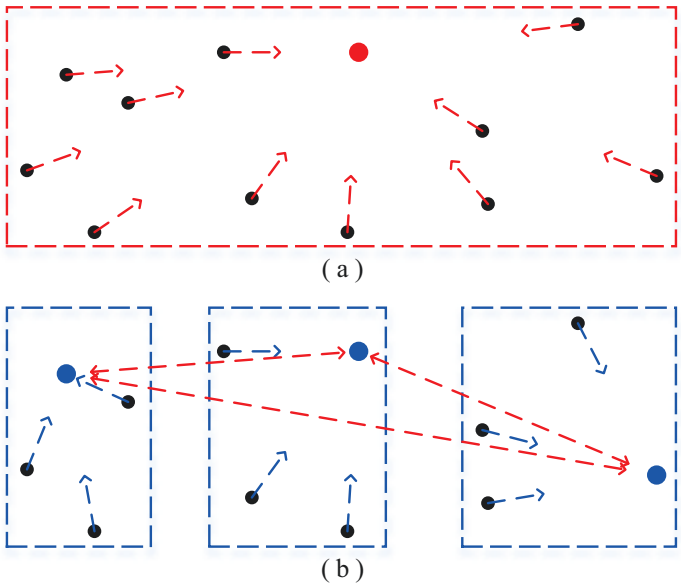


Figure 2.3: Illustration of PSO evolution strategies: (a) single-group evolution; (b) multi-group co-evolution.

into n_g groups to keep diversity, and each group $(j = 1 \cdots n_g)$ has a global best g_j^k until iteration k. The multi-group co-evolution strategy is guided by the following two rules:

- All the global best of different groups are different. If $g_{j_1}^k = g_{j_2}^k$ while $j_1 \neq j_2$, then a mutation process should be triggered on either $g_{j_1}^k$ or $g_{j_2}^k$. This rule makes each group driven by different "leader", which forces the whole population to explore more space.

- The global bests of different groups should share information with each other. Two-point crossover was employed to perform the information exchange on randomly selected $g_{j_1}^k$ and $g_{j_2}^k$, with two obvious benefits: a) improving the gene characteristics of poor fitness individuals, and b) introducing diversity on good fitness candidates.

Fig. 2.3 shows the difference between single-group evolution and multi-group co-evolution, where particles are represented by solid circles and the influences are illustrated with dashed arrows. In Fig. 2.3 (a), all the particles are influenced by the population global best, while in Fig. 2.3 (b) the particles are driven by each group's global best, and each global best is influenced by other global bests, which maintains a good trade-off between population diversity and global convergence.

2.2.2.5 Mutation Mechanism

To prevent premature convergence and enable the MGPSO algorithm escape from the local optimal, the mutation process has been carried out in three steps:

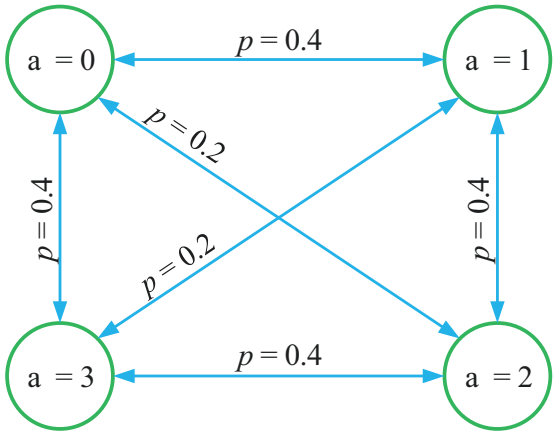


Figure 2.4: Transform probability for point mutation.

- Determine the candidate mutation individual according to a mutation rate r_m. Instead of doing the mutation directly on points of each individual with the same probability, this step could enable large numbers of particles undisturbed, leaving the feasibility maintained.
- Replace the mutation candidate with its local best. This process gives a good starting point for mutation, i.e., the mutation always finds the points near the local best, representing a higher probability to get good results.
- Perform the mutation process on the points determined by a selection probability of r_p. The point mutation process is carried out according to the transformation probability shown in Fig. 2.4. For example, the probability for $a = 0$ to mutate into $a = 1$, $a = 2$, and $a = 3$ is 0.4, 0.2, and 0.4, respectively.

2.2.2.6 Fitness Evaluation

Fitness evaluation and constraint handling consume the largest part of the time for almost all meta-heuristic algorithms. According to the power balance constraints, if the candidate solution $\boldsymbol{n}$ is infeasible, there must be some loss of load for buses, which then will be multiplied by P for penalty. In order to save computational effort, the sum of power imbalance is directly valued as M (which is a big constant number) for all infeasible candidate without any solution of LP. On the other hand, the LES (2.15) should be figured out to get the objective function value for feasible solutions.

For the purpose of checking the feasibility of each candidate $\boldsymbol{n}$, a process of network analysis consisted of connectivity detection and feasibility verification has been proposed,

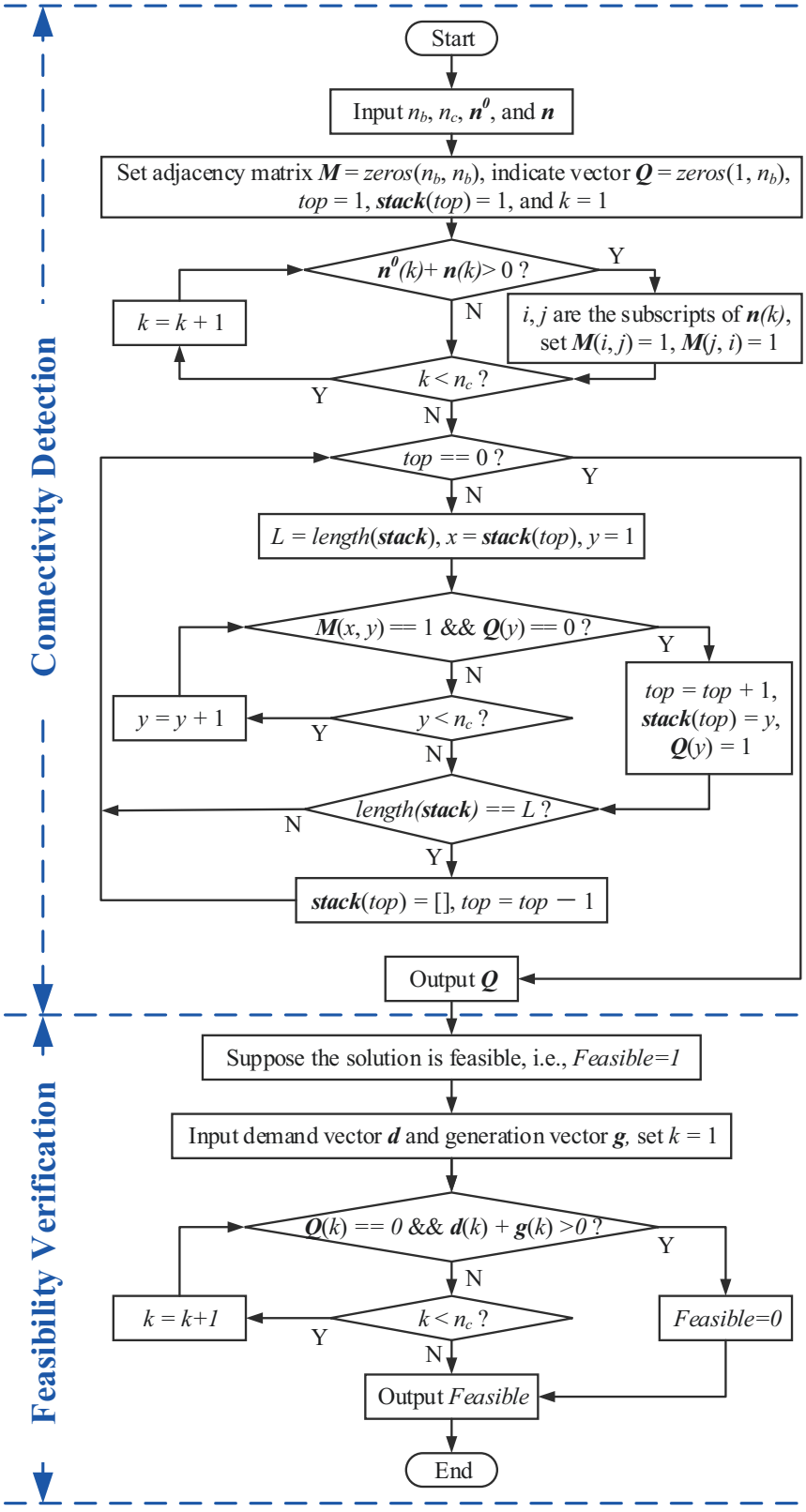


Figure 2.5: Flowchart for network analysis.

Algorithm 2.2 LU decomposition method for LP (2.14) – (2.17)

1: Get the lower triangle matrix $\boldsymbol{L}$ and the upper triangle matrix $\boldsymbol{U}$ of $\boldsymbol{B}_{nonsingular}$ by (2.23), where $lu()$ is the LU decomposition function defined in Matlab.
2: Derive the $\boldsymbol{\theta}_{nonsingular}$ by (2.24), where "\" is the left division operator in Matlab.
3: Set the value of θ_i for the eliminated buses (slack and those with $\boldsymbol{Q}(i) = 0$) as 0.
4: Generate the slack variable values s_{ij} by (2.16).
5: Return the objective function value of (2.14).

which is presented in Fig. 2.5. The connectivity detection is implemented for each bus to check whether it is connected to the grid or not, where the Depth-First Search (DFS) algorithm [128] from graph theory is adopted. The result is represented by a binary vector $\boldsymbol{Q}$ consists of n_b elements, if $\boldsymbol{Q}(i) = 1$, then bus i is connected, otherwise, it is disconnected. Based on $\boldsymbol{Q}$, the feasibility verification is performed. If the set of disconnected buses contains generation or load buses, then the power balance will be destroyed, i.e., the solution is infeasible; otherwise, candidate $\boldsymbol{n}$ is feasible.

As discussed above, the objective value of infeasible solution will be directly valued as P, while for the feasible candidate, LES (2.15) should be solved. It is required that the matrix $\boldsymbol{B}$ in (2.15) should be non-singular for the utilization of LU decomposition. However, the original $\boldsymbol{B}$ is singular, thus two steps are implemented: 1) delete column i and row i of $\boldsymbol{B}$ if $\boldsymbol{Q}(i) = 0$, resulting in $\boldsymbol{B}_{connected}$; 2) select a slack bus j and eliminate the corresponding row and column to get the final non-singular matrix $\boldsymbol{B}_{nonsingular}$. Similarly, the singular LES (2.15) can be transformed into a non-singular LES:

$$\boldsymbol{B}_{nonsingular}\boldsymbol{\theta}_{nonsingular} + \bar{\boldsymbol{g}}_{nonsingular} = \boldsymbol{d}_{nonsingular}, \tag{2.22}$$

where $\boldsymbol{d}_{nonsingular}$ and $\bar{\boldsymbol{g}}_{nonsingular}$ are demand and generation vectors with slack bus and those with $\boldsymbol{Q}(i) = 0$ eliminated. The process, based on LU decomposition since it is more efficient than matrix inverse method, to derive the objective function value of feasible candidate is illustrated in **Algorithm 2.2**.

$$[\boldsymbol{L}, \boldsymbol{U}] = lu(\boldsymbol{B}_{nonsingular}), \tag{2.23}$$

$$\boldsymbol{\theta}_{nonsingular} = \boldsymbol{U}\backslash(\boldsymbol{L}\backslash(\boldsymbol{d}_{nonsingular} - \bar{\boldsymbol{g}}_{nonsingular})). \tag{2.24}$$

2.2.2.7 Terminate Condition

The algorithm terminates if the incumbent (the best solution found so far) does not improve after a specified number n_t of iterations or the maximum numbers of iteration G is reached.

Algorithm 2.3 Pseudo Code of MGPSO

1: Input original data set Φ, where the number of candidate circuits n_c is included;
2: Initialize parameters: population size m, number of group n_g, maximum generation G, power imbalance penalty value M, penalty factor P, and terminate criterion n_t;
3: Initialize variable matrices and vectors: population $X = \textit{Sobol}(m, n_c)$, velocity $V = -1 + 2 \times \textit{rand}(m, n_c)$, global best solution and its fitness value for each group $S_g = \textit{ones}(n_g, n_c)$ and $F_g = \textit{inf}(n_g, 1)$, local best solution and its fitness value for each individual $S_l = \textit{ones}(m, n_c)$ and $F_l = \textit{inf}(m, 1)$; (Reported in section 2.2.2.2)
4: Set unimproved counter $c = 0$ and the global best of the whole population $f_b = \infty$;
5: **for** $i = 1$ to G **do** (Reported in section 2.2.2.7)
6: $F = \textit{Fitness Evaluation}(X, \Phi, P, M)$; (Reported in section 2.2.2.6)
7: $[F_g, F_l, S_g, S_l] = \textit{Group-based Coevolution}(F, n_g, F_g, F_l, S_g, S_l, \Phi, P, M)$; (Reported in section 2.2.2.4)
8: $[X, V, S_l] = \textit{Particle Mutation}(X, V, S_l)$; (Reported in section 2.2.2.5)
9: $[X, V] = \textit{Particle Evolution}(X, V, S_g, S_l)$; (Reported in section 2.2.2.3)
10: **if** $\min(F_g) < f_b$ **then**
11: Let $f_b = \min(F_g)$ and $c = 0$;
12: **else**
13: $c = c + 1$;
14: **end if**
15: **if** $c >= n_t$ **then** (Reported in section 2.2.2.7)
16: break;
17: **end if**
18: **end for**
19: **return** global best f_b and the total number of iteration i.

2.2.2.8 Implementation Framework

The overall implementation framework of MGPSO is illustrated by **Algorithm 2.3** with pseudo code, where the key functions in lines 6 – 9 have been explicated in the above subsections.

2.2.3 Case Studies and Discussion

Two types of tools are employed to perform the case studies, Matlab 2015b and Lingo 11.0, which are all run on a Windows desktop with an Intel Xeon E5-2620 CPU at 2.10 GHz with 32 GB RAM. In addition, results from the literature are also included for discussion.

2.2.3.1 Benchmark Systems

Five systems without generation redispatch are considered in this study: the Garver 6-bus system [9, 17], the IEEE 24-bus system with the third future generation plan [129], the South Brazilian 46-bus system [124], the Southeast Brazilian 79-bus system [130, 131], and

Table 2.1: Scale and complexity of test cases.

Systems	$\bar{n}_{ij}$	n_b	n_c	Search space size		
6-bus	4	6	15	5^{15}	$\approx$	3.05×10^{10}
24-bus	3	24	41	4^{41}	$\approx$	4.84×10^{24}
46-bus	3	46	79	4^{79}	$\approx$	3.65×10^{47}
79-bus	3	79	143	4^{143}	$\approx$	1.24×10^{86}
118-bus	2	118	186	3^{186}	$\approx$	5.55×10^{88}

Table 2.2: Control parameters of MGPSO for different cases.

Systems	m	n_g	P	M	G	n_t
6-bus	20	4	70	400	200	30
24-bus	200	20	30	400	400	80
46-bus	800	40	400	800	1000	100
79-bus	2000	80	600	2000	2000	300
118-bus	2000	80	600	2000	2000	300

the IEEE 118-bus system [132]. The former four are the same with the systems studied in [123], while the last one is modified from [132] by reducing the maximum capacity of each line to 40% of its original value due to over sufficient condition. All of these are classical benchmark systems and have been investigated by several researchers, an overview of whose scale and complexity is shown in Table 2.1.

2.2.3.2 Parameter Settings

Control parameters are significant for algorithm performance in relation to solution quality as well as convergence speed, which are usually problem dependent. In this work, all control parameters related to MGPSO are manually tuned based on a few preliminary experiments, which are given in Table 2.2. It should be noted that these parameters may not be optimal since the comprehensive test is not fully implemented. Apart from the above parameters related with cases, c_1, c_2, and w_0 are robust for all 4 cases, with $c_1 = 2.0$, $c_2 = 2.0$, and w_0 is expressed as follows:

$$w_0 = w_{max} - (w_{max} - w_{min}) \times i/G, \tag{2.25}$$

where $w_{max} = 0.6$ and $w_{min} = 0.2$, i is the current iteration count.

2.2.3.3 Main Results

It will be instructive to review the results reported in the literature before introducing the outputs obtained by MGPSO. In 2014, a work [123] was done to analyze the complexity of

Table 2.3: Run time of different simulations (s).

Systems	6-bus	24-bus	46-bus	79-bus
BARON $\epsilon_r = 0.1$	1	2	972	28800*
BARON $\epsilon_r = 0.01$	1	4	5347	28800*
BARON $\epsilon_r = 0.001$	1	4	6418	28800*

* Best solution has not been found by 28800s.

TEP with the Branch-and-Reduce Optimization Navigator (BARON). The result is given in Table 2.3, where ϵ_r is the relative termination tolerance. It was concluded that BARON converged very fast for the 6-bus and 24-bus systems, however, the execution time increased sharply for the 46-bus system, and it even could not get the historical best solution after a computation time of 8 hours for the 79-bus system.

Table 2.4 illustrates the main result of commercial software Lingo 11.0 and MGPSO programmed with Matlab 2015b running on the same desktop; results reported by Modified PSO (MPSO) [48] are also included for referencing rather than comparison since the simulation platform is different. Different versions of MPSO were reported in [48], where the fastest one with convergence rate over 50% and 100% are chosen for referencing. For the 6-bus and 24-bus systems, MPSO is slower than BARON, but it is a little bit faster for the 46-bus system. The convergence rate of these systems can reach 100% for 10 times of trial. No result has been reported by MPSO for the large-scale 79-bus system.

The default setting of nonlinear programming solver Lingo is adopted in this work for fair comparison. The precision of elapsed time for Lingo is 1s. A total number of 30 times has been tested for each instance. It can be concluded that Lingo performs better than MPSO on the execution time; however, the convergence rate of Lingo is relatively low for medium- and large-scale systems, the reason is that the option 'global solver' is disabled in the default setting. Although the Lingo is very fast with the default setting, the global optimal solution cannot be guaranteed, which is due to the non-convex and multi-modal feature of TEP. On the other hand, the global optimal can be always reached if 'global solver' is enabled, at the expense of much longer solution time as its low efficiency. For example, as shown in Table 2.4, the average solution time of the 24-bus system with Lingo on default setting is only 26.900s, but it turns to be 09:24:11 (33851s, 1258 times longer) after calling the 'global solver'. The same solver has also been implemented on 46-bus and 79-bus systems, but the solution process cannot terminate after a running of 2 days.

A detailed comparison is conducted between MGPSO and Lingo. For the 6-bus system, both methods converge very fast and accurate, but when it comes to the 24-bus system, the global best of 218 Million US$ (MUS) is not reachable for some trials of Lingo, even the run-

Table 2.4: Summary of results for the case studies.

Systems	Alg.	Cost (×1,000,000 US $)			Convergence		Time (s)			Iterations
		Best	Worst	Avg.	Trial	Sucd.	Min.	Max.	Avg.	
6-bus	MPSO[1]	0.200	0.231	0.203	10	9	2.750	3.203	2.845	≤200
	MPSO[2]	0.200	0.200	0.200	10	10	10.672	10.828	10.733	≤400
	LINGO	0.200	0.200	0.200	30	30	<1.000	<1.000	<1.000	——
	MGPSO	0.200	0.200	0.200	30	30	0.139	0.232	0.175	48∼57
24-bus	MPSO[1]	218.000	284.000	232.800	10	7	——	——	26.050	126∼245
	MPSO[2]	218.000	218.000	218.000	10	10	——	——	14.884	173∼342
	LINGO	218.000	243.000	227.200	30	19	16.000	38.000	26.900	——
	MGPSO	218.000	218.000	218.000	30	30	5.780	9.581	6.927	192∼208
46-bus	MPSO[1]	154.420	166.040	158.810	10	6	279.700	1,146.300	——	≤1500
	MPSO[2]	154.420	154.420	154.420	10	10	633.140	1,170.300	816.270	≤2500
	LINGO	154.420	164.752	158.597	30	11	200.000	1,616.000	644.570	——
	MGPSO	154.420	154.420	154.420	30	30	79.142	113.800	95.070	375∼394
79-bus	LINGO	431.900	478.500	458.207	30	15	2,279.000	65,594.000	12,661.200	——
	MGPSO	457.800	457.800	457.800	30	30	876.789	1,096.293	1,014.784	721∼755
118-bus	LINGO	915.800	967.600	933.138	30	13	2,452.000	72,384.000	14,211.700	——
	MGPSO	929.400	929.400	929.400	30	30	926.813	1,259.915	1,110.269	742∼783

—— : Data is not been reported by the literature.

[1] and [2]: The fastest version generated from [48] with convergence rate over 50% and 100%;

Table 2.5: Solutions and costs for 79-bus system ($\times$1,000,000 US \$).

Solution	Cost
$n_{18-19}=1, n_{21-20}=1, n_{21-23}=1, n_{23-25}=1, n_{24-09}=2, n_{28-33}=1,$ $n_{28-60}=1, n_{30-29}=1, n_{34-56}=1, n_{35-38}=1, n_{38-41}=1, n_{40-56}=1,$ $n_{40-72}=1, n_{48-51}=1, n_{58-59}=1, n_{59-53}=1, n_{59-67}=1, n_{62-61}=2,$ $n_{62-64}=1, n_{63-64}=1, n_{64-65}=1, n_{69-72}=1.$	424.800 [131]
$n_{18-19}=1, n_{21-20}=1, n_{21-23}=1, n_{24-09}=2, n_{25-60}=2, n_{30-29}=1,$ $n_{34-56}=2, n_{34-64}=1, n_{35-38}=1, n_{38-41}=2, n_{40-41}=1, n_{40-56}=2,$ $n_{40-72}=1, n_{48-51}=1, n_{48-71}=1, n_{58-59}=1, n_{59-53}=1, n_{59-67}=1,$ $n_{63-64}=1, n_{64-65}=1, n_{69-72}=2.$	454.100 [130]
$n_{18-19}=1, n_{21-20}=1, n_{21-23}=1, n_{24-09}=2, n_{25-26}=2, n_{26-29}=1,$ $n_{29-31}=1, n_{34-56}=1, n_{35-55}=1, n_{38-41}=1, n_{40-56}=1, n_{48-51}=1,$ $n_{58-59}=1, n_{59-53}=1, n_{59-67}=1, n_{62-61}=2, n_{62-64}=1, n_{64-65}=1.$	457.800
$n_{18-19}=1, n_{21-20}=3, n_{21-23}=1, n_{23-25}=1, n_{24-09}=2, n_{28-33}=1,$ $n_{28-60}=1, n_{30-29}=1, n_{34-56}=1, n_{35-38}=1, n_{38-41}=1, n_{40-56}=1,$ $n_{40-72}=1, n_{48-51}=1, n_{58-59}=1, n_{59-53}=1, n_{59-67}=1, n_{62-61}=2,$ $n_{62-64}=1, n_{64-65}=1, n_{69-72}=1.$	431.900

ning time is longer than MGPSO, for which the convergence rate is 100%. MPSO, Lingo, and MGPSO get the same global optimal with [133, 134] for the 46-bus system of 154.42 MUS with a probability of 100%, 63.33%, and 100%, respectively. For the 79- and 118-bus system, no common acceptable global optimal has been reported, thus the best result gained by MGPSO of 457.8 and 929.4 MUS are regarded as the criterion for convergence judgment. As shown in Table 2.4, Lingo performs better or no worse than MGPSO for 15 and 13 times; however, the average cost is slightly higher and running time is more than 12 times longer.

For the 79-bus system, a solution with the cost of 424.8 MUS was reported by [131], however, 9.562 MW loss of load existed on bus #30. Interestingly, [134] also reported a solution of 444.49 MUS, which was indicated by the same author in [133] (where they updated the result to be 478.99 MUS without loss of load) that a total loss of load of about 37 MW was presented. Additionally, a configuration with a cost of 454.1 MUS was illustrated by [130] without loss of load. In this work, a new result of 431.9 MUS without loss of load has been generated. Table 2.5 shows the details of each solution, where the solution with a cost of 457.8MUS is also explicated.

2.2.3.4 Speedup Analysis

In order to do the numerical speedup analysis, the average execution time for Lingo to solve the 6-bus system in Table 2.4 is approximately assumed to be 0.3s, therefore the

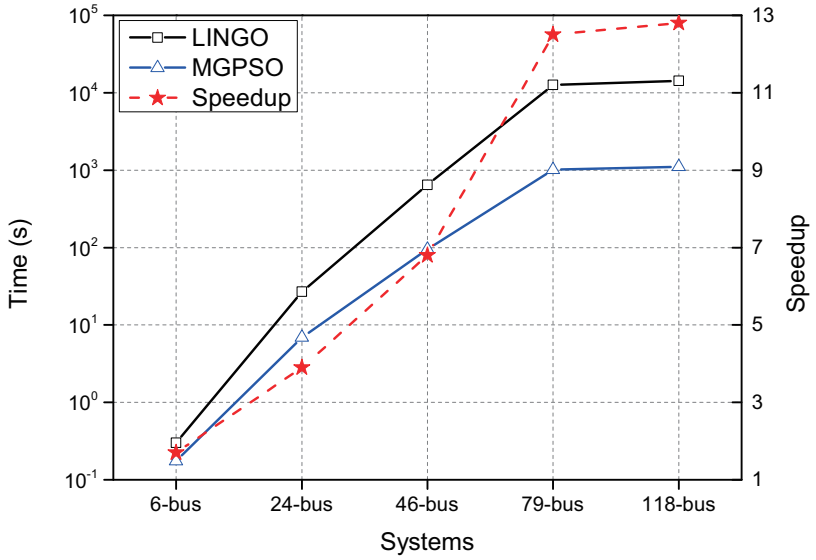


Figure 2.6: Speedup analysis between MGPSO and Lingo 11.0.

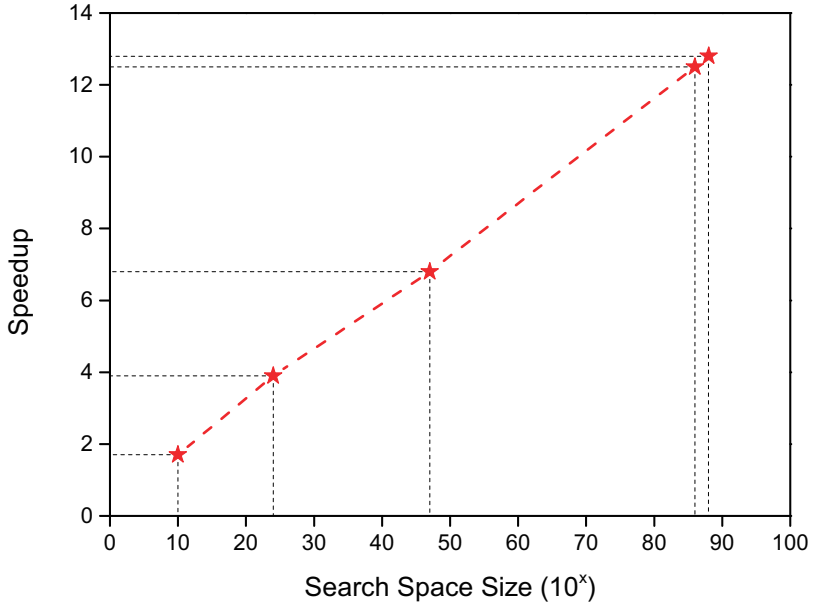


Figure 2.7: Relationship between the speedup and the search space size.

speedup for MGPSO over Lingo is $\times 1.7$. For the other systems, a speedup of $\times 3.9$, $\times 6.8$, $\times 12.5$, and $\times 12.8$ is also achieved respectively, which is illustrated in Fig. 2.6. The dashed line from 6-bus to 79-bus system indicates that the slope goes higher as system scale increases, i.e., the speedup is higher for larger systems. The suddenly flattened trend from 79-bus to 118-bus system is due to their similar search space size shown in Table 2.1. Actually, the speedup has a linear relationship (shown in Fig. 2.7) with the search space size.

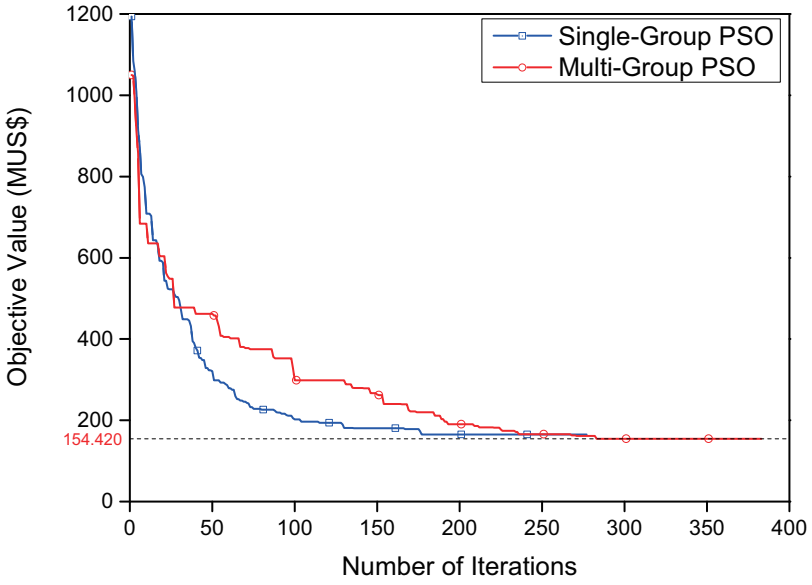


Figure 2.8: Convergence characteristic of single- and multi-group PSO for 46-bus system.

2.2.3.5 Performance Evaluation of Multi-Group Co-evolution

In order to identify the performance improvement brought by multi-group co-evolution, both single- and multi-group PSO have been implemented. The 46-bus system was determined as the target testbed due to its proper difficulty. Fig. 2.8 depicts the behavior of convergence for both algorithms. It can be noticed that the multi-group co-evolution strategy brings two influences: 1) compared with single-group PSO, the MGPSO converges slower and the solution time is also longer; 2) the quality of final solution obtained by MGPSO is better.

2.2.3.6 Performance Evaluation of Initialization Procedure

The performance evaluation of initialization procedure cannot be distinguished with only one execution since the random sequence has been involved. After 30 times of experiment on the 46-bus system, MGPSO with and without the Sobol sequence gains a convergence rate of 100% and 91.4% respectively to the global optimal.

2.2.3.7 Performance Evaluation of LU Decomposition

The LU decomposition does not affect the convergence characteristic; it impacts the reduction of solution time of fitness evaluation at each iteration. As illustrated in Table 2.4, the average execution time for MGPSO is 95.070s. In this section, two more experiments are carried out: 1) if the LU decomposition is replaced by the matrix inverse process, the so-

lution time will increase to be 126.941s; 2) if the network analysis shown in Fig. 2.5 is also eliminated, an average time of 180.107s should be required to finish the solution process.

2.3 Security Constrained Transmission Expansion Planning

In this section, the security criteria are considered for TEP formulation, leading to the SCTEP problem. Instead of the MINLP formulated in the above section with DC power flow, the disjunctive model is employed for SCTEP, resulting in a MILP in section 2.3.1. Decomposition algorithms are employed for problem solution in section 2.3.2 due to large numbers of decision variables and constraints, including BD, BCBD, and acceleration strategies. Finally, computational experiments are implemented in section 2.3.3 to perform detailed sensitivity analysis and performance evaluation.

2.3.1 Problem Formulation

2.3.1.1 Disjunctive Model

Derived from the classical DC power flow model, the disjunctive model is formulated by representing the integer decision variables with binary decision variables, and eliminating the nonlinear property by the utilization of linearizing constant (big-M). Although the linearization process introduces large numbers of decision variables and constraints, it is still widely employed for the solution of SCTEP since the whole characteristic of DC power flow model is maintained. The disjunctive model is given as [10, 44, 56, 58, 59]:

$$\min_{n_{ij}^k,\, r_i^{(s)},\, f_{ij}^{0(s)},\, f_{ij}^{k(s)},\, g_i^{(s)},\, \theta_i^{(s)}} \quad \sum_{k=1}^{K} \sum_{(i,j)\in\mathcal{C}} c_{ij} n_{ij}^k + \sum_{s=1}^{|\mathcal{S}|} \left(P \sum_{i=1}^{|\mathcal{N}_b|} r_i^{(s)} \right), \tag{2.26}$$

$$s.t. \quad n_{ij}^{k+1} \le n_{ij}^k,\, k = 1,...,K-1, \qquad ij \in \mathcal{C} \tag{2.27}$$

$$\sum_{ij\in\mathcal{E}} f_{ij}^{0(s)} + \sum_{k=1}^{K} \sum_{ij\in\mathcal{C}} f_{ij}^{k(s)} + g_i^{(s)} + r_i^{(s)} = d_i, \qquad i \in \mathcal{N}_b \tag{2.28}$$

$$f_{ij}^{0(s)} - \gamma_{ij} n_{ij}^{0(s)} \left(\theta_i^{(s)} - \theta_j^{(s)} \right) = 0, \qquad ij \in \mathcal{E} \tag{2.29}$$

$$|f_{ij}^{k(s)} - \gamma_{ij} \left(\theta_i^{(s)} - \theta_j^{(s)} \right)| \le M \left(1 - n_{ij}^k \right), \qquad ij \in \mathcal{C} \tag{2.30}$$

$$|f_{ij}^{0(s)}| \le \bar{f}_{ij} n_{ij}^{0(s)}, \qquad ij \in \mathcal{E} \tag{2.31}$$

$$|f_{ij}^{k(s)}| \le \bar{f}_{ij} n_{ij}^k, \qquad ij \in \mathcal{C} \tag{2.32}$$

$$0 \le g_i^{(s)} \le \bar{g}_i, \qquad i \in \mathcal{N}_b \tag{2.33}$$

$$0 \le r_i^{(s)} \le d_i, \qquad i \in \mathcal{N}_b \tag{2.34}$$

$$n_{ij}^k \in \{0, 1\},\, k = 1,...,K.$$

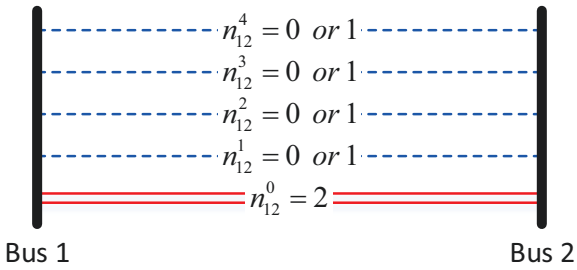


Figure 2.9: Initiated and candidate circuits on corridor $1-2$.

The objective function (2.26) comprises of construction cost and load curtailment penalty; constraint (2.27) is a valid inequality strategy utilized to refine the solution space by eliminating equivalent solutions, which will be explained in section 2.3.2.3; constraint (2.28) is the load balance requirement for each bus, i.e. Kirchhoff's current law; constraints (2.29) and (2.30) represent the Kirchhoff's voltage law for existing and candidate circuits; power flow is limited by (2.31) and (2.32); and finally, the amount of generation and load curtailment are restricted by (2.33) and (2.34). The configuration of parameter n_{ij}^0 and binary decision variables $n_{ij}^k (k = 1,...,K)$ are exemplified by Fig. 2.9 on corridor $1-2$.

Scenario set $\mathcal{S}$ related to $N-1$ transmission line outage can be defined as either a full version for the convenience of formulation,

$$\mathcal{S} = \mathcal{C}, \tag{2.35}$$

or a refined one to save computational time.

$$\mathcal{S} = \{ij | n_{ij}^0 + \sum_{k=1}^{K} n_{ij}^k > 0, ij \in \mathcal{C}\}. \tag{2.36}$$

For a given solution, it is sufficient only when it can withstand the loss of any $ij \in \mathcal{S}$. If (2.35) is utilized, the outage of all $ij \in \mathcal{C}$ should be checked; on the other hand, (2.36) only exams those corridors with original or newly built circuits.

All the components in (2.26) – (2.34) with the superscript $^{(s)}$ are scenario dependent decision variables, which will be valued automatically in the process of problem solving, except for $n_{ij}^{0(s)}$, which is a parameter required to describe each scenario that needs to be predefined. Suppose $i_s j_s$ is the corresponding unavailable corridor in scenario s, then the process of $n_{ij}^{0(s)}$ assignment as well as contingency construction can be given by **Algorithm 2.4**. The main logic of **Algorithm 2.4** lies in the fact that the paralleled circuits are equivalent with each other. When a contingency on corridor $i_s j_s$ needs to be formulated, we first check whether there are any existing initial circuits; if yes, then model this scenario by reducing one initial circuit (step 5); otherwise, set the power flow of the first parallel link to

Algorithm 2.4 Contingency Construction

1: **for** Each considered contingency s **do**
2: **for** Each candidate and existing circuit ij **do**
3: **if** Considered circuit ij is corridor i_sj_s **then**
4: **if** $n^0_{i_sj_s} > 0$ **then**
5: Set $n^{0(s)}_{ij} = n^0_{i_sj_s} - 1$.
6: **else**
7: Replace constraint (2.30) when $k = 1$ by $f^{1(s)}_{ij} = 0$.
8: **end if**
9: **else**
10: Set $n^{0(s)}_{ij} = n^0_{ij}$.
11: **end if**
12: **end for**
13: **end for**

be 0 (step 7), which means $n^1_{i_sj_s}$ is withdrawn. Obviously, step 7 satisfies the circumstance that no power flow needs to be taken by i_sj_s; otherwise, at least 1 parallel link needs to be built; however it must start from $n^2_{i_sj_s} = 1$ (where $n^1_{i_sj_s}$ still needs to be valued as 1 according to valid inequality (2.27)) since $f^{1(s)}_{ij} = 0$ due to step 7, which means the cost of $n^1_{i_sj_s}$ has been counted although it provides no contribution to the power flow. The above algorithm is suitable for both definitions of $\mathcal{S}$ given in (2.35) and (2.36).

2.3.1.2 Stochastic Programming

The MILP disjunctive model of SCTEP is usually treated as the two-stage stochastic programming problem [56–59], which can be formulated as follows:

$$
\begin{array}{lccccccc}
\min\limits_{\boldsymbol{x},\boldsymbol{y}_s} & \boldsymbol{c}^T\boldsymbol{x} & + & P\boldsymbol{q}_1^T\boldsymbol{y}_1 + P\boldsymbol{q}_2^T\boldsymbol{y}_2 \cdots + P\boldsymbol{q}_s^T\boldsymbol{y}_s & & & (2.37)\\
s.t. & \boldsymbol{A}\boldsymbol{x} & & & & \leq & \boldsymbol{b}, \\
 & \boldsymbol{T}_1\boldsymbol{x} & + & \boldsymbol{W}_1\boldsymbol{y}_1 & & \leq & \boldsymbol{h}_1, \\
 & \boldsymbol{T}_2\boldsymbol{x} & + & \boldsymbol{W}_2\boldsymbol{y}_2 & & \leq & \boldsymbol{h}_2, \\
 & \vdots & + & \cdots & & \leq & \vdots \\
 & \boldsymbol{T}_s\boldsymbol{x} & + & \boldsymbol{W}_s\boldsymbol{y}_s & & \leq & \boldsymbol{h}_s, \\
 & & & \boldsymbol{x} \in \boldsymbol{X},\ \boldsymbol{y}_1 \geq 0,\ \cdots,\ \boldsymbol{y}_s \geq 0. & & &
\end{array}
$$

where $\boldsymbol{c}$ and $\boldsymbol{q}_s$ are the cost vectors; $\boldsymbol{x}$ is a vector that denotes the integer decision variables restricted by a set of $\boldsymbol{X}$; $\boldsymbol{y}_s$ are the continuous variables for each scenario s; $\boldsymbol{A}$, $\boldsymbol{b}$, $\boldsymbol{T}_s$, $\boldsymbol{W}_s$, and $\boldsymbol{h}_s$ are coefficient matrices and vectors. The detailed configurations of decision variables $\boldsymbol{x}$ and $\boldsymbol{y}_s$ are given in Table 2.6. The structures of coefficient matrices and vectors can be determined accordingly based on Table 2.6 and (2.26) – (2.34).

Table 2.6: Definition of decision variables $\boldsymbol{x}$ and $\boldsymbol{y}_s$.

Vars.	Properties	Configurations
$\boldsymbol{x}$	Bin.	$\left\{\left[n_{ij}^{k}\right]_{ij\in\mathcal{C},\ k=1\ldots K}\right\}$
$\boldsymbol{y}_1$	Cont.	$\left\{\left[f_{ij}^{0(1)}\right]_{ij\in\mathcal{E}}, \left[f_{ij}^{k(1)}\right]_{ij\in\mathcal{C},\ k=1\ldots K}, \left[g_{i}^{(1)}\right]_{i\in\mathcal{N}_b}, \left[r_{i}^{(1)}\right]_{i\in\mathcal{N}_b}, \left[\theta_{i}^{(1)}\right]_{i\in\mathcal{N}_b}\right\}$
$\boldsymbol{y}_2$	Cont.	$\left\{\left[f_{ij}^{0(2)}\right]_{ij\in\mathcal{E}}, \left[f_{ij}^{k(2)}\right]_{ij\in\mathcal{C},\ k=1\ldots K}, \left[g_{i}^{(2)}\right]_{i\in\mathcal{N}_b}, \left[r_{i}^{(2)}\right]_{i\in\mathcal{N}_b}, \left[\theta_{i}^{(2)}\right]_{i\in\mathcal{N}_b}\right\}$
$\cdots$	Cont.	$\cdots$
$\boldsymbol{y}_s$	Cont.	$\left\{\left[f_{ij}^{0(s)}\right]_{ij\in\mathcal{E}}, \left[f_{ij}^{k(s)}\right]_{ij\in\mathcal{C},\ k=1\ldots K}, \left[g_{i}^{(s)}\right]_{i\in\mathcal{N}_b}, \left[r_{i}^{(s)}\right]_{i\in\mathcal{N}_b}, \left[\theta_{i}^{(s)}\right]_{i\in\mathcal{N}_b}\right\}$

It is relatively straightforward to translate the MILP disjunctive model (2.26) into stochastic programming (2.37) by converting constraints (2.28) – (2.34) into $\boldsymbol{T}_s\boldsymbol{x} + \boldsymbol{W}_s\boldsymbol{y}_s \leq \boldsymbol{h}_s$ for each scenario s and keeping the other constraints and objective function the same. The solution process for stochastic programming is usually implemented in an iterative manner. At each iteration, the master problem (first stage) is solved to obtain a temporary integer solution, which will be verified by subproblems (scenarios, referred as the second stage), where feasibility or optimality cuts may be generated according to specified rules, and then added to the master problem for the next iteration solution. The procedure terminates if no cuts can be extracted from the second stage. During the above course, the non-anticipativity constraint [135] of stochastic programming prescribes that all the second stage subproblems should receive the same temporary solution from the first stage, which means each scenario is independent with the temporary solution. In this work, scenario s is determined by **Algorithm 2.4**, where the intermediate solution $\bar{n}_{ij}^{k}$ is not related, i.e., non-anticipativity constraints are maintained.

It should be noted that, from the viewpoint of practical operation, if any loss of load $r_i^{(s)}$ is positive, the solution is invalid; however, mathematically speaking, with the introduction of $r_i^{(s)}$, the system is always feasible, i.e., any solution of $\bar{n}_{ij}^{k}$ will not violate any constraints since there always has at least a scheme of $r_i^{(s)} = d_i$ to compensate the load imbalance in (2.28). Therefore, the generated two-stage stochastic programming is a complete recourse problem, i.e., all the first stage solutions are feasible in the second stage.

2.3.2 Solution Methodology

2.3.2.1 Benders Decomposition

BD separates the original problem (2.37) into Master Problem [MP] and Sub-Problem [SP] according to binary and continuous decision variables. In addition, the dual problem of [SP] should be generated and marked as [DP].

$$\text{[MP]} \quad \min_{\boldsymbol{x}} \quad \boldsymbol{c}^T\boldsymbol{x} + Q \tag{2.38}$$

$$s.t. \quad \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \in \boldsymbol{X}, \tag{2.39}$$

$$\text{[SP]} \quad \min_{\boldsymbol{y}_s} \quad \boldsymbol{q}_s^T\boldsymbol{y}_s \tag{2.40}$$

$$s.t. \quad \boldsymbol{W}_s\boldsymbol{y}_s \leq \boldsymbol{h}_s - \boldsymbol{T}_s\bar{\boldsymbol{x}}, \boldsymbol{y}_s \geq 0, s \in \mathcal{S}. \tag{2.41}$$

$$\text{[DP]} \quad \max_{\boldsymbol{u}_s} \quad (\boldsymbol{h}_s - \boldsymbol{T}_s\bar{\boldsymbol{x}})^T\boldsymbol{u}_s \tag{2.42}$$

$$s.t. \quad \boldsymbol{W}_s^T\boldsymbol{u}_s \leq \boldsymbol{q}_s, \boldsymbol{u}_s \leq 0, s \in \mathcal{S}, \tag{2.43}$$

where $Q = \sum_{s\in\mathcal{S}} p_s Q_s$ is the weighted sum of objective values from each subproblem for scenario s; $\bar{\boldsymbol{x}}$ is the temporary solution of [MP], $\boldsymbol{u}_s$ is the dual values for constraints in [SP].

As this is a complete recourse problem, [SP] is always feasible, thus [DP] is bounded, and the objective function value of [DP] provides a valid lower bound for [SP] according to dual theory, therefore an optimality cut can be generated:

$$Q_s \geq (\boldsymbol{h}_s - \boldsymbol{T}_s\boldsymbol{x})^T\bar{\boldsymbol{u}}_s, s \in \mathcal{S}, \tag{2.44}$$

where $\bar{\boldsymbol{u}}_s$ is the optimal solution of [DP]. Since the feasibility of [SP] is always valid, feasibility cuts are not required in this problem. After the adding of optimality cut, the [MP] is evolved as:

$$\text{[MP]} \quad \min_{\boldsymbol{x}} \quad \boldsymbol{c}^T\boldsymbol{x} + Q \tag{2.45}$$

$$s.t. \quad \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \in \boldsymbol{X}, \tag{2.46}$$

$$Q \geq \sum_{s\in\mathcal{S}} p_s(\boldsymbol{h}_s - \boldsymbol{T}_s\boldsymbol{x})^T\bar{\boldsymbol{u}}_s^i, i = 1...N \tag{2.47}$$

where N is the current iteration number.

In each iteration, the lower bound LB is the objective value of [MP], and the upper bound UB is determined by the objective value of [DP], i.e., $UB' = \boldsymbol{c}^T\bar{\boldsymbol{x}} + \sum_{s\in\mathcal{S}} p_s(\boldsymbol{h}_s - \boldsymbol{T}_s\bar{\boldsymbol{x}})^T\bar{\boldsymbol{u}}_s$. The solution process of BD is given in **Algorithm 2.5**. After initialization, the [MP] is first solved, with temporary solution $\bar{\boldsymbol{x}}$ and objective value LB' obtained. Update LB into LB' if $LB' > LB$. Based on $\bar{\boldsymbol{x}}$, the [DP] for each scenario s can be solved to generate optimality cuts (2.47) and UB'. Then, cuts are added into [MP] and UB is updated

Algorithm 2.5 Benders Decomposition (simplified version)

1: Set $\epsilon = 10^{-6}$, $LB = -\infty$ and $UB = +\infty$.
2: **while** $|UB - LB| > \epsilon$ **do**
3: Solve [MP] with all generated cuts to get a solution $\bar{\boldsymbol{x}}$ with objective value LB'.
4: **if** $LB' > LB$ **then**
5: Set $LB = LB'$.
6: **end if**
7: Solve [DP] with $\bar{\boldsymbol{x}}$ for each scenario s, denote the optimal solution as $\bar{\boldsymbol{u}}_s$.
8: Calculate $UB' = \boldsymbol{c}^T\bar{\boldsymbol{x}} + \sum_{s\in\mathcal{S}} p_s(\boldsymbol{h}_s - \boldsymbol{T}_s\bar{\boldsymbol{x}})^T\bar{\boldsymbol{u}}_s$.
9: Generate cut (2.47) based on $\bar{\boldsymbol{u}}_s$, and add it into [MP].
10: **if** $UB' < UB$ **then**
11: Set $UB = UB'$.
12: **end if**
13: **end while**

into UB' if $UB' < UB$. The above iterative process terminates if $|UB - LB| \leq \epsilon$. The transferred data between [MP] and [DP] is their optimal solution $\bar{\boldsymbol{x}}$ and $\bar{\boldsymbol{u}}_s$. It should be noted that **Algorithm 2.5** is a simplified version of classical BD since the feasibility cuts are omitted. For more details of BD implementation on the power system, please refer to [136]. An illustrative flowchart of implementation framework is given in Fig. 2.10. Nevertheless, repeatedly solving [MP] to optimality, adding a cut and re-solving it can be very expensive in terms of computational resources.

2.3.2.2 Branch-and-Cut Benders Decomposition

In order to relieve the computational burden of [MP], a Branch-and-check strategy was advocated in [137], where only one MILP search tree of [MP] was built and maintained, i.e., [MP] is solved into optimality by only once, whereas all the efforts spent on each node of the tree are solving the LP relaxed [MP], which is much easier and faster compared with MILP solution. In spirit of this strategy, we integrated BD into the B&C framework provided by ILOG-Cplex concert technology, resulting in the BCBD algorithm. The step by step implementation process is illustrated by **Algorithm 2.6**. It is obvious that the key steps of **Algorithm 2.5** are line 3 and lines 7 – 9 which are all represented and highlighted in **Algorithm 2.6**. The solution framework of BCBD is illustrated in Fig. 2.11. It can be seen that only LP is addressed in both [MP] and [DP].

Large numbers of constraints will be involved in large-scale problems, however, many of them are redundant or at least not binding near the optimal solution, and any of them can be ruled out without prior information. One proper method to handle these constraints is to set them as lazy constraints and put them into a pool in Cplex. When a solution is generated, the solver will check if any lazy constraints are violated and, if so, adds them

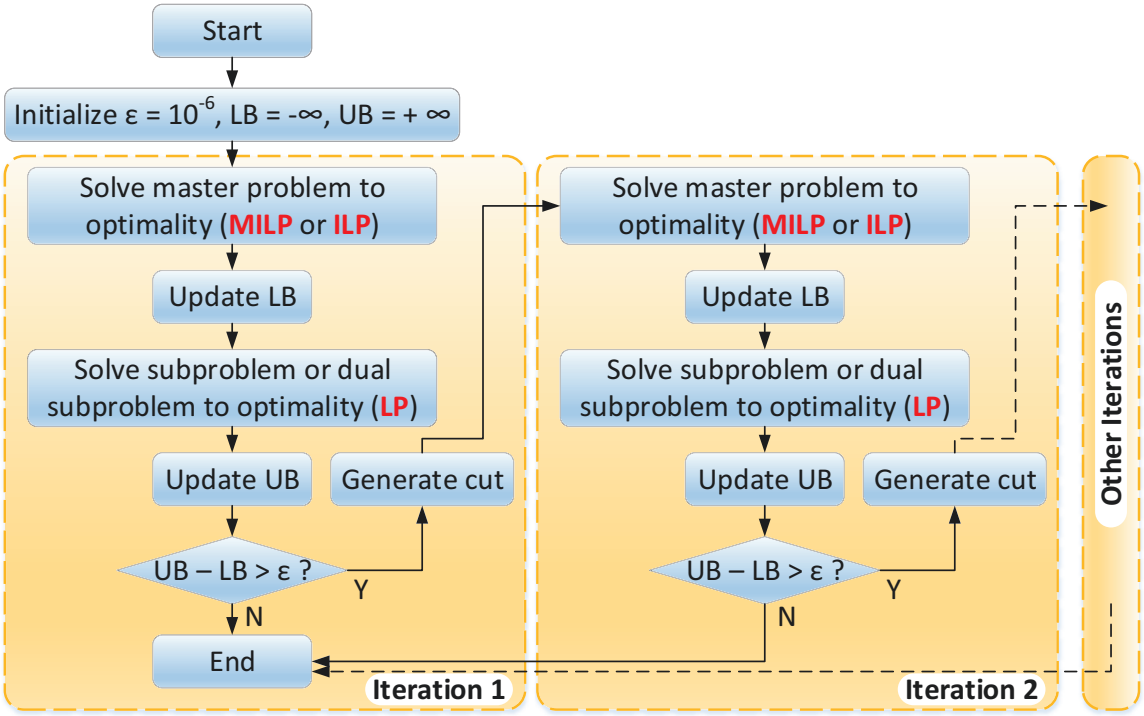


Figure 2.10: Flowchart of BD within classical implementation framework.

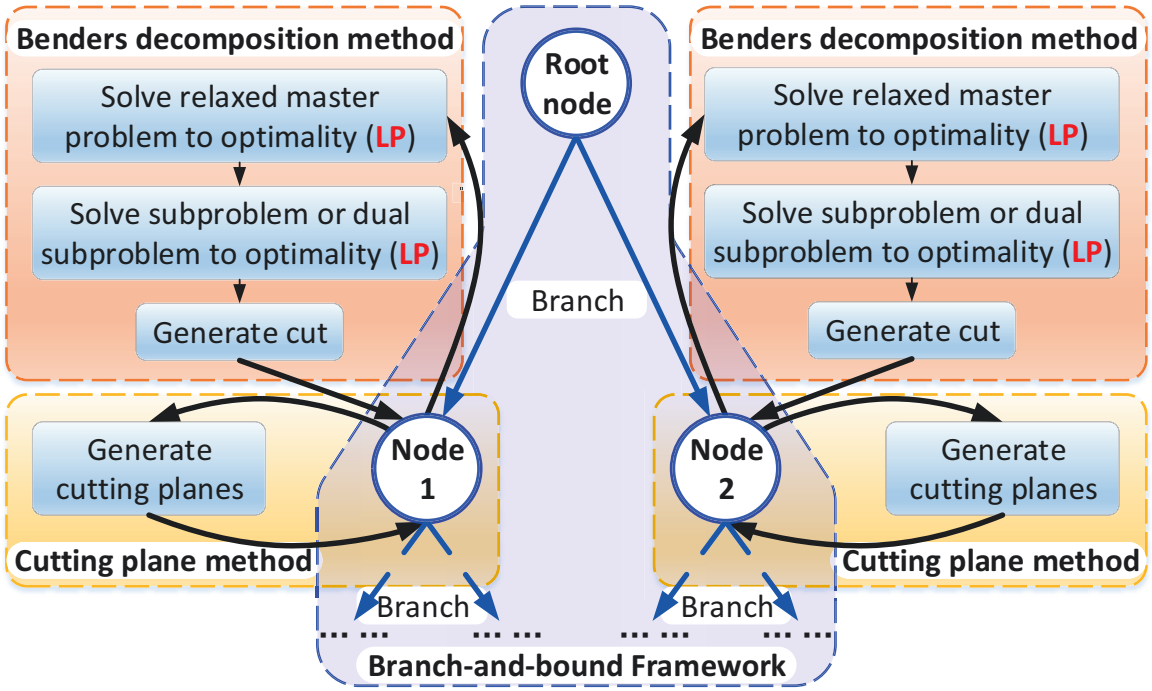


Figure 2.11: Flowchart of BD within B&C implementation framework.

Algorithm 2.6 BCBD Algorithm

1: Add the original [MP] into tree L, set final solution $\boldsymbol{x}^* = null$ and value $v^* = +\infty$.
2: **while** L is not empty **do**
3: Select a node [MP] from L.
4: Solve the LP relaxation of [MP] to obtain an optimal solution $\bar{\boldsymbol{x}}$ with objective value v. (*corresponding to line 3 in **Algorithm 2.5***)
5: **if** LP is infeasible **then**
6: Prune the node.
7: **else if** $v \geq v^*$ **then**
8: Prune the node.
9: **else if** $\bar{\boldsymbol{x}}$ is integer **then**
10: Solve [DP] based on $\bar{\boldsymbol{x}}$ and generate Benders cuts. (*corresponding to lines 7 – 9 in **Algorithm 2.5***)
11: **if** No cuts generated **then**
12: Update $v^* = v$ and $\boldsymbol{x}^* = \bar{\boldsymbol{x}}$, prune the node.
13: **else**
14: Add the cuts to the LP relaxation and return to line 4.
15: **end if**
16: **else**
17: **if** A candidate solution $\bar{\boldsymbol{x}}'$ is found **then**
18: Search for cutting planes that are violated by $\bar{\boldsymbol{x}}'$.
19: Solve [DP] based on $\bar{\boldsymbol{x}}'$ and generate Benders cuts. (*corresponding to lines 7 – 9 in **Algorithm 2.5***)
20: If any cutting planes or Benders cuts are found, add them to the LP relaxation and return to line 4.
21: **end if**
22: Choose one non-integral variable from $\bar{\boldsymbol{x}}$ to branch, create two nodes and add them to L.
23: **end if**
24: **end while**
25: **return** Final solution $\boldsymbol{x}^*$ and value v^*.

to the active set. Lazy constraints that were previously added but have not been binding for a while will be returned to the pool. In **Algorithm 2.6**, the lazy constraint with callback function is utilized, which is guaranteed to be checked every time Cplex B&C framework finds a candidate solution (see line 17), regardless of how the candidate is found (such as, node LP solution, rounding, and various heuristics).

2.3.2.3 Acceleration Strategies

To improve the solution performance, four acceleration strategies are employed in this work. They are beneficial from different aspects: generating high-quality initiate points, increasing the number of cuts, restricting the solution space, and obtaining tighter cuts. Their performance on classical BD framework has been reported in [138]; however, im-

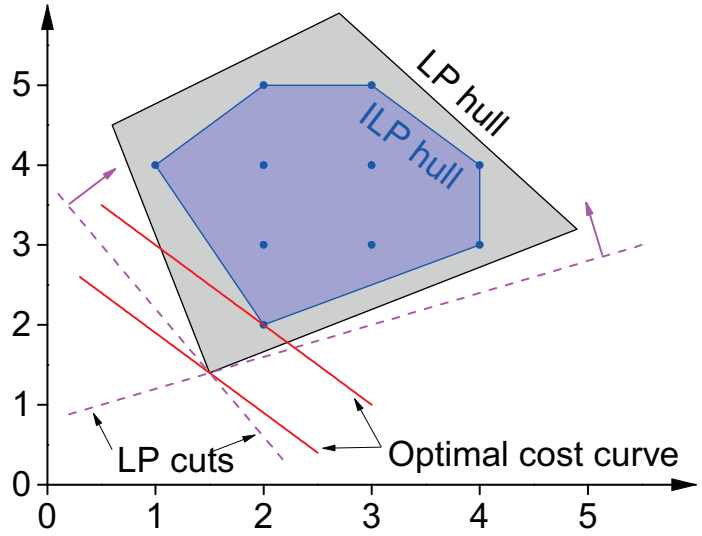


Figure 2.12: ILP hull versus LP hull.

plementation on the developed BCBD framework has never been revealed before. For simplicity, only the first method is depicted in detail.

i. Two-phase Method

As shown in Fig. 2.12, the convex hull of the feasible region of the MILP is always contained within the LP relaxation, thus all added LP cuts are valid for MILP, which leads to the two-phase method [138]. In the two-phase method, the MILP problem is relaxed into LP and solved to optimality by BD in *Phase 1*, all the generated cuts are sent to *Phase 2* where the integrality property of MILP is considered. The general implementation framework is given in **Algorithm 2.7**. It has been proved by experiments that this method plays a major role in making the MILP start from a high-quality point (global optimal of LP relaxation), especially when the MILP has a small integrality gap.

ii. Multicut Method

Instead of returning only one cut at each iteration to [MP] as shown in (2.47) for the classical BD, a multicut strategy can be employed to enhance the convergence efficiency by generating one cut from each scenario [57, 58, 139, 140], i.e., multiple cuts are introduced for each iteration.

iii. Valid Inequality

In SCTEP, the circuits built in parallel are similar, thus there will be several equivalent optimal solutions, which may introduce complexity during the solution process. Results for the Garver 6-bus system is shown in Fig. 2.13, where the maximum number of equivalent solutions is as large as 4.70×10^{11}. Therefore, in order to save computational efforts for solution searching and make the optimal solution logically unique, valid inequality (2.27)

Algorithm 2.7 Two-Phase BCBD Algorithm

1: ***Phase 1:***
2: Remove integrality constraints on all variables.
3: Solve the problem using **Algorithm 2.5**, and keep all the generated cuts.
4: ***Phase 2:***
5: Reintroduce integrality constraints on the master problem variables.
6: Add all cuts generated from *Phase 1* into master problem, and solve the problem using **Algorithm 2.6**.

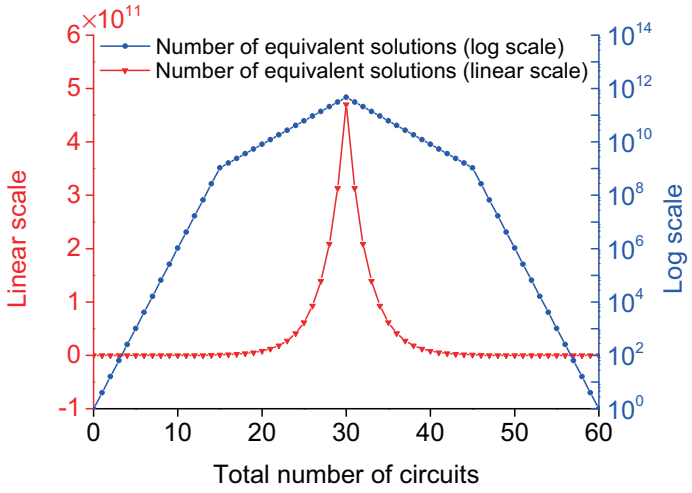


Figure 2.13: Number of equivalent solutions for the Garver 6-bus system without valid inequality.

is employed for all systems and algorithms considered in this work.

iv. Optimal Preconditioning

One of the preconditions for the optimality of SCTEP solution is $Q = 0$ in (2.45), which means no loss of load is tolerable. Since p_s is positive, then $Q = 0$ is equivalent with $Q_s = 0$ for all $s \in \mathcal{S}$. Therefore, this optimality precondition can be embedded into the Benders cut generation (2.44) by forcing $Q_s = 0$.

2.3.3 Computational Experiments

To evaluate the performance of the proposed algorithm, we implemented our method in C++ and embedded it within the ILOG-Cplex concert technology framework, based on ILOG-Cplex 12.5.1. The implementation platform is a 64-bit Windows desktop with 32GB RAM and Intel Xeon E5-2620 CPUs at 2.10GHz.

In our experiment, the Cplex MILP solver is employed for comparison. In order to achieve the best performance from the MILP solver, two key modes are investigated: (i) sequential and parallel implementation mode; and (ii) traditional and dynamic B&C

Table 2.7: Scales and complexity of considered benchmark test systems.

Items	**6-bus**	**24-bus**	**46-bus**	**118-bus**	**300-bus**
No. of buses:	6	24	46	118	300
No. of candidate branches:	15	41	79	186	411
No. of parallel circuits:	4	2	3	2	1
No. of scenarios:	15	41	79	186	411
No. of binary variables:	60	82	237	372	411
No. of continuous variables:	1,395	7,995	35,886	169,632	707,742
No. of equality constraints:	315	2,665	9,875	56,544	292,221
No. of inequality constraints:	10,125	179,867	1,248,200	16,709,868	203,718,726
Data resources	[17]	[129]	[17]	[143]	[143]

search pattern. Therefore, four solvers are identified in total: *MILP_dynamic (1 thread)*, *MILP_dynamic (24 threads)*, *MILP_traditional (1 thread)*, and *MILP_traditional (24 threads)*. All the other parameters of MILP solver are kept at their default settings, such as primal heuristics, branching variable selection, and next node selection. On the other hand, since the lazy constraint callback function does not support the dynamic search and may not be thread safe, our algorithm is only implemented in sequential mode with the traditional pattern: *BCBD_traditional (1 thread)*.

A tolerance of $\epsilon = 10^{-6}$ is employed for all experiments. All algorithms are forced to terminate after exceeding a maximum run time of 48 hours. Big-M is determined by the following equation in accordance with [59, 141], and [142],

$$M_{ij} = 2\bar{\theta}\gamma_{ij}, \tag{2.48}$$

where $\bar{\theta}$ is the maximum bus voltage angle.

2.3.3.1 The Test Bed

In order to investigate the full potential of the considered methods and algorithms, five classical benchmark systems of different sizes are employed: the Garver 6-bus system, the IEEE 24-bus test system, the South Brazilian 46-bus system, the IEEE 118-bus test system, and the IEEE 300-bus test system. An overview of the scales and complexity are illustrated in Table 2.7, as well as the available data resources.

The whole feasible region is fully considered as the input for all algorithms. It should be pointed out that, for IEEE 118- and 300-bus test systems, the maximum capacity of each line has been reduced to 40% of the capacity given originally to increase the complexity

Table 2.8: Computational results for test systems with 5 different types of methods.

Alg.	6-bus		24-bus		118-bus		46-bus		300-bus	
	T.(s)	Gap	T.(s)	Gap	T(s)	Gap	T.(h)	Gap	T.(h)	Gap
A	2.97	99.97%	2,201.6	32.75%	1,861.6	99.89%	48.0	18.5%	48.0	27.5%
B	3.52	99.97%	**246.31**	**0.00%**	1,487.1	99.89%	32.3	17.4%	48.0	18.9%
C	5.72	99.97%	1,230.7	31.99%	709.75	99.89%	48.0	30.3%	48.0	33.2%
D	5.19	99.97%	317.80	27.98%	481.33	99.89%	33.6	16.7%	48.0	20.7%
E	**0.47**	**0.00%**	1,516.1	21.07%	**62.65**	**0.00%**	**15.6**	**0.0%**	**48.0**	**5.4%**

A: MILP_dynamic (1 thread) B: MILP_dynamic (24 threads) C: MILP_traditional (1 thread)
D: MILP_traditional (24 threads) E: BCBD_traditional (1 thread)

of the problem [59]. Additionally, the original data set of [143] does not contain the price information; in this work, an assumption on the transmission line investment cost, similar to [59], is adopted:

$$c_{ij} = 1000 L_{ij} \bar{f}_{ij}, \tag{2.49}$$

where L_{ij} is the length of circuit ij.

2.3.3.2 Results

Unlike other algorithms with parameters which need to be tuned before implementation, BCBD as well as the other 4 MILP solvers are totally parameter free. The comprehensive results of the five test systems are given in Table 2.8, where the gap is defined by (2.50) and sampled when the fastest algorithm terminates.

$$Gap = (UB - LB)/UB. \tag{2.50}$$

Table 2.8 is interpreted as follows according to different systems. Detailed performance comparison between various algorithms within one system will be given in section 2.3.3.5.

i. Garver 6-bus System

Although K is valued as 4, the solution space of this small-scale system is still limited, thus all methods achieve the convergence in seconds. With the help of B&C framework and acceleration strategies, the BCBD is almost $10\times$ faster to arrive at the global optimal solution when compared with MILP solvers. Dynamic MILP algorithms significantly perform better than the traditional ones in this system.

ii. IEEE 24-bus Test System

This is a medium-scale system, all algorithms are able to achieve the global optimal solu-

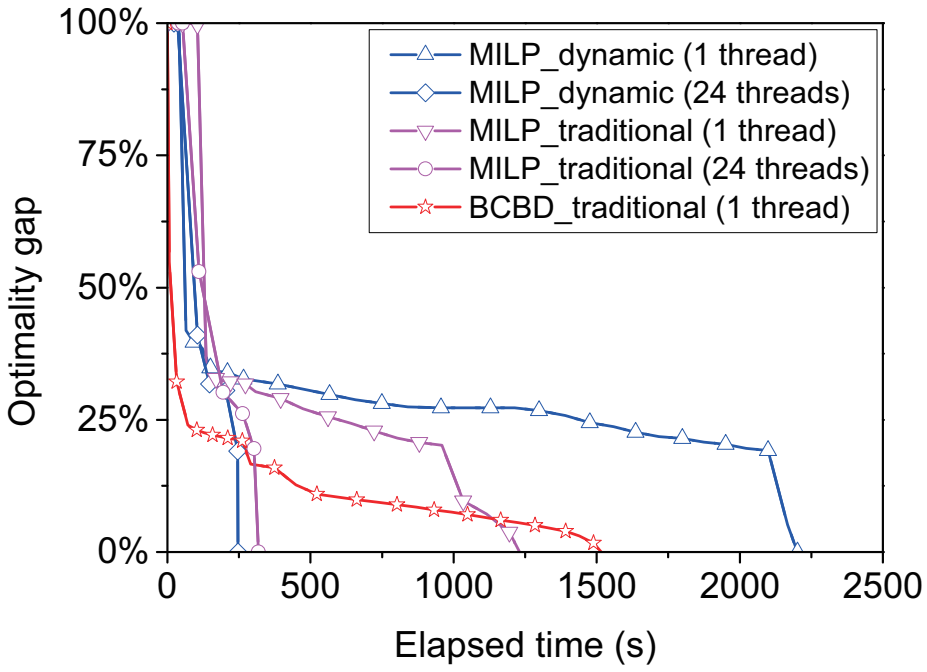


Figure 2.14: Behavior of optimality gap for the IEEE 24-bus test system.

tion within 40 minutes. It can be concluded from Table 2.8 that parallel implementation with 24 threads works much better than the sequential ones, but it is hard to distinguish which one performs better than another for the dynamic and the traditional mode. The convergence curves for different algorithms are illustrated in Fig. 2.14. BCBD converges faster than all the other four MILP solvers in the early stage, which is due to the two-phase method; however, the MILP solvers with 24 threads suddenly converged into the global optimal solution in the later stage, which is due to the built-in heuristics. Different with some efficient heuristics designed for specific problems, the built-in heuristics in Cplex are designed for a general purpose, thus their performance for SCTEP is not stable.

iii. IEEE 118-bus Test System

Although this system has a large solution space due to a large number of decision variables, it is not so much difficult when compared with IEEE 24-bus system from the aspect of solution time. The reason is that the configuration is more sufficient for the 118-bus system, i.e., the number of circuits needed to be built is limited. Different from the results of the former system, BCBD shows its advantage over the other four MILP solvers in this system, where a speedup of 29.71, 23.74, 11.33, and 7.68 is gained respectively. When BCBD terminates, an optimality gap of 99.89% is still holding for all the other methods, although two of them run in parallel with 24 threads. It can be seen from Fig. 2.15 that several MILP solvers experience a long flat at the beginning, indicating that it is very hard to find a feasible solution for this problem. Another interesting phenomenon is the sharp

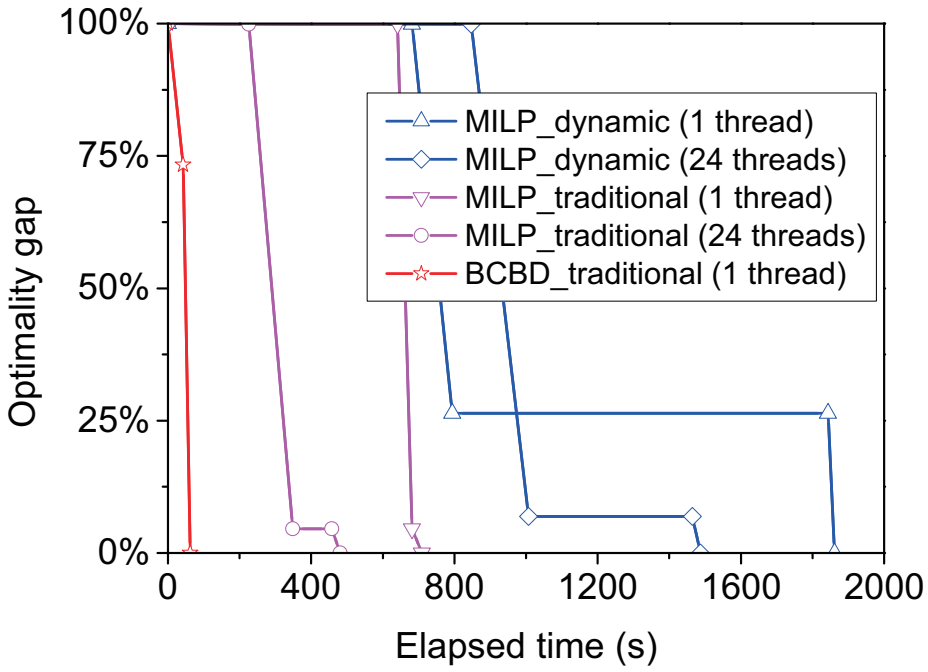


Figure 2.15: Behavior of optimality gap for the IEEE 118-bus test system.

decrease in the final stage, which is due to the special structure of this problem that it may be easy to derive the global optimal solution from lots of feasible solutions. BCBD can find a high-quality feasible solution with less effort by the help of all generated cuts from *Phase 1*; thus it performs dramatically well for this problem. This can also be observed from the fact that the convergence process of BCBD is similar with the last stages of *MILP_traditional (1 thread)*.

iv. South Brazilian 46-bus System

This is a medium-scale real system, but its solution is more difficult than the 118-bus system. The fastest algorithm BCBD requires 15.61h to meet the gap of 0.00%; in contrast, two MILP solvers could not even converge after running for 48h: a gap of 22.80% and 11.22% is still remaining, respectively. It should be noted that parallel implementation plays a major role for MILP solvers in the solution of this test system. The sequential BCBD is 16h faster than the successfully converged MILP solvers, although they are implemented in parallel with 24 threads. Fig. 2.16 demonstrates the behavior of optimality gap for the 46-bus system. It can be seen that it takes almost 16h for MILP solvers working in parallel mode to converge into a gap of 15%; however, it is just hundreds of seconds for BCBD, which is greatly due to the two-phase acceleration strategy. Both BCBD and the other two parallel MILP solvers spent nearly 16h to fulfill the rest searching process, which proves that integrating BD into B&C framework is competitive when compared with parallel MILP solvers for really tough problems.

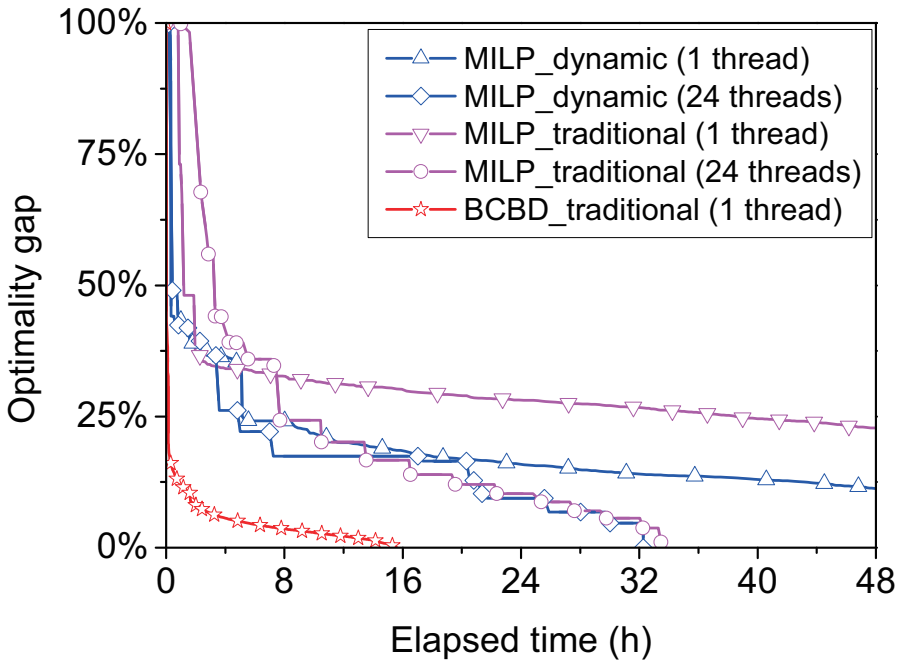


Figure 2.16: Behavior of optimality gap for the South Brazilian 46-bus system.

v. IEEE 300-bus Test System

This large-scale system provides great challenges for all five methods since no method can finish the searching within 48h. Nevertheless, the superiority of BCBD can still be identified by the finally achieved gaps. Although the last gap is only 5.41% for BCBD, it may takes tens of hours to arrive at 0.00% according to the previous experience for 46-bus system shown in Fig. 2.16. In terms of MILP solvers with larger termination gaps, requirement on the execution time and effort is much higher.

2.3.3.3 Qualitative Evaluation

In order to distinguish which pattern and mode of MILP solver performs better for SCTEP, a qualitative evaluation is introduced in Table 2.9, where the number in each column represents the performance rank of the corresponding algorithm for each problem, i.e., the fastest one ranks 1 and the slowest one ranks 5. By adding the numbers for each row, there has $8_{(rowE)} < 12_{(rowB)} < 14_{(rowD)} < 20_{(rowA)} < 21_{(rowC)}$. It can be concluded that BCBD performs better than the others across the five test systems. Two more conclusions can also be drawn for different versions of the MILP solver: (i) parallel implementation works better than sequential, since $12_{(rowB)} < 20_{(rowA)}$ and $14_{(rowD)} < 21_{(rowC)}$ in dynamic and traditional modes respectively; (ii) dynamic mode performs better than traditional, as $20_{(rowA)} < 21_{(rowC)}$ and $12_{(rowB)} < 14_{(rowD)}$ in sequential and parallel environments respectively.

Table 2.9: Rank table for the performance of 5 types of methods.

Algorithms	6-bus	24-bus	118-bus	46-bus	300-bus	Sum
MILP_dynamic (1 thread)	2	5	5	4	4	20
MILP_dynamic (24 threads)	3	1	4	2	2	12
MILP_traditional (1 thread)	5	3	3	5	5	21
MILP_traditional (24 threads)	4	2	2	3	3	14
BCBD_traditional (1 thread)	1	4	1	1	1	8

Circuit	ij	1-2	1-3	1-4	1-5	1-6	2-3	2-4	2-5	2-6	3-4	3-5	3-6	4-5	4-6	5-6	Execution Time (s)				
Cost	c_{ij}	40	38	60	20	38	20	40	31	30	59	20	48	63	30	61	A	B	C	D	E
$K=1$ (Infeasible)	n_{ij}^1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0.51	0.74	0.86	0.81	0.07
$K=2$ ($C=200$)	n_{ij}^2	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0	1.94	2.23	2.70	2.66	0.24
	n_{ij}^1	0	0	0	0	0	0	1	0	1	0	1	0	0	1	0					
$K=3$ ($C=180$)	n_{ij}^3	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	2.61	3.03	4.61	4.47	0.33
	n_{ij}^2	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0					
	n_{ij}^1	0	0	0	0	0	1	0	0	1	0	1	0	0	1	0					
$K=4$ ($C=180$)	n_{ij}^4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2.97	3.52	5.72	5.19	0.47
	n_{ij}^3	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0					
	n_{ij}^2	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0					
	n_{ij}^1	0	0	0	0	0	1	0	0	1	0	1	0	0	1	0					

A: MILP_dynamic (1 thread) B: MILP_dynamic (24 threads) C: MILP_traditional (1 thread)
D: MILP_traditional (24 threads) E: BCBD_traditional (1 thread)

Figure 2.17: Solution configuration and execution time for different K values with the Garver 6-bus system.

2.3.3.4 Sensitivity Analysis

As shown in Table 2.7, K has a large difference on the number of binary variables Kn_c, continuous variables $|\mathcal{S}|\cdot((K+1)n_c+3n_b)$, and inequality constraints $|\mathcal{S}|\cdot(5Kn_c+4n_bn_c+n_c)$. Therefore, a sensitivity analysis on the solution configuration and efficiency for different K values based on the Garver 6-bus system is conducted in this subsection. Fig. 2.17 depicts all the results, which can be analyzed from two aspects:

- Solution configuration: If $K=1$, there is no feasible solution even if all candidate circuits are built. One reason is that the requirement of large number of circuits on key corridors, such as $4-6$, is very hard to be replaced by other corridors. When $K=2$, global optimal solution is accessible with a total cost of 200M$. As K increases to 3, one more circuit can be built on corridor $4-6$, which results in a cost reduction of 20M$. Nevertheless, the cost cannot be reduced further by increasing K since $n_{ij}^4=0$ for all $ij\in\mathcal{C}$, which means the saturation point for the Garver 6-bus system is $K=3$.

- Solution efficiency: Discussion for $K=1$ is skipped since there is no feasible solu-

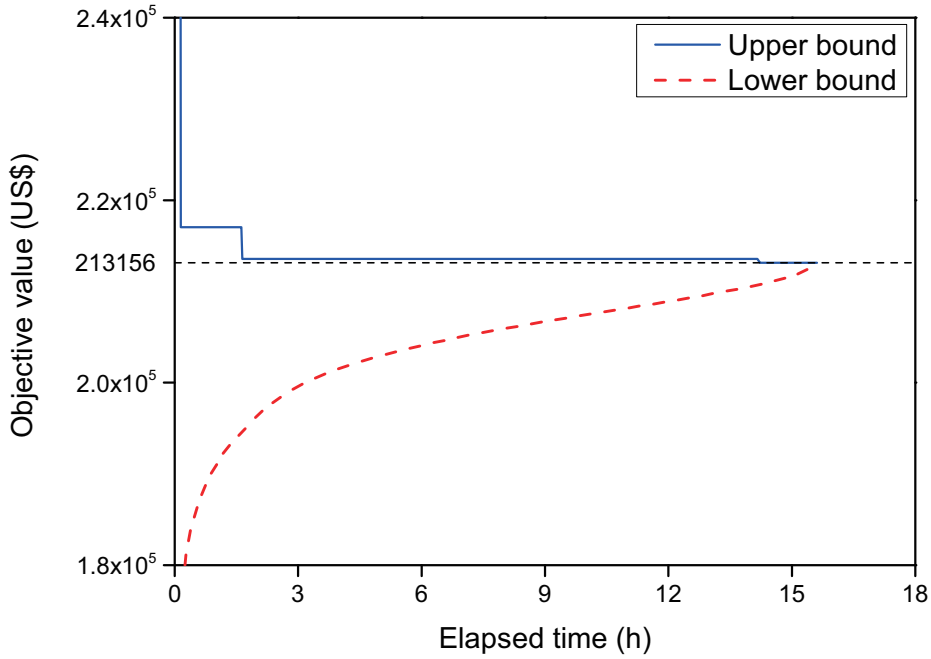


Figure 2.18: Lower and upper bounds for BCBD of the 46-bus system.

tion. Comparing $K = 4$ with $K = 3$ and $K = 2$, it can be seen that the execution time increases for all algorithms as K increases, although $K = 4$ and $K = 3$ have the same optimal solution configuration. The reason is that a large K value represents larger solution space.

2.3.3.5 Performance Analysis

Although the superiority of BCBD over MILP solvers has been revealed in the above discussion, which component (B&C framework or acceleration strategies) facilitates the computational improvement is still not recognized. Therefore, further detailed comparison has been carried out in this subsection to figure out that issue.

Lower bound is employed to depict the convergence process. As shown in section 2.3.2.1, the lower bound is determined by the objective value of master problem. With the adding of cuts at each iteration, the master problem becomes more constrained; therefore, the objective value will increase monotonically, which provides a good parameter to describe the convergence characteristic. On the other hand, the upper bound fluctuates heavily since it is related with subproblems, where a large penalty may be triggered irregularly. Fig. 2.18 shows the lower and upper bounds for BCBD of the 46-bus system, where the fluctuated upper bound is flattened by lines 10 – 12 in **Algorithm 2.5**. Compared with the upper bound with long flat intervals, the lower bound provides more information

Table 2.10: Different types of algorithms for performance analysis.

Algorithms	Description
Alg.1	MILP_dynamic (24 threads)
Alg.2	CLBD without acceleration strategies
Alg.3	BCBD without acceleration strategies
Alg.4	CLBD without multi-cut strategy
Alg.5	BCBD without multi-cut strategy
Alg.6	CLBD with full acceleration strategies
Alg.7	BCBD with full acceleration strategies

about the convergence process.

As shown in Table 2.8 and Fig. 2.14 – Fig. 2.16, the solution processes of 6-, 24-, and 118-bus systems are short and not stable (e.g., frequently sharp decrease), while the execution time of 300-bus system is too long and the convergence is not guaranteed. Therefore, the 46-bus system is finally determined as the test bed due to its moderate convergence process and time consumption.

In order to distinguish the performance enhancement achieved from different components, seven types of algorithms are separated from BCBD and classical BD (CLBD) for comparison, which are listed in Table 2.10. Fig. 2.19 illustrates the growth trend of the lower bound for *Alg.1–Alg.7* on the 46-bus system, where several findings can be observed:

- *Alg.7* is faster than *Alg.1*, showing that the B&C framework and acceleration strategies are successful for the MILP solution of SCTEP.
- *Alg.7* terminates at 16h, while *Alg.6* cannot find the global optimal until 48h, indicating that the B&C framework plays an important role in the performance improvement.
- *Alg.6* converges faster than *Alg.7* at the very beginning, the reason is that the incumbents in the searching tree of BCBD are not yet the optimum solution of the master problem, therefore, the cuts generated from subproblems cannot cut the feasible region efficiently. For CLBD, the optimal solution from master problem is always utilized.
- *Alg.6* performances better than *Alg.1* in the first 30 hours; however, MILP solver suddenly jumps to the global optimal at 33h due to the dynamic search mechanism, which is also an evidence that the MILP solver has been greatly advanced in the last decades [44].

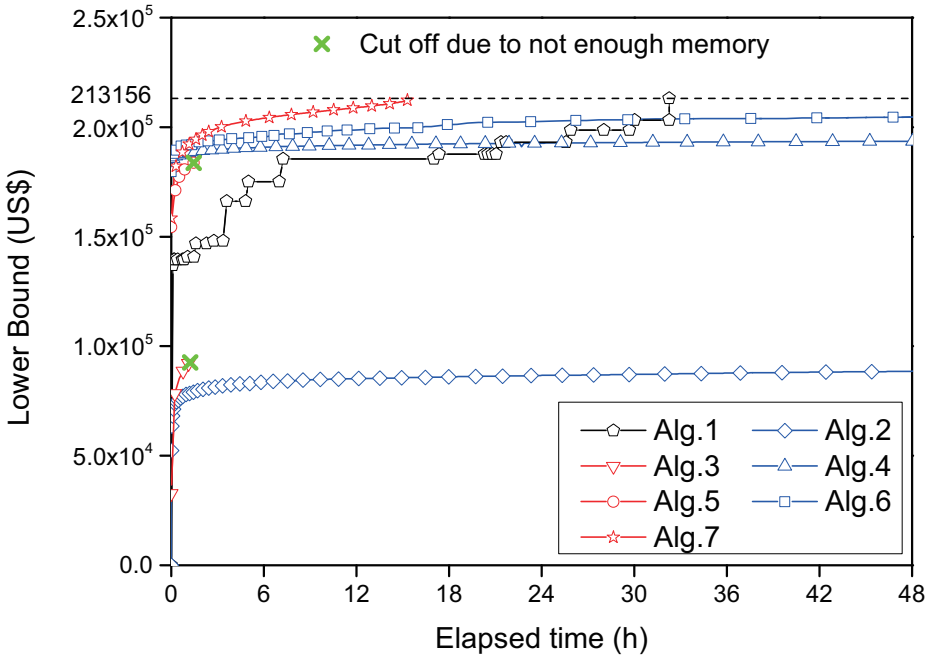


Figure 2.19: Convergence properties of different algorithms for the 46-bus system.

- It has been revealed that BD may not work well without much additional enhancements; therefore *Alg.2* and *Alg.3* show a weak performance.
- *Alg.4* and *Alg.5* present good performance compared to *Alg.2* and *Alg.3* respectively, proving that the other three acceleration strategies except multi-cut contribute a lot in the iterative process.
- The acceleration acquired by the introduction of multi-cut strategy is depicted by the fact that *Alg.7* and *Alg.6* perform better than *Alg.5* and *Alg.4*.
- One drawback of BCBD is that it may run out of memory when a huge B&C searching tree must be maintained, as shown by *Alg.3* and *Alg.5*. On the other hand, CLBD has lower requirements on the computational resources even if it runs for 48 hours.

To sum up, both the B&C framework and acceleration strategies contribute a lot to boost the efficiency of BCBD, making it perform better than the MILP solver and CLBD. Note that the searching tree of BCBD should be trimmed efficiently, otherwise it will grow to be very large and run out of the memory; therefore, the acceleration strategies are paramount for the success of BCBD. In terms of CLBD, the master problem is solved independently at each iteration, and the occupied memory will be freed when solving the subproblems.

2.4 Summary

Two subproblems related to the transmission system optimal expansion planning are investigated in this chapter, i.e., TEP and SCTEP.

Based on the DC power flow model, the TEP is formulated as a MINLP. Due to the NP-hard property, meta-heuristic PSO is introduced for the problem solution. In order to ameliorate the performance on both efficiency and quality, the MGPSO framework as well as enhancement strategies are proposed. Case studies on five test systems with as many as 186 candidate transmission circuits are presented to verify the proposed MGPSO, showing that the method achieves considerable speedup compared to commercial software Lingo in execution time. The achieved speedup has a linear relationship with search space size, indicating that the MGPSO algorithm is scalable. Performance evaluation has also been carried out on several enhancement strategies to distinguish their contribution.

Disjunctive model is employed to formulate the SCTEP problem, resulting in a MILP that can be directly solved with commercial software. Nevertheless, challenges still exist for large-scale systems due to large numbers of decision variables and constraints. Therefore, the BCBD algorithm is proposed by integrating BD into the B&C framework. Different with the classical BD framework, where the master problem is MILP, the BCBD replaces the MILP with LP, resulting in great reduction in computational resource and execution time. Four acceleration strategies have also been investigated to enhance the convergence efficiency as well as to restrict the solution space. Comprehensive computational experiments between BCBD and commercial MILP solver Cplex are conducted on five benchmark test systems ranging from 6 to 300 buses. Although parallel computing with 24 threads is enabled for some MILP solvers, the superiority of BCBD has been validated for the majority of systems. Detailed performance analysis has also been conducted to distinguish the performance improvements from B&C framework and acceleration strategies, where seven types of algorithms separated from BCBD and classical BD are involved. The results indicate that both the B&C framework and acceleration strategies contribute a lot to boost the efficiency of BCBD.

3

Transmission System Optimal Operation: ACPF and RTCA

3.1 Introduction

Alternating Current Power Flow (ACPF) analysis is one of the most fundamental tasks for the transmission system operation and optimization [61], which dominates the essential steps of many practical problems, such as contingency analysis, economic dispatch, optimal power flow, etc. The challenge of quick solution techniques always exists for the ACPF, since the shorter calculation time means better situational awareness, faster response, and less adverse impact on the system. Therefore, achieving high solution efficiency for ACPF analysis from HPC architecture is a leading and important challenge in power system analytics and computation. Except for the time-critical features, ACPF is also confronted with great challenges from the increasing system size [78]. On the other hand, as an application of ACPF, the RTCA is paramount for modern power systems as it forms the basis for important operator actions that help to improve system stability, optimize generator dispatch, manage disparate resources, prevent cascading outages, and enhance market operations. In order to alleviate the solution pressure of ACPF and RTCA, different proposals developed by combining advanced algorithms and modern computation facilities are investigated and evaluated in this chapter.

Historically, a lot of promising algorithms are developed for ACPF analysis, of which the NR [165] and FD [166] method received extensive attention due to their favorable convergence characteristics. Although derived from the NR, the FD is much simpler and more efficient algorithmically [61]. According to their philosophy, the nonlinear ACPF problem is addressed by a successive solution of LESs. Despite the fact that the iterative solver is

more desirable for the solution of large-scale LESs in the context of parallel computing [78], the direct solver is utilized for FD in this work since their solution procedures are coordinated, which is beneficial for data reuse. In addition, the direct solver is more robust for ill-conditioned problems. GPU implementation with Matlab and CUDA are carried out to evaluate various computational architectures, data storage formats, and fill-in reduction algorithms.

A series of ACPF need to be solved in RTCA with short time. Although the ACPF solution procedure investigated above can be directly extended to RTCA, the performance might be further improved with CM, where the sensitivity information between various ACPFs is utilized to save computation resources. Detailed implementation schemes with GPU and CUDA are provided with direct linear solver, including data structure, kernel function, and memory management strategies and principles. Comparison with open-source package Matpower [143] and state-of-the-art parallel computing methods are presented, where the superiority of parallel CM has been established.

3.2 Alternating Current Power Flow

This section intends to address the ACPF on GPU architecture with FD method and direct linear solver. Both NR and FD methods are reviewed in section 3.2.1 to demonstrate the superiority of FD. The direct linear solver is introduced in section 3.2.2 for the solution of LES in each iteration. GPU implementation of FD based on direct linear solver with Matlab and CUDA are presented in sections 3.2.3 and 3.2.4 respectively, as well as the experimental results and discussions.

3.2.1 ACPF Solution Methodologies

3.2.1.1 Newton-Raphson Method

Given a specified network configuration and generator power output, the ACPF determines node voltages and branch power flows such that the system operates under steady-state, i.e., the power imbalance at each bus is less or equal to a predefined tolerance ϵ. The nodal power flow equations can be given as [61]:

$$\Delta \boldsymbol{P}_i = \boldsymbol{P}_{is} - \boldsymbol{V}_i \sum_{j \in i} \boldsymbol{V}_j (G_{ij} \cos\theta_{ij} + B_{ij} \sin\theta_{ij}) = \boldsymbol{0}, \tag{3.1}$$

$$\Delta \boldsymbol{Q}_i = \boldsymbol{Q}_{is} - \boldsymbol{V}_i \sum_{j \in i} \boldsymbol{V}_j (G_{ij} \sin\theta_{ij} - B_{ij} \cos\theta_{ij}) = \boldsymbol{0}, \tag{3.2}$$

Algorithm 3.1 Iterative Process of the Newton-Raphson Method

1: Data preparation.
2: Calculate $\Delta \boldsymbol{P}$ and $\Delta \boldsymbol{Q}$ from (3.1) and (3.2) with guessed $\boldsymbol{V}$ and $\boldsymbol{\theta}$.
3: **while** $||\Delta \boldsymbol{P}||_\infty > \epsilon$ **or** $||\Delta \boldsymbol{Q}||_\infty > \epsilon$ **do**
4: Update Jacobian matrices $\boldsymbol{H}$, $\boldsymbol{N}$, $\boldsymbol{J}$, and $\boldsymbol{L}$ in (3.3).
5: LU factorization of the new coefficient matrix in (3.3).
6: Generate $\Delta \boldsymbol{\theta}$ and $\Delta \boldsymbol{V}$ by F/B substitutions in (3.3).
7: Update $\boldsymbol{V}$ and $\boldsymbol{\theta}$ with $\boldsymbol{\theta} = \boldsymbol{\theta} - \Delta \boldsymbol{\theta}$ and $\boldsymbol{V} = \boldsymbol{V} - \Delta \boldsymbol{V}$.
8: Calculate $\Delta \boldsymbol{P}$ and $\Delta \boldsymbol{Q}$ from (3.1) and (3.2).
9: **end while**
10: Output $\boldsymbol{V}$ and $\boldsymbol{\theta}$.

where $\Delta \boldsymbol{P}/\boldsymbol{P}_{is}$ and $\Delta \boldsymbol{Q}/\boldsymbol{Q}_{is}$ are the errors/specified values of the active and reactive powers. The ACPF can then be expressed as follows: for specified $\boldsymbol{P}_{is}$ and $\boldsymbol{Q}_{is}$, find voltage vector $\boldsymbol{V}$ and $\boldsymbol{\theta}$ such that the power errors $\Delta \boldsymbol{P}$ and $\Delta \boldsymbol{Q}$ are less or equal to a predefined tolerance ϵ, i.e., $||\Delta \boldsymbol{P}||_\infty \leq \epsilon$ and $||\Delta \boldsymbol{Q}||_\infty \leq \epsilon$.

By neglecting the high-order terms of the Taylor expanding series of (3.1) and (3.2), the NR method reduces to a concise form:

$$\begin{bmatrix} \Delta \boldsymbol{P} \\ \Delta \boldsymbol{Q} \end{bmatrix} = \begin{bmatrix} \boldsymbol{H} & \boldsymbol{N} \\ \boldsymbol{J} & \boldsymbol{L} \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{\theta} \\ \Delta \boldsymbol{V}/\boldsymbol{V} \end{bmatrix}, \tag{3.3}$$

where $\boldsymbol{V}$ is a diagonal matrix with the diagonal elements being the node voltage magnitudes; the elements of the Jacobian matrices $\boldsymbol{H}$, $\boldsymbol{N}$, $\boldsymbol{J}$, and $\boldsymbol{L}$ can be generated by taking partial derivations of (3.1) and (3.2). The iterative process of NR is given by **Algorithm 3.1**.

3.2.1.2 Fast Decoupled Method

By eliminating $\boldsymbol{N}$ and $\boldsymbol{J}$ in (3.3), and then rewriting $\boldsymbol{H}$ and $\boldsymbol{L}$ in the form of $\boldsymbol{VBV}$ (where two expressions are utilized for transmission line: $\cos\theta_{ij} \approx 1$ and $G_{ij}\sin\theta_{ij} \ll B_{ij}$), the correction equation of FD method is obtained as [61]:

$$\Delta \boldsymbol{P}/\boldsymbol{V} = \boldsymbol{B}'\Delta \boldsymbol{\theta}, \tag{3.4}$$

$$\Delta \boldsymbol{Q}/\boldsymbol{V} = \boldsymbol{B}''\Delta \boldsymbol{V}, \tag{3.5}$$

where $\boldsymbol{B}'$ and $\boldsymbol{B}''$ are symmetric square matrices with the dimension of $n-1$ and $n-r-1$, whose off-diagonal and diagonal elements are given in (3.6) and (3.7); n and r are the number of buses and PV nodes.

$$\boldsymbol{B}'_{ii} = \sum_{j\in i} \frac{x_{ij}}{r_{ij}^2 + x_{ij}^2}, \qquad \boldsymbol{B}'_{ij} = -\frac{x_{ij}}{r_{ij}^2 + x_{ij}^2}, \qquad \boldsymbol{B}'_{jj} = \sum_{j\in i} \frac{x_{ij}}{r_{ij}^2 + x_{ij}^2}, \tag{3.6}$$

Algorithm 3.2 Iterative Process of the Fast-Decoupled Method

1: Data preparation.
2: Generate matrices $\boldsymbol{B}'$ and $\boldsymbol{B}''$ according to (3.6) and (3.7).
3: LU factorization of $\boldsymbol{B}'$ and $\boldsymbol{B}''$ for (3.4) and (3.5).
4: Calculate $\Delta\boldsymbol{P}$ and $\Delta\boldsymbol{Q}$ from (3.1) and (3.2) with guessed $\boldsymbol{V}$ and $\boldsymbol{\theta}$.
5: Set P-iteration indicator $p_{iter} = 1$.
6: **while** $||\Delta\boldsymbol{P}||_\infty > \epsilon$ **or** $||\Delta\boldsymbol{Q}||_\infty > \epsilon$ **do**
7: **if** $p_{iter} == 1$ **then**
8: Generate $\Delta\boldsymbol{\theta}$ by F/B substitutions in (3.4).
9: Update $\boldsymbol{\theta}$ with $\boldsymbol{\theta} = \boldsymbol{\theta} - \Delta\boldsymbol{\theta}$.
10: Set $p_{iter} = 0$.
11: **else**
12: Generate $\Delta\boldsymbol{V}$ by F/B substitutions in (3.5).
13: Update $\boldsymbol{V}$ with $\boldsymbol{V} = \boldsymbol{V} - \Delta\boldsymbol{V}$.
14: Set $p_{iter} = 1$.
15: **end if**
16: Calculate $\Delta\boldsymbol{P}$ and $\Delta\boldsymbol{Q}$ from (3.1) and (3.2).
17: **end while**
18: Output $\boldsymbol{V}$ and $\boldsymbol{\theta}$.

$$\boldsymbol{B}''_{ii} = \frac{\boldsymbol{B}''_{jj}}{k_{ij}^2}, \qquad \boldsymbol{B}''_{ij} = \frac{-1}{k_{ij}x_{ij}}, \qquad \boldsymbol{B}''_{jj} = \sum_{j\in i}\left(\frac{1}{x_{ij}} + 0.5b_{ij}\right), \tag{3.7}$$

where r_{ij}, x_{ij}, b_{ij}, and k_{ij} are the resistance, reactance, total line charging susceptance, and transformer off nominal turns ratio of branch ij, respectively.

The iterative process of FD is demonstrated by **Algorithm 3.2**. It should be noted that the Lines 4 and 5 in **Algorithm 3.1** are replaced by Lines 2 and 3 in **Algorithm 3.2** and moved out of the *while* loop, resulting in a lot of time saving on the Jacobian matrix update and factorization, i.e., the FD is more efficient in terms of execution time. On the other hand, the variable coefficient matrix of NR's correction equation (3.3) is fixed for FD in (3.4) and (3.5), leading to a deterioration of convergence rate from quadratic to linear on the logarithmic coordinates, which means that the NR converges faster in accordance with the number of iterations.

In order to solve (3.4) efficiently, their coefficient matrices $\boldsymbol{B}'$ and $\boldsymbol{B}''$ are factorized at the very beginning, and then at each iteration, the modification step length $\Delta\boldsymbol{\theta}$ and $\Delta\boldsymbol{V}$ can be quickly identified by B/F substitutions. For a LES, the B/F substitutions dependent on the factorization result of coefficient matrix and the Right Hand Side (RHS) vector. In terms of LES (3.4), the RHSs are the active and reactive power mismatches, which can be quickly generated by the nodal power equations at each iteration. Fig. 3.1 depicts a general framework of the FD for the ACPF.

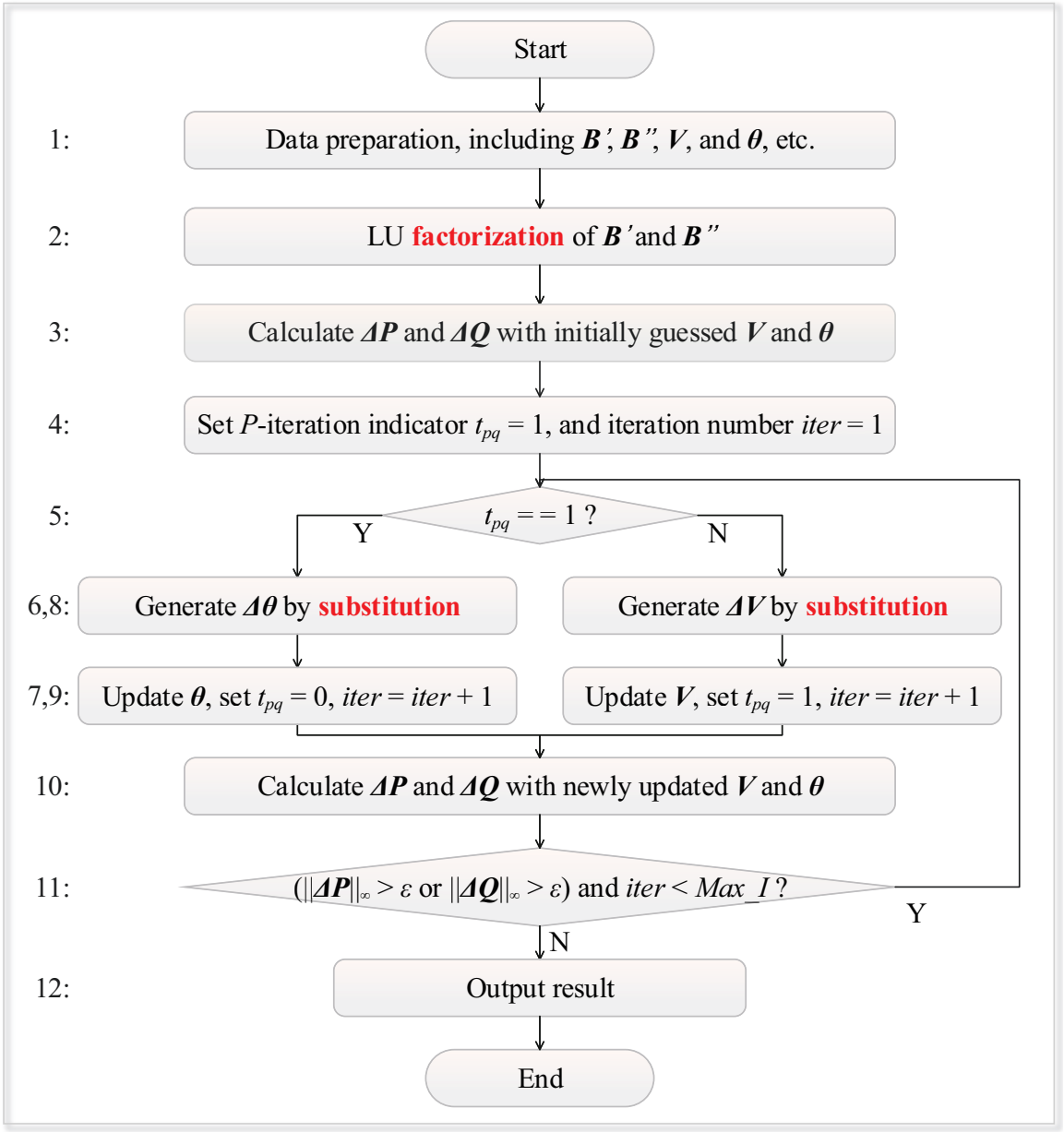


Figure 3.1: General framework of the fast decoupled method for power flow analysis.

3.2.2 Direct Linear Solver

For simplicity, the LES (3.4) is represented by a standard form in this section,

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \tag{3.8}$$

where the coefficient matrix $\boldsymbol{A}$ is sparse due to the nature of the power system structure. Generally, after factorization, the lower and upper triangular matrices of a sparse matrix are still sparse [79]. Nevertheless, the fill-ins (matrix entries modified from zero to non-zero by the factorization) are usually inevitable as shown in Fig. 3.2, which demands extra

$$A=\begin{bmatrix}8&2&2&2\\2&4&&\\2&&4&\\2&&&4\end{bmatrix}\Rightarrow A=L_AU_A=\begin{bmatrix}1&&&\\0.25&1&&\\0.25&-0.1429&1&\\0.25&-0.1429&-0.1667&1\end{bmatrix}\begin{bmatrix}8&2&2&2\\&3.5&-0.5&-0.5\\&&3.4286&-0.5714\\&&&3.3333\end{bmatrix}$$

$$B=\begin{bmatrix}4&&&2\\&4&&2\\&&4&2\\2&2&2&8\end{bmatrix}\Rightarrow B=L_BU_B=\begin{bmatrix}1&&&\\&1&&\\&&1&\\0.5&0.5&0.5&1\end{bmatrix}\begin{bmatrix}4&&&2\\&4&&2\\&&4&2\\&&&5\end{bmatrix}$$

Figure 3.2: Difference on the number of fill-ins by row and column switching.

memory space and more arithmetic operations. Fortunately, it can be greatly reduced by simple row and column switching, whose performance is demonstrated in Fig. 3.2 by shifting $\boldsymbol{A}$ to $\boldsymbol{B}$. The transformation is commonly described as,

$$\boldsymbol{B}=\boldsymbol{Q}\boldsymbol{A}\boldsymbol{Q}^T, \tag{3.9}$$

where $\boldsymbol{Q}$ is the permutation matrix derived from permutation array $\boldsymbol{q}$. In terms of Fig. 3.2, $\boldsymbol{Q}$ and $\boldsymbol{q}$ are given as,

$$\boldsymbol{Q}=\begin{bmatrix}0&0&0&1\\0&0&1&0\\0&1&0&0\\1&0&0&0\end{bmatrix},\qquad \boldsymbol{q}=\begin{bmatrix}4\\3\\2\\1\end{bmatrix}.$$

It should be noted that there is only one entry with value 1 for each row and column in $\boldsymbol{Q}$, while all the other elements are 0. In addition, $\boldsymbol{Q}$ has the following property,

$$\boldsymbol{Q}\boldsymbol{Q}^T=\boldsymbol{Q}^T\boldsymbol{Q}=\boldsymbol{I}. \tag{3.10}$$

Based on the introduction of $\boldsymbol{Q}$, the following equations can be deduced,

$$\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}\ \Rightarrow\ \boldsymbol{A}\boldsymbol{Q}^T\boldsymbol{Q}\boldsymbol{x}=\boldsymbol{b}\ \Rightarrow\ \boldsymbol{Q}\boldsymbol{A}\boldsymbol{Q}^T\boldsymbol{Q}\boldsymbol{x}=\boldsymbol{Q}\boldsymbol{b}. \tag{3.11}$$

Remark $\boldsymbol{Q}\boldsymbol{x}=\hat{\boldsymbol{x}}$ and $\boldsymbol{Q}\boldsymbol{b}=\hat{\boldsymbol{b}}$, then equation (3.11) can be rewritten as,

$$\boldsymbol{B}\hat{\boldsymbol{x}}=\hat{\boldsymbol{b}}. \tag{3.12}$$

On the basis of the above analysis, the solution process of (3.8) can be summarized as follows:

- **Step 1**: Generate permutation array $\boldsymbol{q}$ and matrix $\boldsymbol{Q}$.

- **Step 2**: Construct $\boldsymbol{B}$ according to (3.9) and then factorize it into $\boldsymbol{L}_B$ and $\boldsymbol{U}_B$.
- **Step 3**: Establish $\hat{\boldsymbol{b}}$. Except for the matrix-vector multiplication $\hat{\boldsymbol{b}} = \boldsymbol{Q}\boldsymbol{b}$, the vector $\hat{\boldsymbol{b}}$ can also be quickly generated with,

$$\hat{b}_i = b_{q_i}. \tag{3.13}$$

- **Step 4**: Deduce $\hat{\boldsymbol{x}}$ with B/F substitution,

$$\boldsymbol{L}_B\hat{\boldsymbol{y}} = \hat{\boldsymbol{b}} \quad \Rightarrow \quad \hat{\boldsymbol{y}} = \boldsymbol{L}_B^{-1}\hat{\boldsymbol{b}}, \tag{3.14}$$

$$\boldsymbol{U}_B\hat{\boldsymbol{x}} = \hat{\boldsymbol{y}} \quad \Rightarrow \quad \hat{\boldsymbol{x}} = \boldsymbol{U}_B^{-1}\hat{\boldsymbol{y}}. \tag{3.15}$$

- **Step 5**: Retrieve the final result $\boldsymbol{x}$ by any of the following methods,

$$\boldsymbol{x} = \boldsymbol{Q}^T\hat{\boldsymbol{x}}, \tag{3.16}$$

$$x_{q_i} = \hat{x}_i. \tag{3.17}$$

3.2.3 GPU Implementation with Matlab

3.2.3.1 GPU Programming Features in Matlab

Without user intervention, the Matlab code will run on the CPU and all data will be stored in the workspace allocated by Matlab in CPU. On the other hand, the GPU also provides a few Gigabytes of space called device memory. All the data stored in the device memory should be in the type of `gpuArray`. The data transformation from CPU to GPU is explicitly fulfilled by the function `gpuArray()`, or it can also be performed implicitly by any GPU-Enabled Built-in Functions (GEBFs), such as `mtimes()`. A full list of the latest GEBFs is posted in [159]. In contrast, retrieving data from GPU to CPU can be achieved by the function `gather()`.

The type of the input data determines where the GEBF will be executed. If any input arguments are with the type of `gpuArray`, the GEBF will be executed on the GPU; otherwise, the CPU will be utilized for calculation. Therefore, the simplest way to employ GPU for computation in Matlab is to employ two steps: 1) convert all the input data into `gpuArray` type; and 2) fetch results from device memory after the algorithm termination. Intermediate data generated from GEBFs running on GPU will be automatically stored in device memory in the type of `gpuArray`. The data transfer rate between CPU and GPU is limited by the PCIe interface bandwidth.

Table 3.1: General information of benchmark systems.

Cases	System scales		$\boldsymbol{B}'$		$\boldsymbol{B}''$	
	Bus	Branch	Size	Sparsity	Size	Sparsity
A	300	411	299	0.9875	231	0.9851
B	1,354	1,991	1,353	0.9974	1,094	0.9972
C	2,746	3,279	2,745	0.9988	2,382	0.9988
D	9,241	16,049	9,240	0.9996	7,796	0.9995
E	13,659	20,467	13,658	0.9997	9,567	0.9996

3.2.3.2 Various Implementation Strategies

In order to explore the performance of the FD for ACPF in detail, different data storage formats, LES solution techniques, and implementation platforms are investigated and compared, which can be divided into the following three pairs:

- **CPU versus GPU**: As two different architectures, CPU and GPU have distinctive area of expertise. Generally, CPU is suitable for randomly accessed computing, while GPU is skillful for intensively regulated calculation.
- **`lu()` versus `mldivide()`**: Except for the factorization strategy introduced in section 3.2.2, which is based on the GEBF `lu()`, Matlab also provides another powerful LES solution technique `mldivide()`. The former gains profits from the iterative process of ACPF, where the coefficient matrices of LESs are fixed. Based on the detection of the coefficient matrix property, the latter dispatches an appropriate solver from its formidable arsenal to minimize the computation time.
- **Dense versus Sparse**: As shown in Table 3.1, the coefficient matrices $\boldsymbol{B}'$ and $\boldsymbol{B}''$ are extremely sparse and suitable for sparse technique application. Nevertheless, the two GEBFs `lu()` and `mldivide()` do not support sparse `gpuArray` at present, i.e., the sparse version of FD on GPU is not feasible on the basis of GEBFs. Therefore, only the dense FD is implemented on the GPU.

3.2.3.3 Experimental Results and Discussions

Five benchmark systems retrieved from [143] are utilized for numerical experiments. Table 3.1 summarizes basic information on the power system scale, matrix size, and sparsity. The implementation platform includes: Intel Xeon E5-2620 CPU with 32GB RAM, Nvidia® GeForce Titan Black GPU, Matlab version 2015b, CUDA version 8.0, and Visual Studio 2015.

Table 3.2: Execution time of different types of FD with dense matrices using Matlab (s).

Cases	`lu()`		`mldivide()`	
	CPU	GPU	CPU	GPU
A	**0.018**	0.257	0.063	0.160
B	**0.282**	1.402	0.751	0.762
C	**1.420**	8.048	6.606	4.203
D	**17.413**	out of memory	141.371	out of memory
E	**37.847**	out of memory	327.209	out of memory

Table 3.3: Execution time of different types of FD with sparse matrices using Matlab (s).

Cases	`lu()`		`mldivide()`	
	CPU	GPU	CPU	GPU
A	**0.008**	not supported	0.015	not supported
B	0.062	not supported	**0.059**	not supported
C	0.198	not supported	**0.173**	not supported
D	4.143	not supported	**0.964**	not supported
E	7.986	not supported	**1.303**	not supported

All the results are grouped into Table 3.2 and Table 3.3 according to dense and sparse storage types respectively. Fig. 3.3 and Fig. 3.4 give the visualization for Table 3.2 and Table 3.3 for the purpose of identifying the increasing trend of execution time along with system size. The following observations can be collected corresponding to the above comparison categories:

- **CPU versus GPU**: Since there is no GPU result in Table 3.3, the finding is drawn from Table 3.2 and Fig. 3.3. If `lu()` is employed, the CPU is always faster than GPU, but the speedup decreases from 14.5× in `CaseA` to 5.7× in `CaseC`. As highlighted in Table 3.2, GPU outperforms CPU in `CaseC` where `mldivide()` is utilized. Overall, two remarks should be given: 1) GPU performs better for larger systems; and 2) the limited device memory space restricts its utilization for large-scale systems with dense matrices.

- **`lu()` versus `mldivide()`**: To evaluate the performance of `lu()` and `mldivide()`, the implementation platform should be separated. On the GPU, the superiority of `mldivide()` has been validated by all successive cases. On the other hand, if run on CPU, `lu()` outperforms `mldivide()` with dense matrices in Table 3.2; nevertheless, the circumstance is totally reversed for sparse matrices in Table 3.3. Therefore, the superiority depends on which architecture is utilized.

- **Dense versus Sparse**: According to Table 3.2 and Table 3.3, it is obvious that the sparse techniques benefit both `lu()` and `mldivier()` in CPU. Although dense ma-

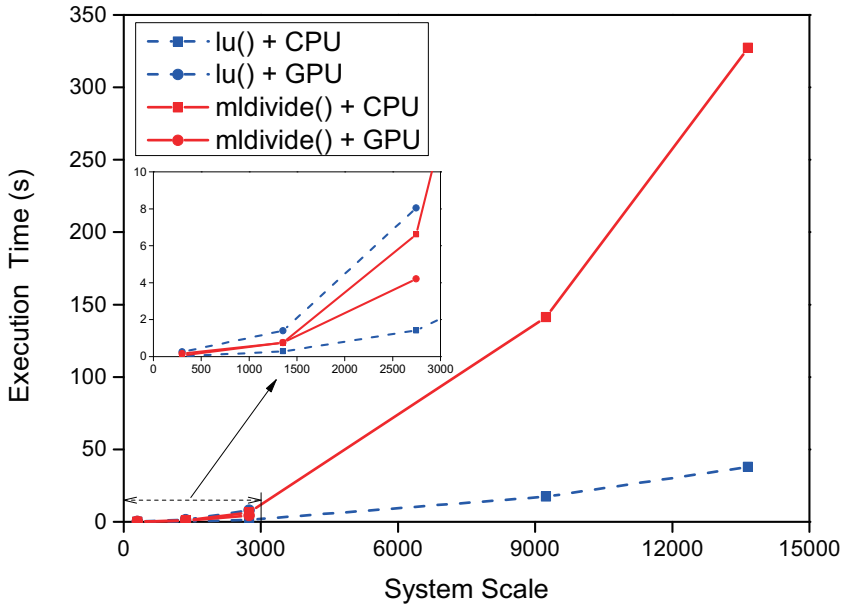


Figure 3.3: Execution time of different types of FD with dense matrices.

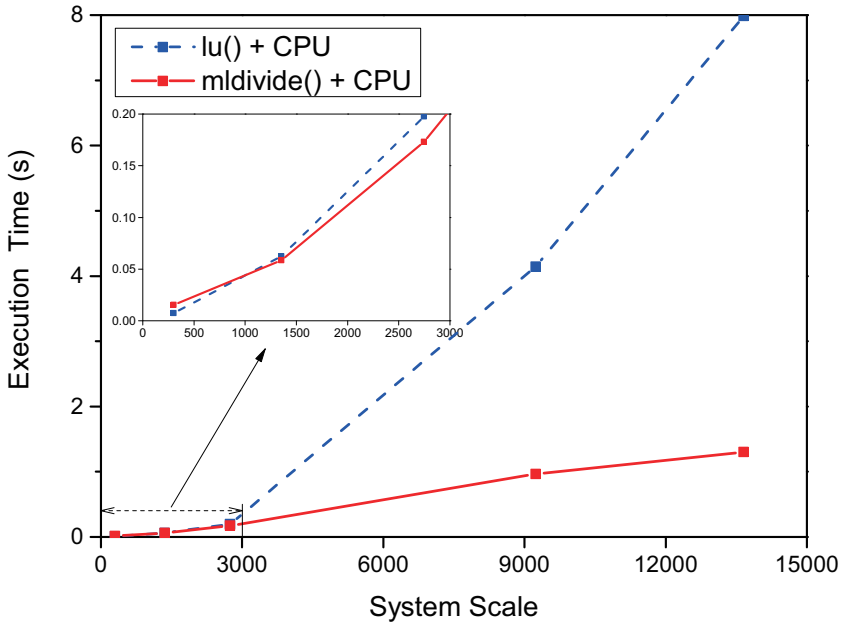


Figure 3.4: Execution time of different types of FD with sparse matrices.

trix is fully supported with GPU, the performance is only mediocre. On the contrary, the support for GPU with sparse matrices requires further investigation.

In addition to the above findings corresponding to implementation, more observations related with computation complexity and scalability are accessible. Without rigorous mathematical analysis, the execution time in the same platform can be regarded as an

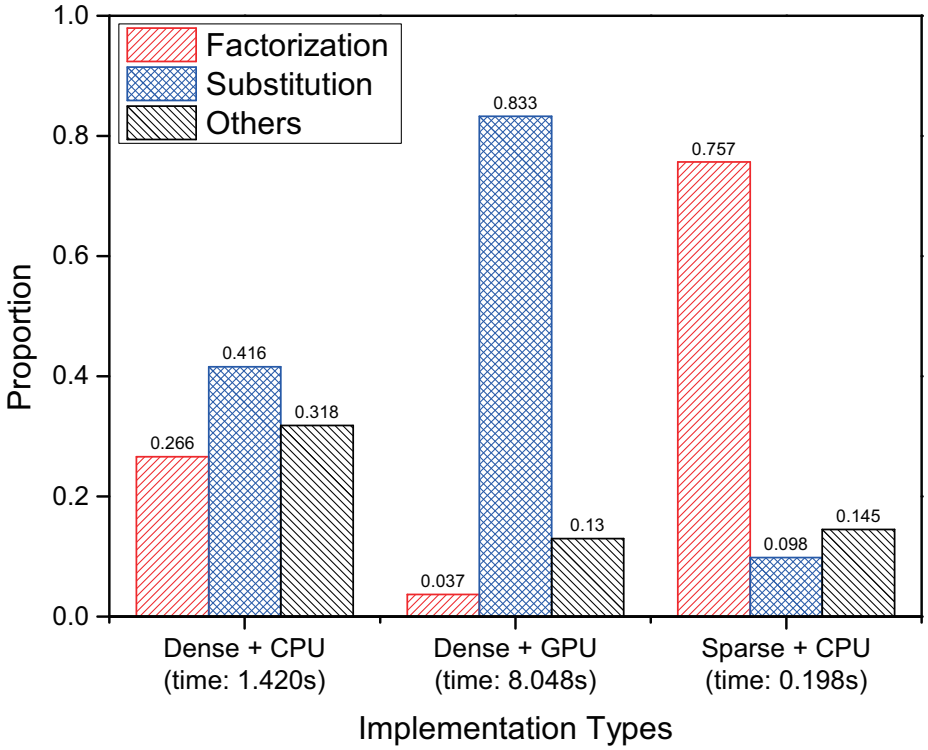


Figure 3.5: Execution time proportions of different steps for the FD with `lu()`.

indication of computation burden. For each line in Fig. 3.3 and Fig. 3.4, the computing environment of all five cases is the same; therefore, the line increasing trend represents the computation complexity. It can be seen in Fig. 3.3 that all lines are steep, which means the execute time increases faster than the system scales. On the other hand, the solid line in Fig. 3.4 is the mostly flat, i.e., the scalability of sparse `mldivide()` is more favorable.

The main steps of FD shown in Fig. 3.1 and section 3.2.1 are also analyzed. Fig. 3.5 illustrates the execution time proportion of main steps for `CaseC`. It can be seen that the substitution process, which is highly sequential, heavily drags the performance in GPU with dense matrices. The improvement on sparse matrices should be put on the factorization process in the future since it consumes the largest amount of time.

3.2.4 GPU Implementation with CUDA

3.2.4.1 GPU Programming Features in CUDA

Different with C functions running on the CPU only once with one call, the `kernels` are CUDA C extended functions that can be executed N times simultaneously with N different threads. `Kernels` access the input data from device memory spaces, including global, constant, texture, shared, and local memories [8]. The memory throughput and multipro-

Table 3.4: Fill-in reductions achieved by the AMD and RCM algorithms.

Cases	Default	AMD reordering		RCM reordering	
	Size	Size	Reduction	Size	Reduction
A	7,889	1,640	79.21%	2,515	68.12%
B	149,064	6,934	95.35%	13,036	91.25%
C	451,657	17,328	96.16%	58,326	87.09%
D	3,709,484	65,876	98.22%	200,921	94.58%
E	4,078,641	79,751	98.04%	228,221	94.40%

cessor occupancy achieved by the `kernels` greatly determine the parallel efficiency of the whole application, which demands careful code tuning and proper algorithm structure design. Fortunately, a lot of GPU-accelerated libraries containing highly-optimized algorithms and functions are provided by CUDA [160], such as cuBLAS, cuSPARSE, and cuSOLVER.

3.2.4.2 Implementation Schemes

Although execution on GPU with single data type is much faster, it cannot meet the precision requirement of $\epsilon = 10^{-8}$; therefore, the double precision data is utilized in this work. Except for the data preparation and condition judgments, the majority of FD steps shown in Fig. 3.1 are fulfilled with the refined `kernels` contained in cuSOLVER, such as LU factorization and substitution. As indicated in section 3.2.2, for sparse coefficient matrices, the permutation is of key importance for the reduction of the fill-ins. Two strategies for reordering provided by cuSOLVER are implemented, i.e., reverse Cuthill-Mckee (RCM) and Approximate Minimum Degree (AMD) algorithms. The intuitive performance of RCM and AMD is illustrated in Fig. 3.6, where $\boldsymbol{B}'$ is generated from `CaseB`. It can be seen that both AMD and RCM gain excellent performance by curtailing the number of fill-ins from 149,064 to 6,934 and 13,036 respectively, with the reduction rate reaching 95.35% and 91.25% respectively. The behavior of AMD and RCM for other cases are summarized in Table 3.4.

3.2.4.3 Experimental Results and Discussions

It is noticeable in Table 3.4 that the AMD outperforms RCM in the fill-in reduction; nevertheless, the performance is reversed when they are integrated in the FD, which is shown in Fig. 3.7. The speedup of RCM over AMD is also demonstrated in Fig. 3.7, which indicates that the difference is even higher for large-scale systems. One of the explanation for this reversal is that the AMD pursues more powerful algorithmic performance with the sacrifice

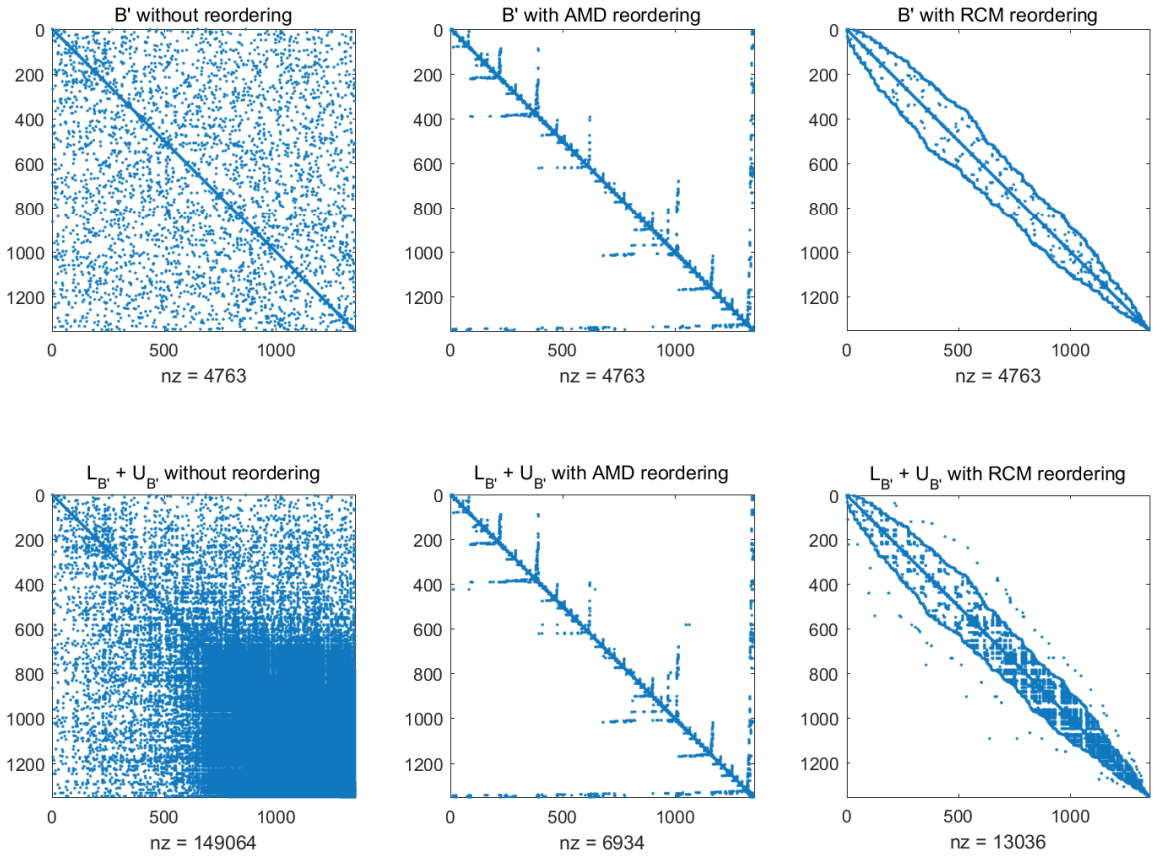


Figure 3.6: Sparsity structure of $\boldsymbol{B}'$ and $\boldsymbol{L}'_B + \boldsymbol{U}'_B$ for `caseB`.

of longer execution time, which means the AMD is more preferable for memory-restricted circumstances.

Several types of FD coded with Matlab are implemented in section 3.2.3.1; however, the performance of GPU-enabled ones is unsatisfactory. Therefore, the most efficient CPU version (sparse matrix and `mldivide()` with Matlab running on CPU, the fourth column of Table 3.3) is utilized in this subsection for comparison with AMD and RCM, whose execution time is given in Fig. 3.7. Table 3.5 summarizes the results. Although AMD is much slower than RCM, it is still more efficient than the Matlab implementation. RCM gains a maximum speedup of 4.16× over the fastest Matlab execution in `CaseE` with only 0.313s.

3.3 Real-Time Contingency Analysis

Although FD method (**Algorithm 3.2**) achieves good performance in the solution of single ACPF, its performance on RTCA (where multiple ACPFs need to be solved in short time)

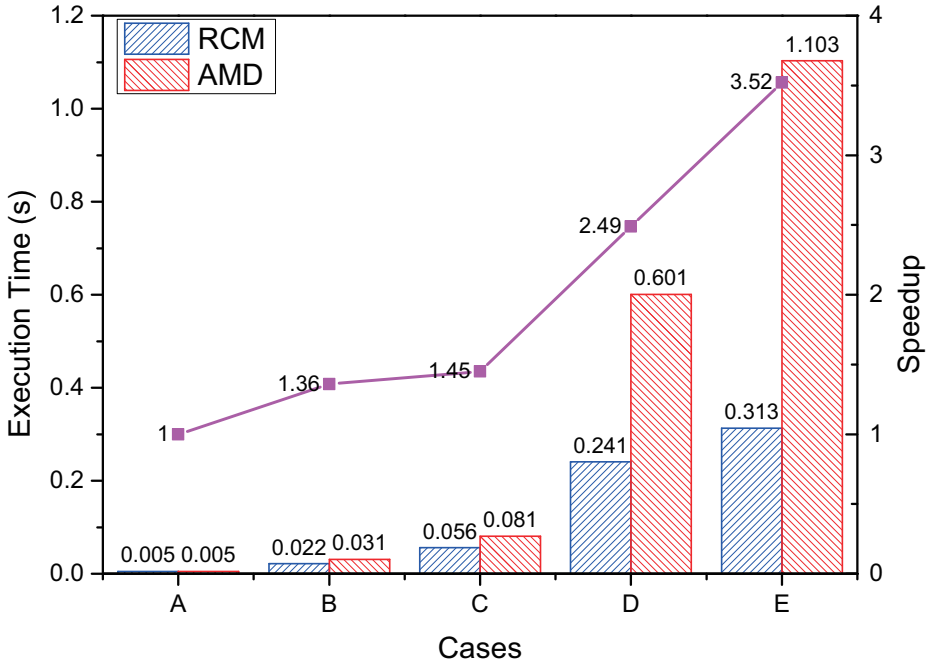


Figure 3.7: Execution time of FD with cuSOLVER based on AMD and RCM.

Table 3.5: Speedups gained by the AMD and RCM algorithms implemented with CUDA over the fastest Matlab implementation.

Algorithms	CaseA	CaseB	CaseC	CaseD	CaseE
AMD	3.05	1.95	2.14	1.60	1.18
RCM	3.05	2.66	3.09	4.00	4.16

is limited. If n_c ACPFs are addressed with **Algorithm 3.2**, there will be n_c times of LU decomposition for $\boldsymbol{B}'$ and $\boldsymbol{B}''$, which is a heavy computational workload. Fortunately, the CM is capable to reduce the number of LU factorization from n_c to 1 since the coefficient matrices for different scenarios are similar. Section 3.3.1 illustrates the solution process of CM with detailed formulation and flowchart. Sparse linear solver as well as programming strategies related to data structure designing and coding are given in section 3.3.2. Comprehensive comparisons with Matpower and state-of-the-art methods are given in section 3.3.3 to validate the performance of the proposal.

3.3.1 Compensation Method

Instead of concentrating on single ACPF, this section demands to address practical RTCA with multiple ACPFs/scenarios. Evidently, **Algorithm 3.2** can be directly utilized to tackle each scenario, but the performance is poor since large numbers of LU decomposition should be performed. Inversely, the CM intends to save the time and effort correspond-

ing to LU factorization of different scenarios since their coefficients $\boldsymbol{B}'$ and $\boldsymbol{B}''$ are very similar. Take the scenario of branch ij outage as an example, the implementation process of CM [80] can be summarized as follows. As inputs, the information related to the base case is available, i.e., $\boldsymbol{B}'$ and $\boldsymbol{B}''$ are known and already decomposed into $\boldsymbol{B}' = \boldsymbol{L}'\boldsymbol{U}'$ and $\boldsymbol{B}'' = \boldsymbol{L}''\boldsymbol{U}''$. For simplicity, only the solution steps of (3.4) are given; (3.5) can be addressed accordingly.

1. Based on the topology variation induced by the outage of branch ij, $\boldsymbol{B}'$ can be updated as:
$$\boldsymbol{B}'_* = \boldsymbol{B}' + \Delta\boldsymbol{B}'_* = \boldsymbol{B}' + \boldsymbol{M}'_*\delta\boldsymbol{b}'_*\boldsymbol{M}'^T_*, \tag{3.18}$$
where subscript $_*$ is a stamp for the specified scenario, i.e., outage of branch ij; $\delta\boldsymbol{b}'_*$ is a $m \times m$ matrix containing correction information, $m = \{1, 2\}$; $\boldsymbol{M}'_*$ is a $n \times m$ incidence matrix relates to i and j. The details on generating $\delta\boldsymbol{b}'_*$, $\boldsymbol{M}'_*$, and m will be demonstrated later.

2. Calculate intermediate matrix:
$$\boldsymbol{B}'^{-1}_* = \boldsymbol{B}'^{-1} - \boldsymbol{B}'^{-1}\boldsymbol{M}'_*\boldsymbol{c}_*\boldsymbol{M}'^T_*\boldsymbol{B}'^{-1}, \tag{3.19}$$
$$\boldsymbol{c}'_* = \left[\boldsymbol{I} + \delta\boldsymbol{b}'_*\boldsymbol{M}'^T_*(\boldsymbol{U}'^{-1}(\boldsymbol{L}'^{-1}\boldsymbol{M}'_*))\right]^{-1}\delta\boldsymbol{b}'_*, \tag{3.20}$$
where $\boldsymbol{I}$ is a $m \times m$ identical matrix.

3. Calculate the voltage angle variation vector as follows:
$$\Delta\boldsymbol{\theta}_* = \boldsymbol{L}'^{-1}(\Delta\boldsymbol{P}_*/\boldsymbol{V}_*), \tag{3.21}$$
$$\Delta\boldsymbol{\theta}_* = \Delta\boldsymbol{\theta}_* - \boldsymbol{L}'^{-1}\left(\boldsymbol{M}'_*\boldsymbol{c}'_*\boldsymbol{M}'^T_*(\boldsymbol{U}'^{-1}\Delta\boldsymbol{\theta}_*)\right), \tag{3.22}$$
$$\Delta\boldsymbol{\theta}_* = \boldsymbol{U}'^{-1}\Delta\boldsymbol{\theta}_*. \tag{3.23}$$

The detailed derivation process of the above steps based on inverse matrix modification lemma [170] is given in Appendices A and B.

It should be noted that all the inverse operations of $\boldsymbol{L}'$ and $\boldsymbol{U}'$ can be performed by F/B substitutions to reduce the computation burden, which are marked in the above by parentheses. Although the inverse operation indicated by brackets in (3.20) is inevitable, fortunately, the matrix size ($m \times m$) is limited. The matrix inverse operations for $m = 1$ and $m = 2$ are trivial, i.e.,
$$\left[a\right]^{-1} = \left[\tfrac{1}{a}\right] \text{ and } \begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \begin{bmatrix} \frac{d}{ad-bc} & \frac{-b}{ad-bc} \\ \frac{-c}{ad-bc} & \frac{a}{ad-bc} \end{bmatrix}. \tag{3.24}$$

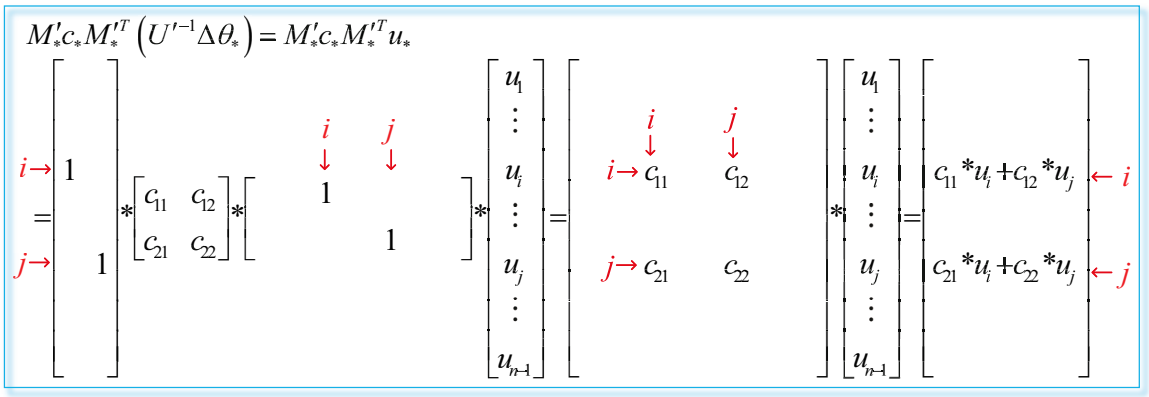


Figure 3.8: Demonstration of fixed pattern calculation.

Algorithm 3.3 Data preparation of $\delta \boldsymbol{b}'_*$ and $\boldsymbol{M}'_*$

1: **if** Node i is a slack bus **then**
2: Let $\boldsymbol{M}'_* = \boldsymbol{e}'_j$ and $\delta \boldsymbol{b}'_* = -\frac{x_{ij}}{r_{ij}^2+x_{ij}^2}$.
3: **else**
4: **if** Node j is a slack bus **then**
5: Let $\boldsymbol{M}'_* = \boldsymbol{e}'_i$ and $\delta \boldsymbol{b}'_* = -\frac{x_{ij}}{r_{ij}^2+x_{ij}^2}$.
6: **else**
7: Set $\rho' = -\frac{x_{ij}}{r_{ij}^2+x_{ij}^2}$,
8: Let $\boldsymbol{M}'_* = \begin{bmatrix} \boldsymbol{e}'_i & \boldsymbol{e}'_j \end{bmatrix}$ and $\delta \boldsymbol{b}'_* = \begin{bmatrix} \rho' & -\rho' \\ -\rho' & \rho' \end{bmatrix}$.
9: **end if**
10: **end if**
11: Output $\delta \boldsymbol{b}'_*$ and $\boldsymbol{M}'_*$.

More notes are given for the calculations related with $\boldsymbol{M}'_*$. Due to its highly sparse and fixed pattern, the calculation can be predefined to save time and effort, i.e., only few fixed points of the final result should be calculated and filled. Take $\left(\boldsymbol{M}'_* \boldsymbol{c}'_* \boldsymbol{M}'^T_* (\boldsymbol{U}'^{-1}\Delta\boldsymbol{\theta}_*)\right)$ in (3.22) as an example, suppose $m = 2$ and rewriting the dense column vector $(\boldsymbol{U}'^{-1}\Delta\boldsymbol{\theta}_*)$ as $\boldsymbol{u}_*$, the solution process is illustrated by Fig. 3.8, where the final result can be directly derived and filled without any intermediate calculation.

The critical step of utilizing CM is constructing $\delta \boldsymbol{b}'_*$ and $\boldsymbol{M}'_*$, which dominates the solution steps (3.20)–(3.23). The pattern and value of $\delta \boldsymbol{b}'_*$ and $\boldsymbol{M}'_*$ are dependent on the parameters of branch ij and the node type of bus i and j. The pseudo code of generating $\delta \boldsymbol{b}'_*$ and $\boldsymbol{M}'_*$ is summarized in **Algorithm 3.3**, where $\boldsymbol{e}'_i$ and $\boldsymbol{e}'_j$ are basis vectors with size $n-1$. Accordingly, the generation process for $\delta \boldsymbol{b}''_*$ and $\boldsymbol{M}''_*$ is given by **Algorithm 3.4**, where $\boldsymbol{e}''_i$ and $\boldsymbol{e}''_j$ are basis vectors with size $n-r-1$.

Algorithm 3.4 Data preparation of $\delta \boldsymbol{b}''_*$ and $\boldsymbol{M}''_*$

1: **if** Node i is a PQ bus **then**
2: **if** Node j is a PQ bus **then**
3: Set $\sigma = \frac{1}{k_{ij}x_{ij}}$ and $\rho'' = -\frac{1}{x_{ij}} - 0.5b_{ij}$,
4: Let $\boldsymbol{M}''_* = \begin{bmatrix} \boldsymbol{e}''_i & \boldsymbol{e}''_j \end{bmatrix}$ and $\delta \boldsymbol{b}''_* = \begin{bmatrix} \rho''/k_{ij}^2 & \sigma \\ \sigma & \rho'' \end{bmatrix}$.
5: **else**
6: Let $\boldsymbol{M}''_* = \boldsymbol{e}''_i$ and $\delta \boldsymbol{b}''_* = -\frac{1}{x_{ij}} - 0.5b_{ij}$.
7: **end if**
8: **else**
9: **if** Node j is a PQ bus **then**
10: Let $\boldsymbol{M}''_* = \boldsymbol{e}''_j$ and $\delta \boldsymbol{b}''_* = -\frac{1}{x_{ij}} - 0.5b_{ij}$.
11: **else**
12: Let $\boldsymbol{M}''_* = \boldsymbol{0}$ and $\delta \boldsymbol{b}''_* = 0$.
13: **end if**
14: **end if**
15: Output $\delta \boldsymbol{b}''_*$ and $\boldsymbol{M}''_*$.

Fig. 3.9 illustrates the flowchart of CM. It can be seen that the kernel of FD has been fully inherited, i.e., each step of **Algorithm 3.2** has been reused without revision except for Steps 8 and 12. The calculation of $\Delta\boldsymbol{\theta}$ and $\Delta\boldsymbol{V}$ is performed by F/B substitutions with Lines 8 and 12 in **Algorithm 3.2**, while in CM, it is fulfilled by the execution of (3.20) – (3.23) corresponding to Lines 12 and 13 in Fig. 3.9. Another difference lies in Steps 6 and 7 of CM. The latter has been described by **Algorithm 3.3** and **Algorithm 3.4**, while the former is omitted since it is similar to (3.18).

In order to validate the solution efficiency, the CM is implemented on a 2746-bus test system. It takes 40.31ms to analyze each scenario, which means 1,488 contingencies can be evaluated every minute. Nevertheless, in industry application, this performance is far from satisfactory. For example, Midcontinent Independent System Operator (MISO) [161], Electric Reliability Council of Texas (ERCOT) [66], and Pennsylvania-New Jersey-Maryland Interconnection (PJM) [162] simulate 2,875, 3938, and 6,000 contingency scenarios of large-scale power system in one minute, respectively.

3.3.2 Parallel Implementation on GPUs

Since the sequential CM is not sufficient for industry application, advanced parallel hardware GPU is resorted for acceleration in this section, where CUDA [8] version 8.0 is employed for programming.

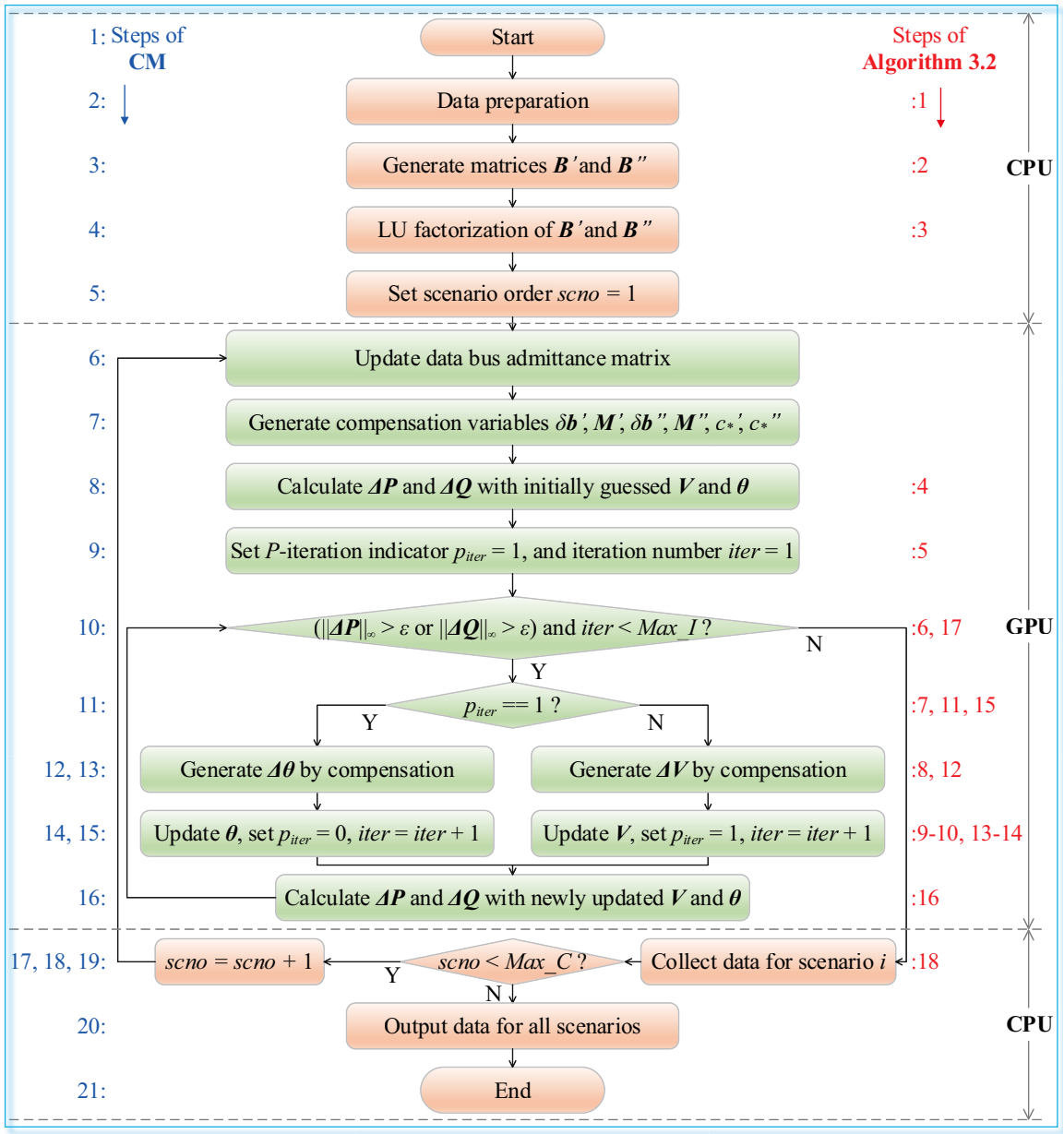


Figure 3.9: Flowchart of the compensation method.

3.3.2.1 Data Structure and Precision

It is widely accepted that sparse matrix techniques should be adopted for the solution of large-scale power systems due to their high sparsity ratio of bus admittance matrix $\boldsymbol{Y}$. In order to reduce the amount of data transfer and the number of atomic operations, the sparse storage formats are commonly employed. In accordance with [79] and [143], the Compressed Sparse Column (CSC) format is utilized for the storage of $\boldsymbol{Y}$, $\boldsymbol{B}'$, $\boldsymbol{L}'$, $\boldsymbol{U}'$, $\boldsymbol{B}''$, $\boldsymbol{L}''$ and $\boldsymbol{U}''$. Unfortunately, part of `cuSparse` library [163] operations do not support the CSC format, such as matrix-vector multiplication (related with $\boldsymbol{Y}$) and F/B substitutions

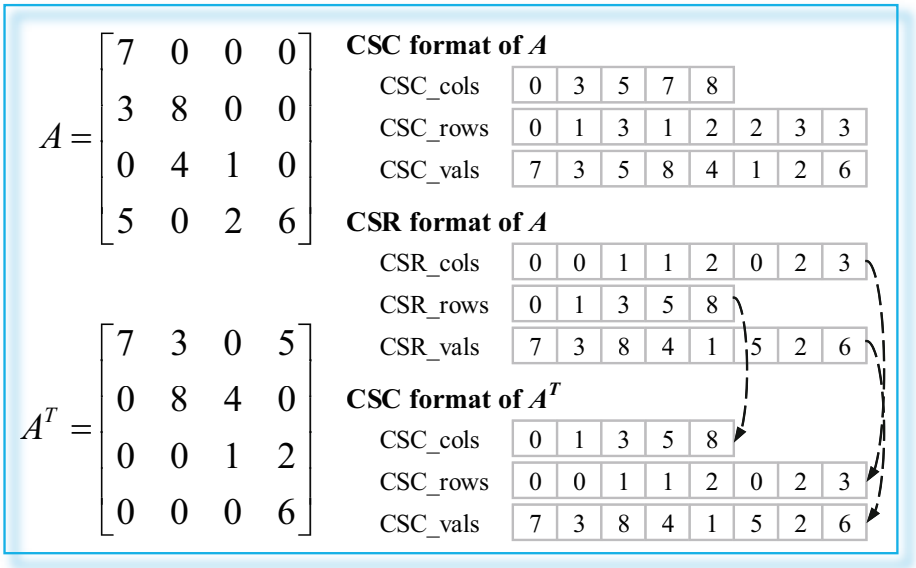


Figure 3.10: Transformation from CSC to CSR by matrix transposition.

(implemented on $\boldsymbol{L}'$, $\boldsymbol{U}'$, $\boldsymbol{L}''$ and $\boldsymbol{U}''$). Therefore, $\boldsymbol{L}'$, $\boldsymbol{U}'$, $\boldsymbol{L}''$ and $\boldsymbol{U}''$ are translated into the Compressed Sparse Row (CSR) format from CSC. The transformation can be performed by a matrix transposition, which is demonstrated in Fig. 3.10. On the other hand, the matrix $\boldsymbol{Y}$ is free from transformation due to its symmetry, i.e., its CSC is the same with CSR. Except for the matrices, all the vectors are stored in dense format.

In this work, the NVIDIA GeForce GTX 1080 GPU is utilized, whose compute capability is 6.1 with 8.876TFLOP/s and 277.36GFLOP/s on single and double precision. Although the single precision is much faster, the accuracy is limited, which is illustrated by Fig. 3.11, therefore the double precision is finally applied.

3.3.2.2 Sparse Linear Solver

The linear solver is a basic component of key importance for the whole performance of NR, FD, and CM, especially for NR, where it may take 80% of the total execution time [75]. Two types of sparse linear solver are intensively discussed: iterative methods and direct methods. Although the former demands less memory and the solution process is controllable based on absolute or relative error [69,78], the latter is more popular in the community [71–77].

Decomposition or factorization of the coefficient matrix $\boldsymbol{A}$ is ubiquitous in direct methods. Although the sparsity of $\boldsymbol{A}$ is inherited by $\boldsymbol{L}$ and $\boldsymbol{U}$, there still exists a lot of fill-ins (entries in $\boldsymbol{L}$ and $\boldsymbol{U}$ that do not appear in $\boldsymbol{A}$). Generally, fewer fill-ins means less require-

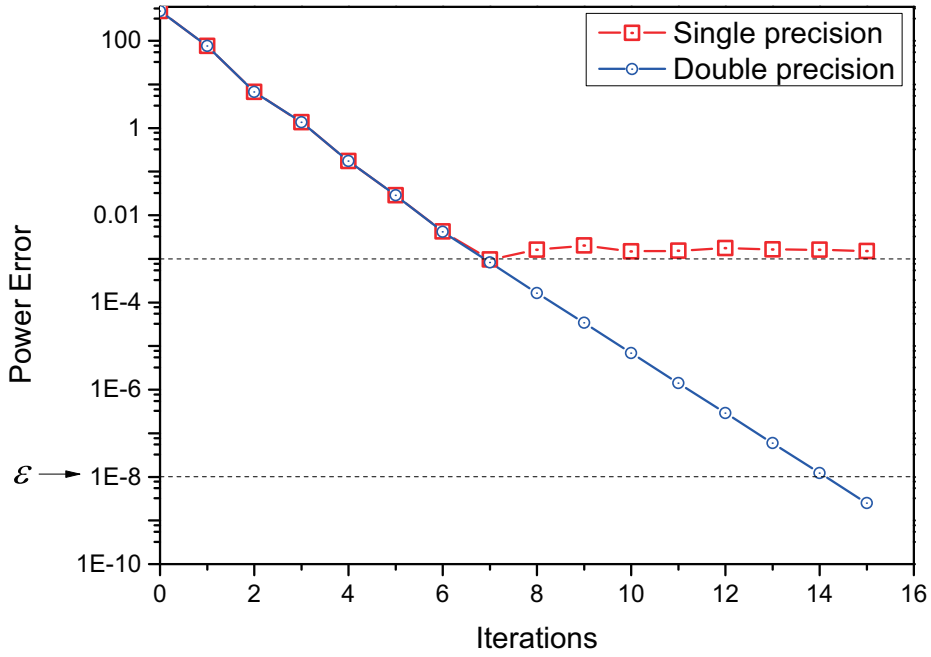


Figure 3.11: Convergence properties of CM for a 2746-bus test system with single and double precision.

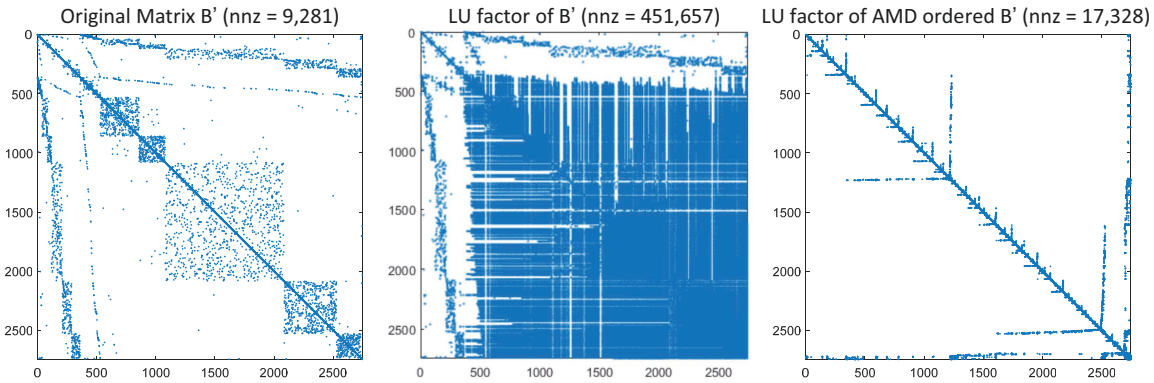


Figure 3.12: Performance illustration of AMD on a 2746-bus test system with sparsity pattern (nnz is the number of nonzero elements).

ment on the system memory and atomic operations; therefore, the AMD algorithm [79] is utilized, which is a heuristic to find the permutation P such that $P\boldsymbol{A}P^T$ has fewer fill-ins than $\boldsymbol{A}$ after the factorization. Fig. 3.12 illustrates the performance of AMD on the Jacobian matrix $\boldsymbol{B}'$ of a 2746-bus system. After AMD ordering, the $\boldsymbol{B}'$ is more concise, and the number of fill-ins after LU factorization is reduced from 451,657 to 17,328, which means the reduction rate reaches 96.16%.

3.3.2.3 Single GPU Architecture

i. Kernel Design Strategies

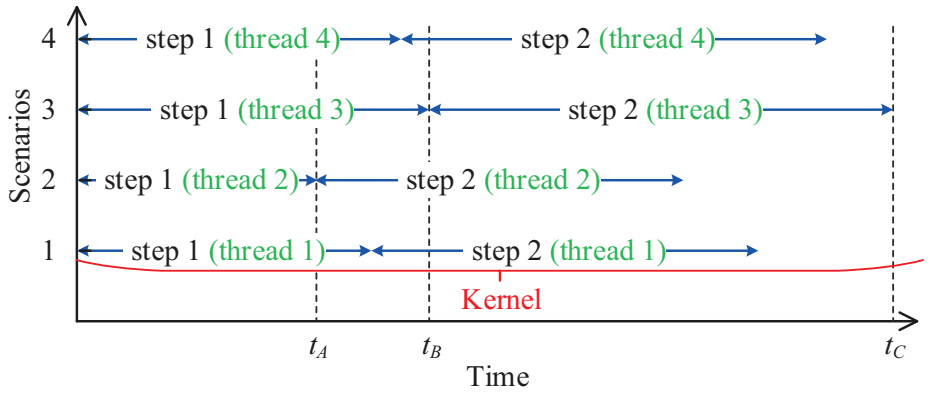
One kernel function can be executed N times in parallel if N different CUDA threads are launched. Since all scenarios are independent, a naive parallel implementation strategy is that integrating all the CM steps into one whole kernel function and running in a fixed thread, whose execution pattern is demonstrated by Fig. 3.13 (a). Although the integrated kernel strategy is straightforward and the waiting time can be minimized by static/dynamic load balancing [65], it is not suitable for CUDA due to its random data access feature. In CUDA, the parallel threads are managed, scheduled, and executed in groups of 32 called warps. A warp executes one common instruction at a time, and the full efficiency is achieved when all 32 threads within the warp agree on their execution path (coalesced access) [8]. Faced with path diversity, the warp serially executes each branch path by disabling threads that are not on that path. When all paths complete, the warp converges back to the same execution path. In Fig. 3.13 (a), threads 1 – 4 in the same warp should access successive addresses and do the same operations to achieve growth in performance. However, at t_A, thread 2 starts to execute step 2 whereas the other threads are still doing step 1, thus the coalesced access is declined. In addition, the path diversity increases as time goes on.

In order to achieve coalesced access, the whole kernel function is decoupled in Fig. 3.13 (b), i.e., each step is realized with one kernel function. It is observable that all threads do the same operations (go to the same path) within various intervals, e.g., execute step 1 from 0 to t_E and step 2 from t_E to t_F. Therefore, there is no branch diversity within warps. On the other hand, successive addresses are easily accessed by threads since scenario data is commonly stored in regulation. Detailed access pattern will be exemplified in the following subsection. Compared with Fig. 3.13 (a) and (b), the execution efficiencies of thread 2 for step 1 should be similar, i.e., $t_A \approx t_D$. The reason is that coalesced execution pattern is also achieved by Fig. 3.13 (a) from 0 to t_A. However, due to path diversity, deterioration will subsequently emerge in Fig. 3.13 (a), thus $t_B > t_E$ and $t_C > t_F$.

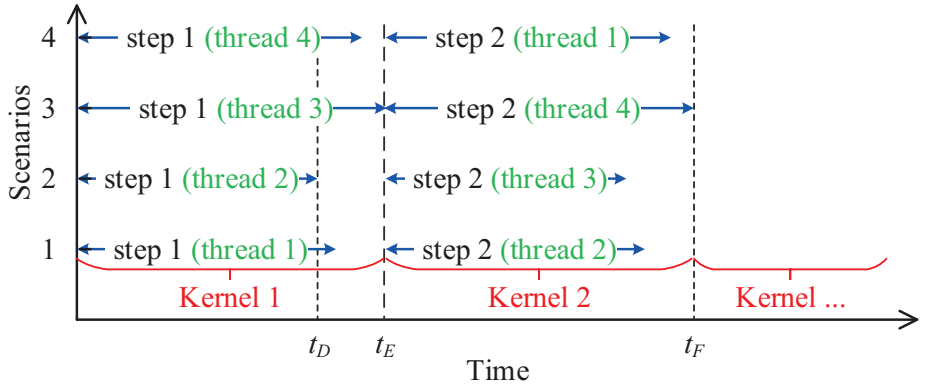
It should be noted that there is an implicit barrier between the successive kernels, which guarantees the logical sequence between iterations. Theoretically, wait time always exists before the barrier, but it is relatively small since the work load for each scenario on each kernel is even. Due to the multiple kernels execution, different steps of one specified scenario may not be executed on the same thread, which is illustrated in Fig. 3.13 (b).

ii. Kernel Functions

In this work, n_c scenarios are solved simultaneously in a step-by-step pattern, with each step corresponding to one kernel. Fig. 3.14 demonstrates the implementation scheme of



(a) Integrated kernel: all steps are included in one whole kernel



(b) Decoupled kernel: different steps are performed by various kernels

Figure 3.13: Execution pattern of integrated kernel and decoupled kernels.

key steps of CM, where both self-built and `cusparse` library provided kernels are utilized. It should be noted that, every operation shown in Fig. 3.14 will be executed for n_c times with one for each scenario. As indicated in Section 3.3.1, all scenarios derive the solutions from the base case, i.e., triangle matrices $\boldsymbol{L}'$ and $\boldsymbol{U}'$ are the same for all scenarios. Therefore, equations $\boldsymbol{T}_a = \boldsymbol{L}'^{-1}\boldsymbol{M}'_*$ for all scenarios can be jointly considered as a sparse triangular linear system with multiple right-hand sides. Library `cusparse` provides efficient solution routine for this kind of systems. Kernel `cusparseDcsrsm_analysis()` performs the matrix analysis of $\boldsymbol{L}'$ and $\boldsymbol{U}'$. This function needs to be executed only once since its result is reusable. Kernel `cusparseDcsrsm_solve()` is utilized to generate the result based on matrix analysis information.

For all `cusparse` kernels, the thread utilization mechanism is concealed. On the other hand, the thread organization of self-built kernel is tuned based on the target data storage pattern to achieve coalesced access. Take the `kernel_Update()` in Fig. 3.15 as an example, where $\boldsymbol{\theta}$ is stored scenario after scenario. If each scenario is intuitively performed with one thread, diversity will occur since threads 1 and 2 will access discrete addresses $\boldsymbol{\theta}_0$ and $\boldsymbol{\theta}_n$ at the same time. Therefore, each scenario is attributed to one warp in Fig. 3.15, where

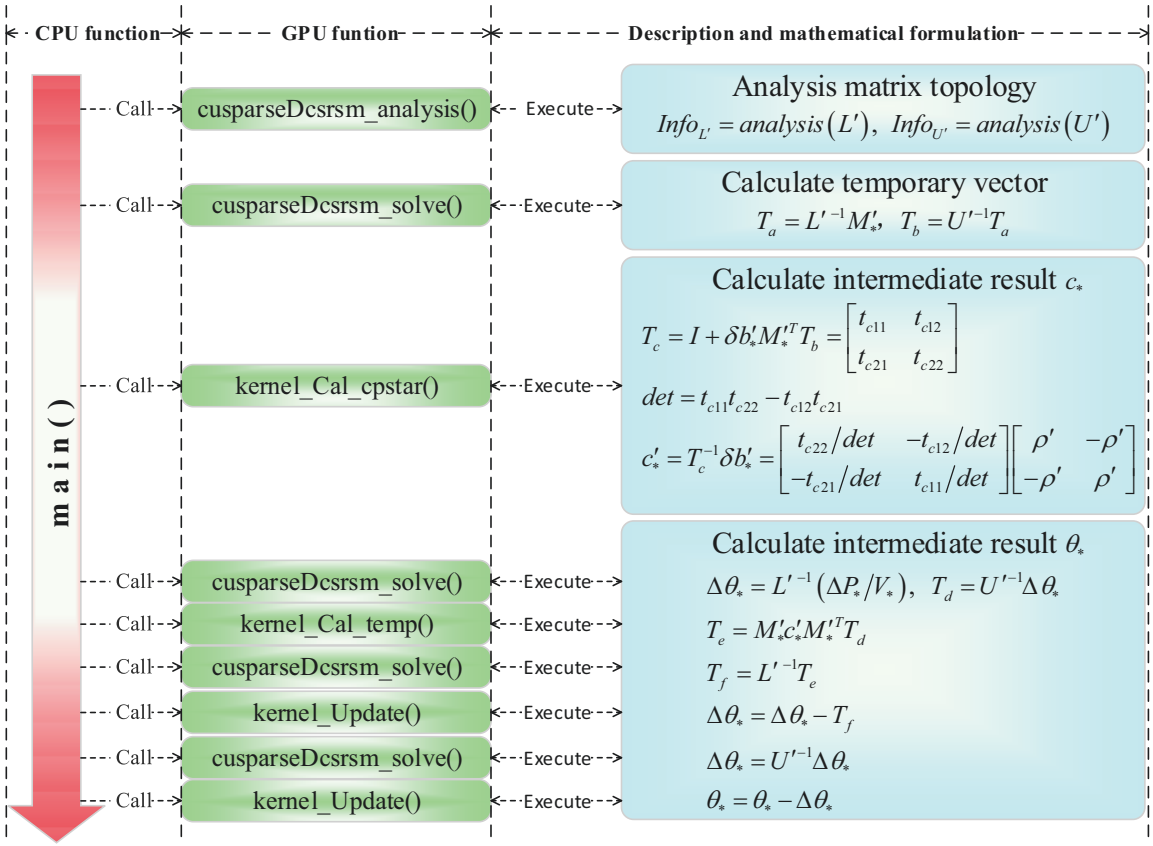


Figure 3.14: Decoupled kernels for the solution of equations (3.20)–(3.23).

lanes 0 – 31 will access a series of successive addresses $\boldsymbol{\theta}_0 - \boldsymbol{\theta}_{31}$. Different warps are independent of each other, and the branch divergence over warps does not affect performance.

ii. Memory Management

CUDA threads may access data from on-chip (register, shared memory, and L2 cache) and off-chip memories (global memory). The former is very fast but the size is limited, while the latter is large but more latency is required for accessing. To pursue a balance between the performance and feasibility, both are utilized with principles:

- To achieve the best performance of on-chip memory, all the small size intermediate vectors are stored as scalar variables. Take T_c in Fig. 3.14 as an example, instead of storing a matrix or vector, all the elements are declared as independent variables.
- All the long vectors and large matrices are stored in global memory, which can be accessed by every kernel for reading and writing. The maintenance of data in device memory is also beneficial to minimize the data exchange between host and device memories.
- Shared memory is widely utilized due to its low access latency, especially for the

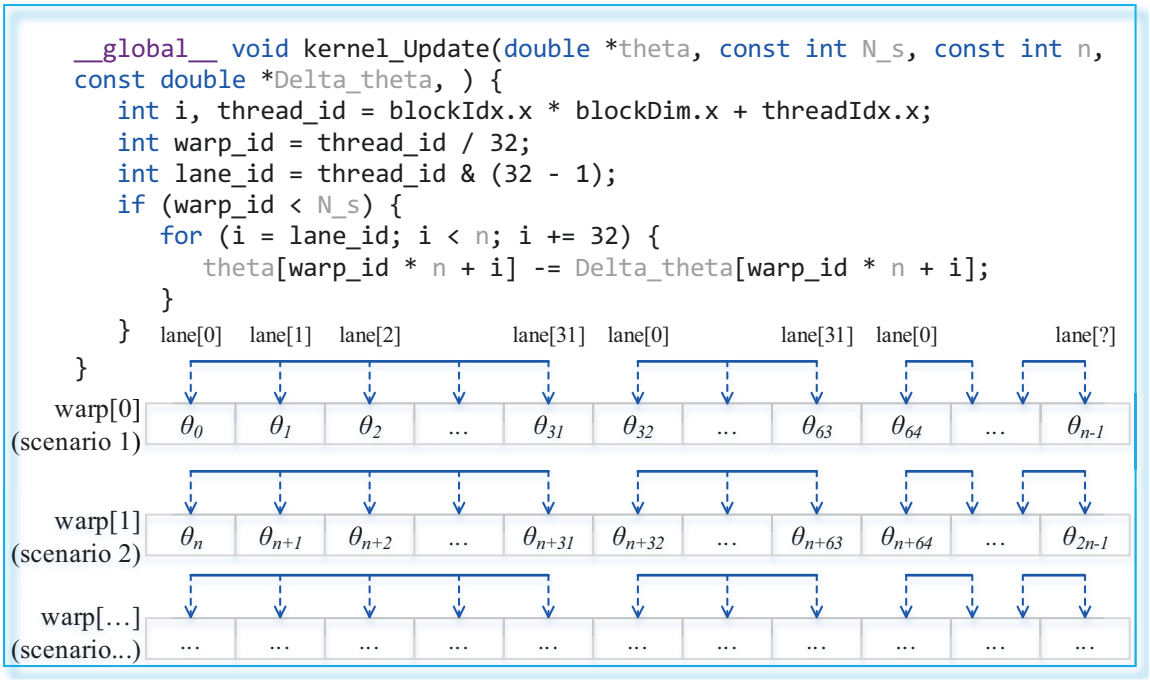


Figure 3.15: Coalesced access of θ in `kernel_Update()`.

reduction operations, including the calculation of summation, minimum, and maximum values of a vector. For example, vector $\Delta \boldsymbol{P}$ is copied into the shared memory before calculating $||\Delta \boldsymbol{P}||_\infty$.

As indicated in Fig. 3.14, the CPU main function calls the GPU kernels but cannot get any feedback from them. The common strategy of controlling is data exchange between CPU and GPU via `cudaMemcpy()`. However, due to the limited bandwidth of PCIe, fewer data transformations are better. In this work, except for the data to start CM, which is copied from CPU to GPU at the beginning, only one boolean variable (showing that the termination condition has been met or not) is transferred from GPU to CPU at the end of each iteration.

3.3.2.4 Multiple-GPU Architecture

In order to further accelerate the computation, multiple-GPU architecture is also explored. As the scenarios are solidly independent, there is no communication between different devices; therefore, the implementation is relatively straightforward, i.e., evenly distribute the workload to all the devices. Each device is administrated by one CPU thread, which is launched by OpenMP in this work.

Table 3.6: Solution differences between parallel CM and Matpower NR and FD.

Cases	Names and scales in [143]	$\Delta = \max\{\|\Delta P\|, \|\Delta Q\|\}$	
		CM vs. NR	CM vs. FD
CaseA	`case`**`300`**	6.16×10^{-9}	2.16×10^{-14}
CaseB	`case`**`1354`**`pegase`	6.79×10^{-9}	8.33×10^{-13}
CaseC	`case`**`2746`**`wp`	2.85×10^{-9}	9.99×10^{-13}
CaseD	`case`**`9241`**`pegase`	4.93×10^{-9}	7.61×10^{-13}
CaseE	`case`**`13659`**`pegase`	2.38×10^{-9}	1.09×10^{-13}

3.3.3 Experimental Results

Five benchmark cases reported in [143] with scales ranging from 300 to 13,659 buses are employed in this section to validate the performance of the parallel CM with respect to accuracy and execution time. Based on the open source package Matpower 6.0b2 [143], the first test is provided to validate the accuracy and convergence properties of CM. The second test measures the execution time and speedup of CM in different platforms (CPU and GPU) with various implementation schemes (sequential and parallel). Finally, three types of state-of-the-art GPU-based parallel computing methods reported in the literature are included for discussion. Both CPU- and GPU-based CM are implemented with Visual Studio 2015 on a PC equipped with 12 physical Intel Xeon E5-2620 2.10GHz CPU cores and 2 NVIDIA GTX 1080 GPUs, running on Windows 8.1 operating system. Matlab version 2015b is utilized to execute Matpower. For all experiments, the convergence criteria ϵ is set as 10^{-8} p.u.

3.3.3.1 Accuracy and Convergence of GPU-based Parallel CM

For any RTCA solution method, the credibility is of higher priority over efficiency. Thus the accuracy and convergence properties of GPU-based parallel CM are evaluated in this subsection. Matpower is introduced as a reference, where both NR and FD algorithms are utilized. For each test system, the same scenario is solved with three different methods. Table 3.6 summarizes the maximum differences between them on the node active/reactive powers $\max\{|\Delta P|, |\Delta Q|\}$. The data of column 3 is determined by the tolerance ϵ, while column 4 shows that the parallel CM is highly identical with Matpower FD. Therefore, it can be concluded that the proposed parallel implementation of CM based on GPU is credible.

Fig. 3.16 illustrates the convergence properties of parallel CM and NR on different cases, where parabola and straight lines can be approximated for the NR and CM on the logarithmic coordinate due to their quadratic and geometric convergent properties respectively. It should be noted that only the maximum error after Q iteration is collected for

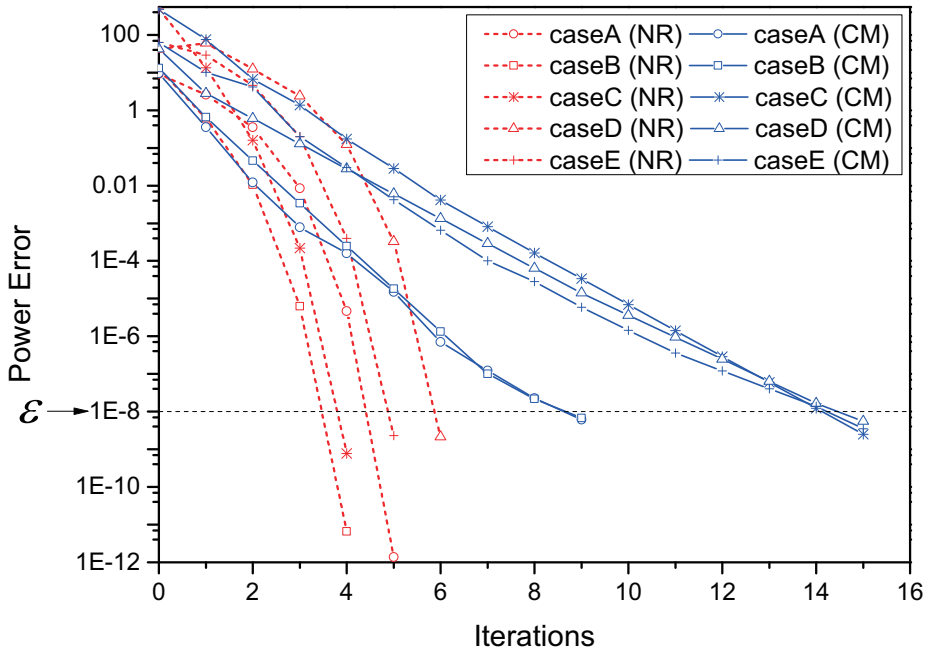


Figure 3.16: Convergence properties of FD and parallel CM on different cases.

the lines of CM. If maximum errors after both P and Q iterations are recorded, a wavy line with the same trend will appear. Although the CM takes more iterations to meet the convergence criteria, its computing requirement and execution time is far less than that of the NR. The comparison on the convergence process between CM and FD is omitted since they always take the same number of iterations before termination, which can also be accessed from the theoretical analysis in Section 3.3.1.

3.3.3.2 Performance of CM on Various Parallel Architectures

In order to fully explore the potential of CM and parallel architectures, many implementation schemes are tested and compared. For each case, n_c different scenarios generated by the withdrawing of a single transmission line are considered and solved, i.e., the $N-1$ contingency criterion is addressed without any scenario reduction strategy. Table 3.7 illustrates the values of n_c for various cases. Due to the large variance of n_c, the total execution time for RTCA with the solution of all scenarios presents great differences across cases. Therefore, it is regulated by the division of n_c in the following, i.e., the henceforth reported time is the average execution time for the full iterative solution of single scenario.

All CM implementation schemes to be presented below follow the flowchart shown in Fig. 3.9, where the LU decomposition for $\boldsymbol{B}'$ and $\boldsymbol{B}''$ at the very beginning is performed based on algorithms developed in [79]. The main differences between various implementation schemes are: 1) all considered scenarios will be evaluated in either sequential or

Table 3.7: Number of scenarios considered for different cases.

Cases	CaseA	CaseB	CaseC	CaseD	CaseE
n_c	411	1,991	3,514	16,049	20,467

Table 3.8: Execution time (ms) of sequential CM with single-thread CPU.

Cases	CaseA	CaseB	CaseC	CaseD	CaseE
T_0	2.514	12.127	40.382	157.671	220.976

Table 3.9: Execution time (ms) and speedup of parallel CM with multi-thread CPU.

Cases	CaseA	CaseB	CaseC	CaseD	CaseE
$T_{1,2}$	1.386	6.627	21.971	82.897	115.573
$T_{1,4}$	0.768	3.631	11.976	43.604	60.508
$T_{1,8}$	0.453	2.102	6.831	23.689	32.382
$T_{1,12}$	0.391	1.739	5.571	17.877	23.977
$S_{1,2}$	1.81	1.83	1.84	1.90	1.91
$S_{1,4}$	3.27	3.34	3.37	3.62	3.65
$S_{1,8}$	5.54	5.77	5.91	6.66	6.82
$S_{1,12}$	6.43	6.97	7.25	8.82	9.22

parallel, which will be specified subsequently; 2) on GPU platform, the F/B substitution is performed with `cusparse` kernel functions presented in Section 3.3.2.3; 3) on CPU platform, the F/B substitution is executed based on the routines provided by [79]. It should be noted that the basic mechanism and program logic of other steps are the same for both CPU and GPU versions, but the realized codes are different due to special GPU implementation strategies. Take the updating of $\boldsymbol{\theta}$ as an example, the code for GPU execution is shown in Fig. 3.15, where a lot of indices are included for threads, lanes, and warps.

i. Sequential CM with Single-Thread CPU

In this test, the single-thread CPU computing architecture is utilized, where all scenarios are evaluated in series with CPU. The execution time T_0 is reported in Table 3.8 with the coverage of all steps after the initial data reading, including admittance matrix generation, LU decomposition, compensation matrices construction, F/B substitution, and power mismatch calculation, etc. This test is determined as the reference for speedup analysis of other tests since it is the basic version of CPU implementation. The speedup S is defined as follows:

$$S_x = \frac{T_0}{T_x}, \tag{3.25}$$

where subscript $_x$ represents the following tests.

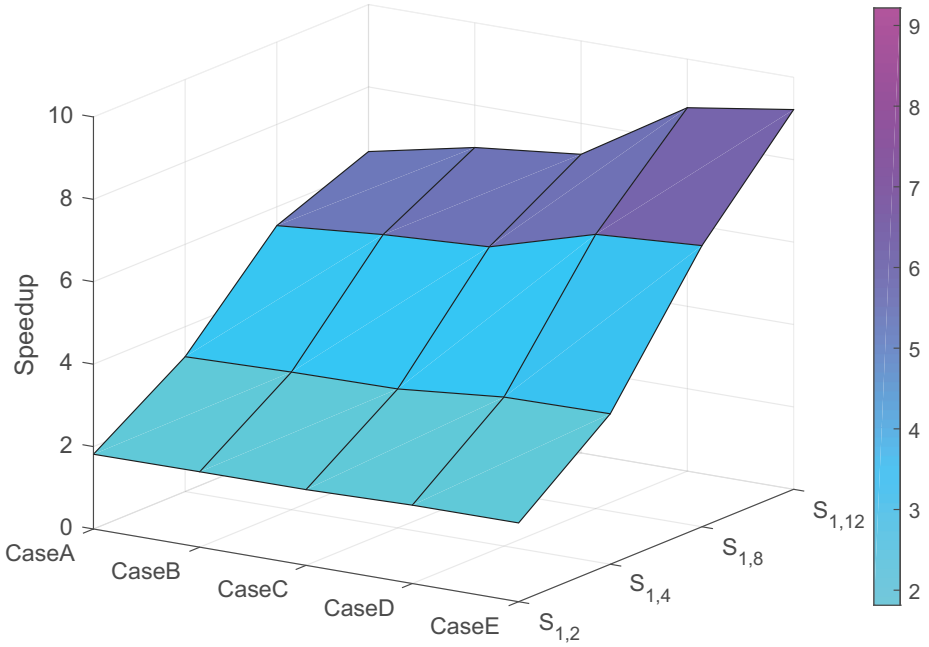


Figure 3.17: Speedup of parallel CM with multi-thread CPU.

ii. Parallel CM with Multi-Thread CPU

In order to accelerate the solution process of RTCA, multi-thread CPU is employed in this test, where all scenarios are assessed in parallel with CPU based on OpenMP. The CPU part shown in Fig. 3.9 is performed with the main thread, and the GPU part is fulfilled by y threads for concurrent execution. All scenarios are evenly distributed to y threads. In this test, y takes the values of 2, 4, 8, and 12 with the PC specified above. The execution time $T_{1,y}$ and speedup $S_{1,y}$ are summarized and illustrated in Table 3.9 and Fig. 3.17. Compared with T_0, a speedup is achieved with parallel computing. Two observations can be obtained: 1) For every y, $S_{1,y}$ is higher with larger cases. The parallel efficiency depends on both task workload and thread launch latency. If the latency is fixed, heavier work for each launch brings higher parallel efficiency. Thus, larger cases have higher speedups. 2) For each case, $S_{1,y}$ rises with y, but the increase rate is decreasing. The biggest performance gain is obtained by the thread number increasing from 1 to 2. More threads bring complex coordination challenge, thus the parallel efficiency is lower although the absolute speedup value is higher.

iii. Parallel CM with Single GPU

Due to the limited memory bandwidth, the saturation point of multi-thread CPU implementation is approaching, which means adding more CPU cores does not result in remarkable enhancement on the computational performance. Therefore, the GPU is introduced as an alternative. Based on the flowchart shown in Fig. 3.9 and details revealed in Section III.C, two tests are implemented, i.e., single GPU with integrated and decoupled kernels,

Table 3.10: Execution time (ms) and speedup of CM with single GPU.

Cases	CaseA	CaseB	CaseC	CaseD	CaseE
$T_{2,I}$	1.002	1.358	2.749	7.455	9.804
$T_{2,D}$	0.899	1.118	2.069	4.548	5.750
$S_{2,I}$	2.51	8.93	14.69	21.15	22.54
$S_{2,D}$	2.80	10.85	19.52	34.67	38.43

whose execution time are marked as $T_{2,I}$ and $T_{2,D}$ respectively. In addition to all steps covered by T_0 and $T_{1,y}$, the device memory allocation and data transmission between CPU and GPU are included in $T_{2,I}$ and $T_{2,D}$. As shown in Fig. 3.9, all the iterative processes within each scenario are executed on GPU without the data exchange with CPU. The data copy from CPU to GPU is mainly done before the launch of GPU, where grid configuration and decomposed matrices ($\boldsymbol{L}'$, $\boldsymbol{U}'$, $\boldsymbol{L}''$, and $\boldsymbol{U}''$) are transferred to device memory. The feedback from GPU to CPU is only one binary vector with the length of n_c, where '1' represents the corresponding scenario is sufficient, and vice versa. Therefore, the communication time T_c mainly depends on the system scale (which dominates the size of matrices), whereas n_c has a limited influence on T_c. In our tests, T_c is always less than 5% of the total execution time. Table 3.10 summarizes the main results. It is observable from $S_{2,I}$ and $S_{2,D}$ that the proposed decoupled kernel performs better than integrated kernel for all cases. In addition, $S_{2,D}$ has a better scalability from CaseA to CaseE since the increase rates are higher, such as $\frac{34.67}{2.80} > \frac{21.15}{2.51}$ and $\frac{38.43}{2.80} > \frac{22.54}{2.51}$. The fastest multi-thread CPU execution is compared with GPU implementations in Fig. 3.18, which validates the superiority of GPU parallel architecture for RTCA.

iv. Parallel CM with Multiple GPUs

Based on $T_{2,D}$, the total execution time of $N-1$ RTCA for CaseD and CaseE is 72.99s and 117.69s, respectively. In order to finish the solution within one minute, two-GPUs architecture is employed for acceleration. The implementation scheme is similar to single GPU, except that all scenarios are evenly distributed to two GPUs. Each GPU is managed with one CPU thread, and the CPU threads are separated with OpenMP. Since the CPU threads are launched for only once, the latency is insignificant when compared with the total execution time. Table 3.11 reports the results, where a maximum of 75.70$\times$ speedup is gained by the decoupled kernel strategy, which facilitates the RTCA to complete within one minute. In order to investigate the parallel efficiency of multiple GPUs, comparison with single GPU is demonstrated in Fig. 3.19. It is noticeable that both $S_{3,I}/S_{2,I}$ and $S_{3,D}/S_{2,D}$ are close to 2.0, which means the scalability of multiple GPUs is satisfactory. The main reasons are: 1) all scenarios are independent, such that there is no communication and synchronization between two GPUs; 2) the iterative process of ACPF is fully executed on GPU without the data exchange between CPU and GPU.

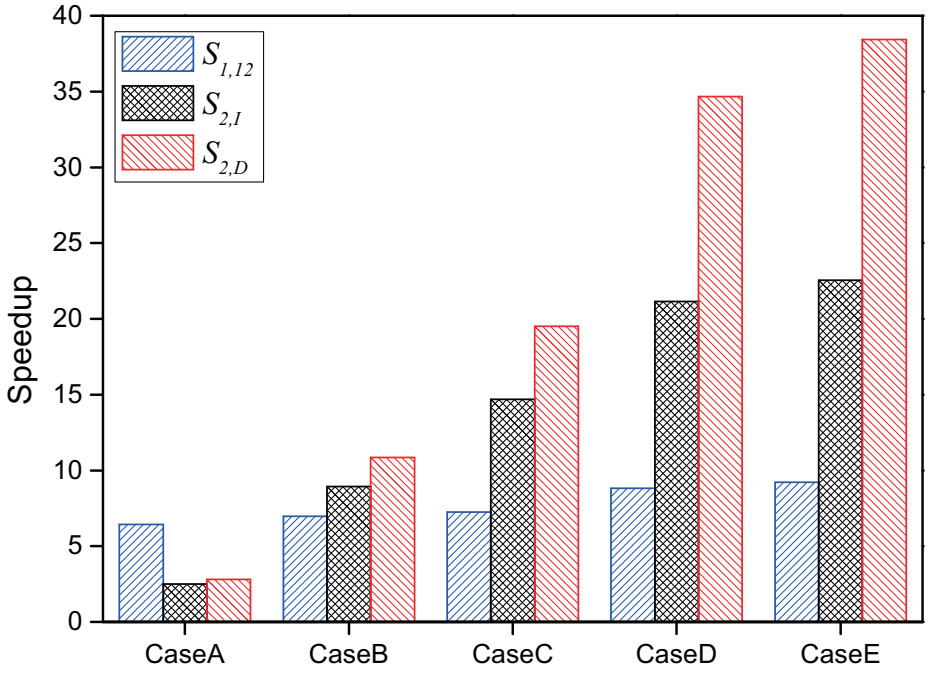


Figure 3.18: Speedup comparison between multi-thread CPU and GPU.

Table 3.11: Execution time (ms) and speedup of CM with multiple GPUs.

Cases	CaseA	CaseB	CaseC	CaseD	CaseE
$T_{3,I}$	0.553	0.733	1.468	3.895	5.077
$T_{3,D}$	0.473	0.582	1.072	2.320	2.919
$S_{3,I}$	4.55	16.54	27.51	40.48	43.52
$S_{3,D}$	5.32	20.84	37.67	67.96	75.70

3.3.3.3 Comparison with Other Parallel Computing Methods

The superiority of parallel CM running on GPUs is established in the above subsection with the comparison of CM running on CPU. This subsection is devoted to the comparison against three types of state-of-the-art parallel computing methods running on GPU.

i. Comparison with Parallel Gauss-Seidel (GS) Method

In [72, 73, 76], the GS was implemented on GPU for power flow analysis, whose solution time is summarized in Table 3.12. Although larger cases have been reported in [76] and [72], the performance is far away from [73] due to the utilization of dense matrix. The maximum case given in [73] is the 300-bus system, which consumes 56.293ms for each scenario, while that number is only 0.473ms for parallel CM in this work. Therefore, the advantage of CM over GS is identified.

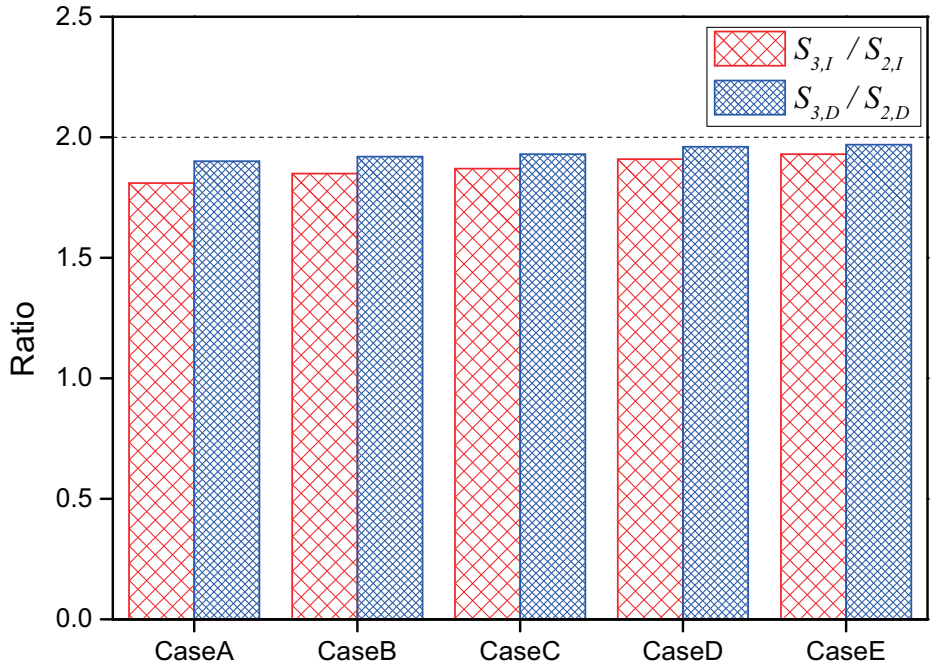


Figure 3.19: Speedup ratio of multiple GPUs over single GPU.

Table 3.12: Runtime reported in the literature with GS running on GPU.

System Scales	Ref. [76]	Ref. [72]	Ref. [73]
4-bus			0.012
9-bus	22.397	327.600	
30-bus	22.551	705.100	0.511
118-bus	50.719	3,296.300	2.346
300-bus	118.519	7,299.200	56.293
678-bus	154.513		
974-bus		19,603.000	
2,383-bus	826.617		
3,061-bus	3,061.353		

ii. Comparison with Parallel NR Method

Although the combination of NR with GPU has also been introduced in [72, 76, 77], only the high-quality results reported by [73–75] are included in Table 3.13 for comparison. The data in Table 3.13 has been illustrated in Fig. 3.20. It is obvious that the parallel CM consumes less time for all scales of cases, including small, medium, and large.

iii. Comparison with Parallel FD Method

In [72], the performance of FD on GPU is better than GS and NR; however, it is limited due to the utilization of dense matrix. [78] implements the FD on GPU with preconditioned conjugate gradient iterative solver and inexact Newton method. Although the reported

Table 3.13: Runtime reported in the literature with NR running on GPU.

System Scales	Ref. [73]	Ref. [74]	Ref. [75]	This work
4-bus	0.012			
30-bus	0.052			
118-bus	0.142			
300-bus	0.632			0.473
1,354-bus				0.582
1,586-bus		0.614*		
2,228-bus		1.104*		
2,383-bus	9.010			
2,746-bus				1.072
8,503-bus			5.471*	
9,241-bus				2.320
12,404-bus		4.026*		
13,659-bus				2.919
21,801-bus		4.767*		

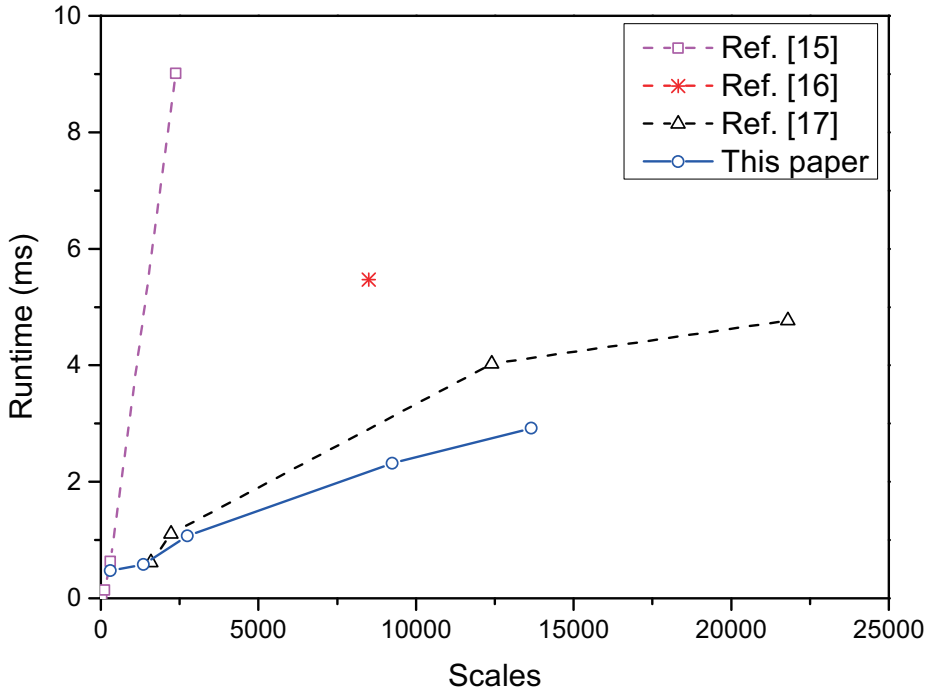


Figure 3.20: Runtime reported in the literature with NR running on GPU.

execution time is 260 $\sim$ 1,210ms for cases with scales 1,354 $\sim$ 13,173, it is a worthwhile endeavor (especially for large-scale systems) since the majority of works rely on the direct solver.

3.4 Summary

This chapter devotes to the acceleration of single and multiple ACPFs for the optimal operation of transmission system.

Based on the algorithmic analysis of NR and FD, the single scenario ACPF is addressed with FD and direct linear solver. In order to obtain further acceleration performance, GPU implementations with Matlab and CUDA are explored with various data storage formats and fill-in reduction algorithms. Case studies validate the efficiency of direct linear solver and FD method. The superiority of CUDA over Matlab with GPU is established.

RTCA is an important energy management system functionality in many electric utilities, it is faced with the challenge of high computational burden since multiple ACPFs need to be solved in limited time. To address this issue, the parallel implementation of CM on multiple-GPU architecture is investigated. Accuracy and efficiency have been validated with case studies on five benchmark systems ranging from 300 to 13,659 buses. The generated speedups are significant when compared with Matpower and sequential CM. In addition, superiority over other state-of-the-art parallel computing methods is observed. In conclusion, the parallel CM is promising for industrial application of RTCA with the capability of analyzing 20,556 scenarios for the 13,659-bus system.

4

Generation System Optimal Operation: SCUC and RTOPF

4.1 Introduction

All power in the grid is provided by the generator, thus its optimal operation is of great significance for the whole power system on the aspects of both economy and security. Since generators cannot instantly turn on and produce power, on-off operation schedule must be planned in advance so that enough generation is always available under various predefined contingency scenarios, such as generators or transmission lines go out or load demand increases. The decision process is summarized as SCUC and conducted over a certain time period, such as 24h. Due to the discrete on and off status, the SCUC presents great challenges on execution time and memory space, especially for large-scale systems. For a fixed time-stamp, decisions on the output of each generator to minimize the total generation cost should be made with respect to power equilibrium. This is a real-time decision process where forecasting error is inevitable with the remarkable penetration of intermittent Renewable Energy Generators (REGs).

Unexpected outage of power grid components, which is mainly triggered by extreme weather events or other random factors, can result in dramatic electricity shortages or even large-scale blackouts. For the purpose of withstanding a specified level of destruction, the $n - K^G - K^L$ security criterion is employed for SCUC formulation, which furnishes the system (n components) with the capability of surviving from the sudden unavailability of K^G generation units and K^L transmission lines. RO is introduced for the solution of SCUC, whose two-stage decomposition framework is implemented with implicit and explicit methods. Different types of acceleration strategies and parallel computing potentials

are investigated to demonstrate the promising of RO for SCUC.

RTOPF determines the optimal output of each generator to minimize the operation cost in real-time. To improve the solution efficiency and accuracy of RTOPF, a three-stage framework for parallel processing with GPU is employed in this chapter. In Stage 1, uncertainties from renewable generators and demand loads are characterized with scenarios. Large numbers of RTOPFs corresponding to each scenario are formulated and addressed in Stage 2, where the linear systems are regulated into the same sparsity pattern and then tackled in a batched style with the GPU. Results from Stage 2 are utilized in Stage 3 to perform a hot-start RTOPF, where the forecasting error can be minimized.

4.2 Security Constrained Unit Commitment

This section intends to reveal the capability of explicit and implicit decomposition frameworks of RO for the solution of SCUC. In section 4.2.1, problem formulation of SCUC is given. Detailed methodology development for RO is presented in section 4.2.2, as well as their inner feedback strategies, such as Benders decomposition and Column-and-Constraint Generation (CCG) algorithm. Sensitivity analysis, multi-cut strategy, and parallel implementation have also been analyzed and discussed in section 4.2.3 with case studies.

4.2.1 Problem Formulation

Based on [20], [85], and [144], the SCUC can be given as:

$$\min_{v_g,\, c_g^u,\, c_g^d,\, Q^w,\, c_g^{p(0)},\, r_i^{(0)},\, p_g^{(0)},\, \theta_i^{(0)},\, f_{ij}^{(0)}} \left\{ \sum_{t\in\mathcal{T}} \sum_{g\in\mathcal{N}_g} \left(c_g^{p(0)}(t) + c_g^u(t) + c_g^d(t) \right) + Q^w \right\}, \tag{4.1}$$

$$s.t. \quad -v_g(t-1) + v_g(t) - v_g(h) \le 0, \qquad \forall g \in \mathcal{N}_g, \forall t \in \mathcal{T}, \forall h : 1 \le h-(t-1) \le T_g^U \tag{4.2}$$

$$v_g(t-1) - v_g(t) + v_g(h) \le 1, \qquad \forall g \in \mathcal{N}_g, \forall t \in \mathcal{T}, \forall h : 1 \le h-(t-1) \le T_g^D \tag{4.3}$$

$$c_g^u(t) \ge C_g^U \left(v_g(t) - v_g(t-1) \right), \qquad \forall g \in \mathcal{N}_g, \forall t \in \mathcal{T} \tag{4.4}$$

$$c_g^d(t) \ge C_g^D \left(v_g(t-1) - v_g(t) \right), \qquad \forall g \in \mathcal{N}_g, \forall t \in \mathcal{T} \tag{4.5}$$

$$c_g^{p(0)}(t) = A_g v_g(t) + B_g p_g^{(0)}(t), \qquad \forall g \in \mathcal{N}_g, \forall t \in \mathcal{T} \tag{4.6}$$

$$c_g^u(t), c_g^d(t) \ge 0, v_g(t) \in \{0,1\}, \qquad \forall g \in \mathcal{N}_g, \forall t \in \mathcal{T} \tag{4.7}$$

$$\left(p_g^{(0)}, f_{ij}^{(0)}, r_i^{(0)}, \theta_i^{(0)} \right) \in X^{(0)} \tag{4.8}$$

$$Q^w = \max_{s\in S} \left\{ \min_{p_g^{(s)},\, f_{ij}^{(s)},\, r_i^{(s)},\, \theta_i^{(s)}} \sum_{t\in\mathcal{T}} \sum_{i\in\mathcal{N}_b} Pr_i^{(s)}(t) \right\} \tag{4.9}$$

$$\left(p_g^{(s)}, f_{ij}^{(s)}, r_i^{(s)}, \theta_i^{(s)} \right) \in X^{(s)}, \forall s \in \mathcal{S} \tag{4.10}$$

where

$$
\begin{aligned}
X^{(s)} = \Big\{ & \big(p_g^{(s)}, f_{ij}^{(s)}, r_i^{(s)}, \theta_i^{(s)}\big): \\
& z_g^{(s)} v_g(t) \underline{P}_g \le p_g^{(s)}(t) \le z_g^{(s)} v_g(t) \bar{P}_g, && \forall g \in \mathcal{N}_g, \forall t \in \mathcal{T} && (4.11)\\
& -z_{ij}^{(s)} \bar{f}_{ij} \le f_{ij}^{(s)}(t) \le z_{ij}^{(s)} \bar{f}_{ij}, && \forall (i,j) \in \mathcal{N}_l, \forall t \in \mathcal{T} && (4.12)\\
& f_{ij}^{(s)}(t) = \frac{z_{ij}^{(s)}}{x_{ij}} (\theta_i^{(s)}(t) - \theta_j^{(s)}(t)), && \forall (i,j) \in \mathcal{N}_l, \forall t \in \mathcal{T} && (4.13)\\
& p_g^{(s)}(t) - p_g^{(s)}(t-1) \le (2 - v_g(t-1) - v_g(t)) \underline{P}_g + (1 + v_g(t-1) - v_g(t)) R_g^U, \\
& && \forall g \in \mathcal{N}_g, \forall t \in \mathcal{T} && (4.14)\\
& p_g^{(s)}(t-1) - p_g^{(s)}(t) \le (2 - v_g(t-1) - v_g(t)) \bar{P}_g + (1 - v_g(t-1) + v_g(t)) R_g^D, \\
& && \forall g \in \mathcal{N}_g, \forall t \in \mathcal{T} && (4.15)\\
& -r_i^{(s)}(t) \le \sum_{\forall j \in \mathcal{N}_{l(\cdot,i)}} f_{ji}^{(s)}(t) - \sum_{\forall j \in \mathcal{N}_{l(i,\cdot)}} f_{ij}^{(s)}(t) + \sum_{g \in G_i} p_g^{(s)}(t) - D_i(t) \le r_i^{(s)}(t), \\
& && \forall i \in \mathcal{N}_b, \forall t \in \mathcal{T} && (4.16)\\
& p_g^{(s)}(t), r_i^{(s)}(t) \ge 0, \forall i \in \mathcal{N}_b, \forall g \in \mathcal{N}_g, \forall t \in \mathcal{T} \Big\}. && && (4.17)
\end{aligned}
$$

The minimizing objective function (4.1) comprises of pre-contingency operation cost and post-contingency power imbalance penalty. Pre- and post-contingency constraints are modeled by (4.2)–(4.8) and (4.9)–(4.10) respectively. Constraints (4.2) and (4.3) represent the restrictions on the minimum up and down time for units. Start-up and shut-down costs are modeled in (4.4) and (4.5) respectively. Nonnegative and binary constraints are stated in (4.7). Economic Dispatch (ED) problems for pre- and post-contingency are given by (4.8) and (4.10). (4.9) indicates that Q^w is the penalty for the worst-case of contingency.

For a fixed unit commitment decision $v_g(t)$ under any contingency scenario s, the ED is formulated as (4.11)–(4.17), where constraints on unit generation limit (4.11), transmission line capacity (4.12), power flow (4.13), ramping up/down limit (4.14)–(4.15), nodal power balance (4.16) are considered. (4.17) indicates the nonnegative constraints.

For simplicity, the piecewise linear approximation of the quadratic production cost function (4.18) is represented by (4.6), where parameters A_g and B_g are defined by (4.19) and (4.20).

$$c_g^p(t) = a_g v_g(t) + b_g p_g(t) + c_g p_g^2(t), \forall g \in \mathcal{N}_g, \forall t \in \mathcal{T}, \tag{4.18}$$

$$A_g = a_g - c_g \underline{P}_g \bar{P}_g, \forall g \in \mathcal{N}_g, \tag{4.19}$$

$$B_g = b_g + c_g \bar{P}_g + c_g \underline{P}_g, \forall g \in \mathcal{N}_g. \tag{4.20}$$

The contingency set S corresponding to the $n - K^G - K^L$ contingency criterion is

defined as,

$$\mathcal{S} = \left\{ (z_g, z_{ij}) \in \{0,1\} \;\middle|\; \begin{array}{l} \sum_{g\in G} z_g \geq |G| - K^G, \\ \sum_{(i,j)\in\mathcal{N}_l} z_{ij} \geq |L| - K^L \end{array} \right\}. \tag{4.21}$$

4.2.2 Solution Methodology

4.2.2.1 Decomposition Framework

The specific SCUC problem (4.1)–(4.10) can be reformulated as a compact form (4.22), with $\boldsymbol{c}^T\boldsymbol{x} + P\sum_{s=1}^{|S|} \boldsymbol{q}_s^T\boldsymbol{y}_s$, $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$, and $\boldsymbol{T}_s\boldsymbol{x} + \boldsymbol{W}_s\boldsymbol{y}_s \leq \boldsymbol{h}_s$ corresponds to (4.1) and (4.9), (4.2)–(4.8), and (4.10), respectively. Where $\boldsymbol{x}$ and $\boldsymbol{y}_s$ are the pre- and post-contingency decision variables; $\boldsymbol{c}$, $\boldsymbol{q}_s$, $\boldsymbol{A}$, $\boldsymbol{b}$, $\boldsymbol{T}_s$, $\boldsymbol{W}_s$, and $\boldsymbol{h}_s$ are coefficient matrices and vectors. The objective function (4.22) should be $\min_{\boldsymbol{x},\boldsymbol{y}_s} \left\{\boldsymbol{c}^T\boldsymbol{x} + P\max_{s\in S} \boldsymbol{q}_s^T\boldsymbol{y}_s\right\}$ in its original form, which is not suitable for the off-the-shelf MILP solver due to the $\min-\max$ programming. Since the final objective is eliminating the power imbalance, i.e., pursuing $\min\left\{\max_{s\in S} \boldsymbol{q}_s^T\boldsymbol{y}_s\right\} = 0$, which is equivalent with seeking $\min\left\{\sum_{s=1}^{|S|} \boldsymbol{q}_s^T\boldsymbol{y}_s\right\} = 0$ as $\boldsymbol{q}_s^T\boldsymbol{y}_s \geq 0$. Therefore, MILP problem (4.22) is deduced.

Although (4.22) is applicable for available solver, it may present great challenges or even be intractable when faced with large-scale systems due to large numbers of decision variables and constraints. Therefore, decomposition strategy is usually employed to reduce the problem size. Fig. 4.1 presents a general decomposition framework suitable for both explicit and implicit methods.

$$\begin{array}{lccccccccc} \min\limits_{\boldsymbol{x},\boldsymbol{y}_s} & \boldsymbol{c}^T\boldsymbol{x} & + & P\boldsymbol{q}_1^T\boldsymbol{y}_1 + P\boldsymbol{q}_2^T\boldsymbol{y}_2 \cdots + P\boldsymbol{q}_s^T\boldsymbol{y}_s & & & & & & \\ s.t. & \boldsymbol{A}\boldsymbol{x} & & & & & & \leq & \boldsymbol{b}, \\ & \boldsymbol{T}_1\boldsymbol{x} & + & \boldsymbol{W}_1\boldsymbol{y}_1 & & & & \leq & \boldsymbol{h}_1, \\ & \boldsymbol{T}_2\boldsymbol{x} & + & & \boldsymbol{W}_2\boldsymbol{y}_2 & & & \leq & \boldsymbol{h}_2, \\ & : & + & & & \cdots & & \leq & : \\ & \boldsymbol{T}_s\boldsymbol{x} & + & & & & \boldsymbol{W}_s\boldsymbol{y}_s & \leq & \boldsymbol{h}_s. \end{array} \tag{4.22}$$

4.2.2.2 Explicit Method

i. Subproblems

Explicit method formulate each realization of the uncertainty set S into a subproblem (4.23), where $z_g^{(s)}$ and $z_{ij}^{(s)}$ are fixed. By solving all of them, the most violated scenario can be determined.

$$\min_{p_g^{(s)},\, f_{ij}^{(s)},\, r_i^{(s)},\, \theta_i^{(s)}} \quad \sum_{t\in\mathcal{T}}\sum_{i\in\mathcal{N}_b} Pr_i^{(s)}(t) \tag{4.23}$$

$$s.t. \quad (p_g^{(s)},\, f_{ij}^{(s)},\, r_i^{(s)},\, \theta_i^{(s)}) \in X^{(s)}. \tag{4.24}$$

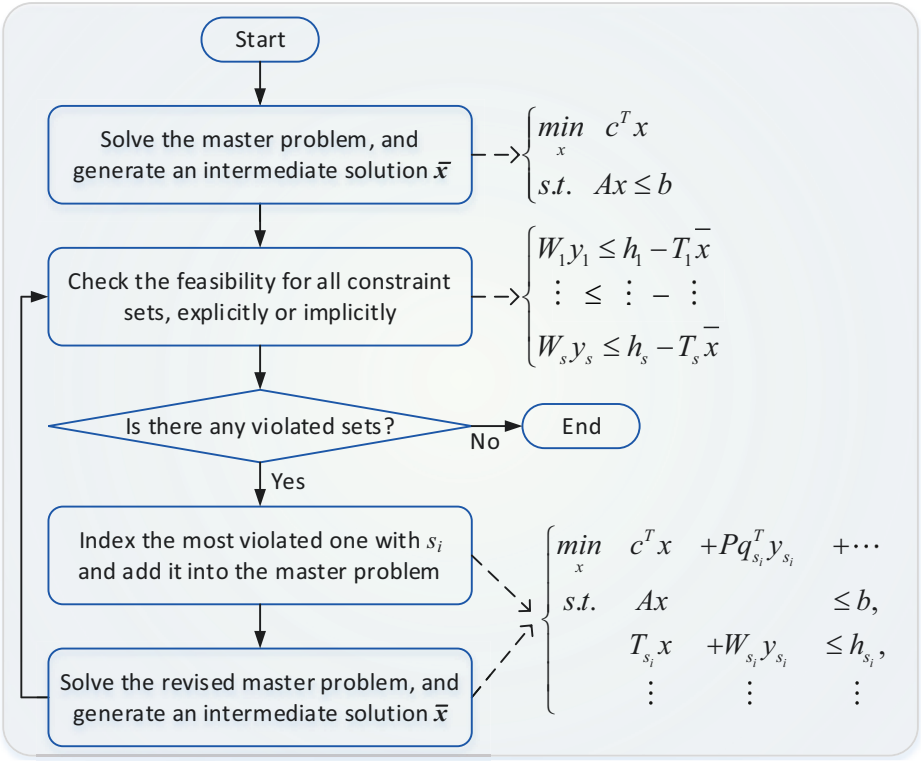


Figure 4.1: Decomposition framework for the solution of SCUC with CCG.

$$
\begin{aligned}
Q_{it}^{(s)} = \sum_{t\in\mathcal{T}} \Bigg[& \sum_{g\in G} \beta_g^{t+} z_g^{(s)} v_g(t) \bar{P}_g - \sum_{g\in G} \beta_g^{t-} z_g^{(s)} v_g(t) \underline{P}_g + \sum_{(i,j)\in\mathcal{N}_l} \tau_{ij}^{t+} z_{ij}^{(s)} \bar{f}_{ij} + \sum_{(i,j)\in\mathcal{N}_l} \tau_{ij}^{t-} z_{ij}^{(s)} \bar{f}_{ij} \\
& + \sum_{i\in I} \xi_i^{t+} D_i(t) + \sum_{g\in G} \eta_g^{t-} \left((2 - v_g(t-1) - v_g(t)) \underline{P}_g + (1 + v_g(t-1) - v_g(t)) R_g^U \right) \\
& - \sum_{i\in I} \xi_i^{t-} D_i(t) + \sum_{g\in G} \eta_g^{t+} \left((2 - v_g(t-1) - v_g(t)) \bar{P}_g + (1 - v_g(t-1) + v_g(t)) R_g^D \right) \Bigg].
\end{aligned} \tag{4.25}
$$

In order to deduce Benders cuts, the corresponding dual problem (4.25)–(4.30) should be generated.

$$
\begin{aligned}
\max \quad & Q_{it}^{(s)}(\beta, \tau, \zeta, \eta, \xi \mid v^*) && \text{see (4.25)} \\
s.t. \quad & \beta_g^{t+} - \beta_g^{t-} + \eta_g^{t-} - \eta_g^{(t+1)-} + \eta_g^{(t+1)+} - \eta_g^{t+} + \sum_{\mathcal{G}_i \ni g} \xi_i^{t+} - \sum_{\mathcal{G}_i \ni g} \xi_i^{t-} \le 0, \forall g \in G, \forall t \in \mathcal{T} && (4.26) \\
& \tau_{ij}^{t+} - \tau_{ij}^{t-} + \zeta_{ij}^t - \sum_{i\in(i,j)} \xi_i^{t+} + \sum_{j\in(i,j)} \xi_j^{t+} + \sum_{i\in(i,j)} \xi_i^{t-} - \sum_{j\in(i,j)} \xi_j^{t-} = 0, \forall (i,j) \in \mathcal{N}_l, \forall t \in \mathcal{T} && (4.27)
\end{aligned}
$$

$$\sum_{j\in\mathcal{N}_{l(i,\cdot)}} \frac{z_{ij}^{(s)}}{x_{ij}}\zeta_{ij}^{t} - \sum_{j\in\mathcal{N}_{l(\cdot,i)}} \frac{z_{ji}^{(s)}}{x_{ji}}\zeta_{ji}^{t} = 0, \forall i \in I, \forall t \in \mathcal{T} \tag{4.28}$$

$$-\xi_i^{t+} - \xi_i^{t-} \leq P, \forall i \in I, \forall t \in \mathcal{T} \tag{4.29}$$

$$\beta^{\pm}, \tau^{\pm}, \eta^{\pm}, \xi^{\pm} \leq 0, \zeta \text{ unrestricted.} \tag{4.30}$$

where the subscript $_{it}$ represents the number of iteration; β^-/β^+, τ^-/τ^+, and ξ^-/ξ^+ are dual variables for the left/right hand side of constraints (4.11), (4.12), and (4.16); ζ, η^-, and η^+ are dual variables for constraints (4.13), (4.14), and (4.15), respectively. Constraints (4.26), (4.27), (4.28), and (4.29) correspond to the variables $p_g^{(s)}(t)$, $f_{ij}^{(s)}(t)$, $\theta_i^{(s)}(t)$, and $r_i^{(s)}(t)$ in (4.11)–(4.16). The upper script * represents the fixed value, and henceforth.

For the worst scenario and all the other violated scenarios, a Benders optimality cut can be generated, which is shown in (4.31). The Benders feasibility cut is omitted since the subproblem is always feasible due to the slack variable $r_i^{(s)}(t)$.

$$Q^w \geq Q_{it}^{(s)}(v \mid \beta^*, \tau^*, \zeta^*, \eta^*, \xi^*), \forall s \in \mathcal{S}. \tag{4.31}$$

ii. Master Problem

Take the optimality cuts (4.31) into consideration, the master problem is described as:

$$\min_{v_g} \quad \sum_{t\in\mathcal{T}}\sum_{g\in G}\left(c_g^{p(0)}(t) + c_g^{u}(t) + c_g^{d}(t)\right) + Q^w \tag{4.32}$$

$$s.t. \quad \text{constraints } (4.2)\text{–}(4.8) \text{ and } (4.31). \tag{4.33}$$

iii. Bounds

The lower bound LB is the objective function value of master problem (4.32), and the upper bound UB is,

$$UB = \sum_{t\in\mathcal{T}}\sum_{g\in G}\left(c_g^{p(0)*}(t) + c_g^{u*}(t) + c_g^{d*}(t)\right) + \max_{s\in S}\left\{Q_{it}^{(s)}(v^*, \beta^*, \tau^*, \zeta^*, \eta^*, \xi^*)\right\}. \tag{4.34}$$

If the optimality criteria is met, i.e., $|UB - LB| \leq \epsilon$, then stop the process; otherwise, start a new iteration to solve the master problem.

4.2.2.3 Implicit Method

i. Subproblem

Implicit decomposition considers all the contingency scenario into one whole subproblem,

where $z_g^{(s)}$ and $z_{ij}^{(s)}$ are binary decision variables. The primal $\max$–$\min$ subproblem is:

$$Q^w = \max_{z_g, z_{ij}} \quad \Big\{ \Delta, \tag{4.35}$$

$$s.t. \quad (z_g, z_{ij}) \in \mathcal{S}, \tag{4.36}$$

$$\Delta = \min_{p_g^{(s)},\, f_{ij}^{(s)},\, r_i^{(s)},\, \theta_i^{(s)}} \quad \Big[\sum_{t \in \mathcal{T}} \sum_{i \in \mathcal{N}_b} Pr_i(t), \tag{4.37}$$

$$s.t. \qquad \text{(4.11)–(4.17) without } {}^{(s)}. \Big] \Big\} \tag{4.38}$$

The inner problem (4.37) has the same form with (4.23), therefore, its dual problem is similar with (4.25)–(4.30). By replacing (4.37) with (4.25)–(4.30), the $\max$–$\min$ subproblem turns into $\max$–$\max$, which can be equivalently rewritten as a maximizing problem. However, several bilinear terms emerged in objective function (4.25) and constraints (4.28) since z_g and z_{ij} are also decision variables, such as $\beta_g^{t+} z_g$. In order to linearize the problem, the bilinear term will be replaced with new variables, i.e., $B_g^{t+} = \beta_g^{t+} z_g$, and the following disjunctive constraints should be added.

$$-(1 - z_g)M \le B_g^{t+} - \beta_g^{t+} \le (1 - z_g)M, \tag{4.39}$$

$$-z_g M \le B_g^{t+} \le z_g M. \tag{4.40}$$

ii. Master Problem and Bounds

By solving the MILP subproblem, the worst contingency can be identified, then Benders cuts (4.31) can be generated. The remaining steps corresponding to the master problem and bounds are the same with the explicit method.

4.2.2.4 Acceleration Strategies

i. Parallel Computing

For the explicit method, all subproblems are independent and of the same scale, if parallel computing is utilized, the solution time for subproblems is almost inversely proportional to the number of processors [145]. On the other hand, there is only one MILP subproblem in the implicit method, which is not suitable for parallel implementation. In terms of MILP master problem, the parallel implementation can be worthwhile only if the calculation time for the subproblems can be reduced to the point where the master problem time becomes a significant fraction of the whole [146].

ii. Multi-cut Strategy

As shown above, each contingency scenario can generate one Benders cut and one con-

Table 4.1: Scales and complexity of benchmark test systems under $n-1-1$ contingency criterion.

Items	24-bus system	118-bus system
No. of buses:	24	118
No. of generators:	14	56
No. of transm. lines:	46	189
No. of scenarios:	645	10,585
No. of bin. variables:	336	1,344
No. of cont. variables:	1,889,233	136,422,169
No. of constraints:	3,378,347	230,945,121

straint set, containing information of the solution space, which benefits for the convergence process. If the complexity of the master is not considered, more cuts and constraints mean faster convergence. Therefore, a π-cut strategy can be proposed for the explicit method, which determines the most violated π realizations and adds corresponding cuts and constraints into the master problem.

4.2.3 Numerical Results and Discussion

In this section, all the numerical tests are performed on AMPL IDE 3.1.0 using CPLEX 12.6.3 solver on a PC running on 64-bit Windows 8.1 operating system, with quad-core Intel Xeon E5-2620 v2 CPU (2.1 GHz) and 32 GB RAM. Parameters P, M and ϵ are valued as 10^3, 10^3 and 10^{-3}.

4.2.3.1 Benchmark Systems

Extensive performance evaluation and sensitivity analysis are implemented on the IEEE 24-bus test system [147] to illustrate the efficacy of explicit and implicit methods. The resilience against multiple contingencies is increased by adding 4 generators and 5 transmission lines. As a consequence, the system is able to withstand the $n-3-0$ and $n-2-1$ contingency criteria. The IEEE 118-bus test system [132] is introduced to show the potential under large-scale circumstances, where 2 generators and 3 circuits are enhanced to meet the $n-1-1$ contingency criterion. Table 4.1 gives an overview of the scales and complexity of both systems.

In order to extensively compare the explicit and implicit methods, several algorithms are separated and shown in Table 4.2. The convergence behavior and time consumption is illustrated in Fig. 4.2 and Fig. 4.3, where a cutoff time of 1000s is employed. The global optimal objective value is 952,164$.

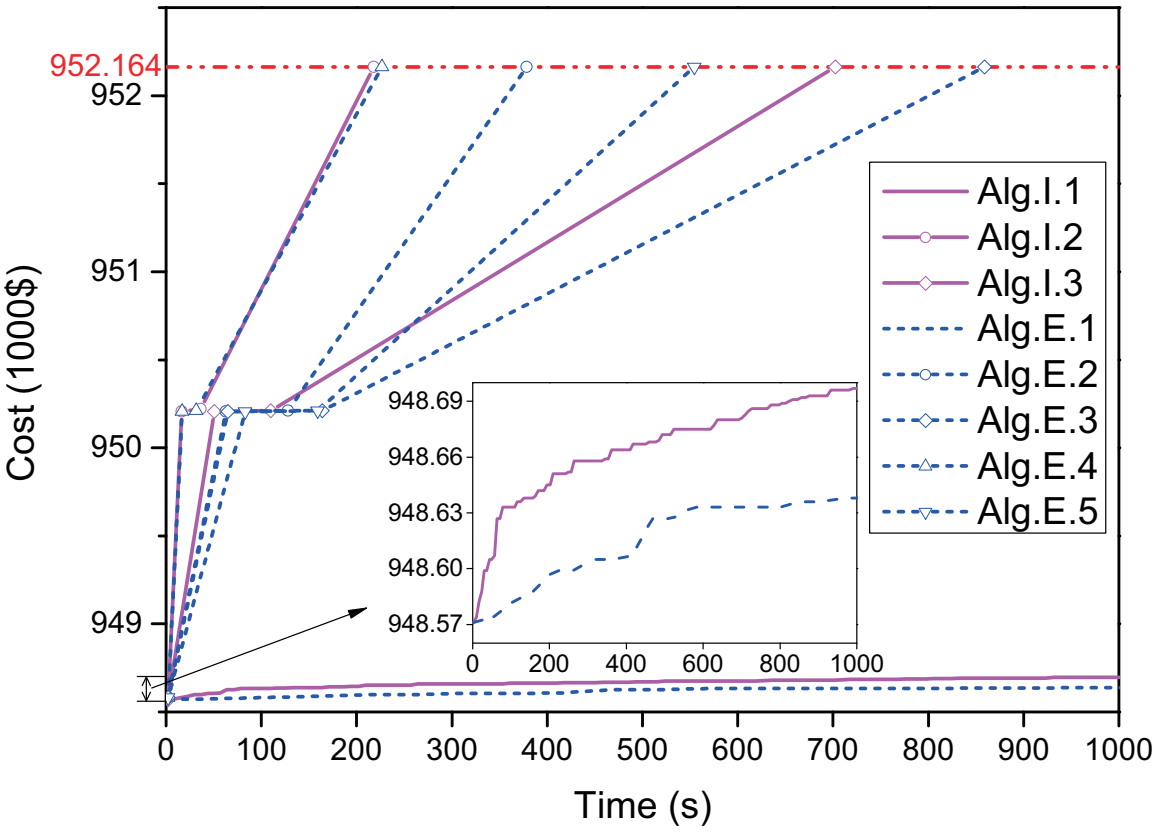


Figure 4.2: Behavior of convergence for different algorithms.

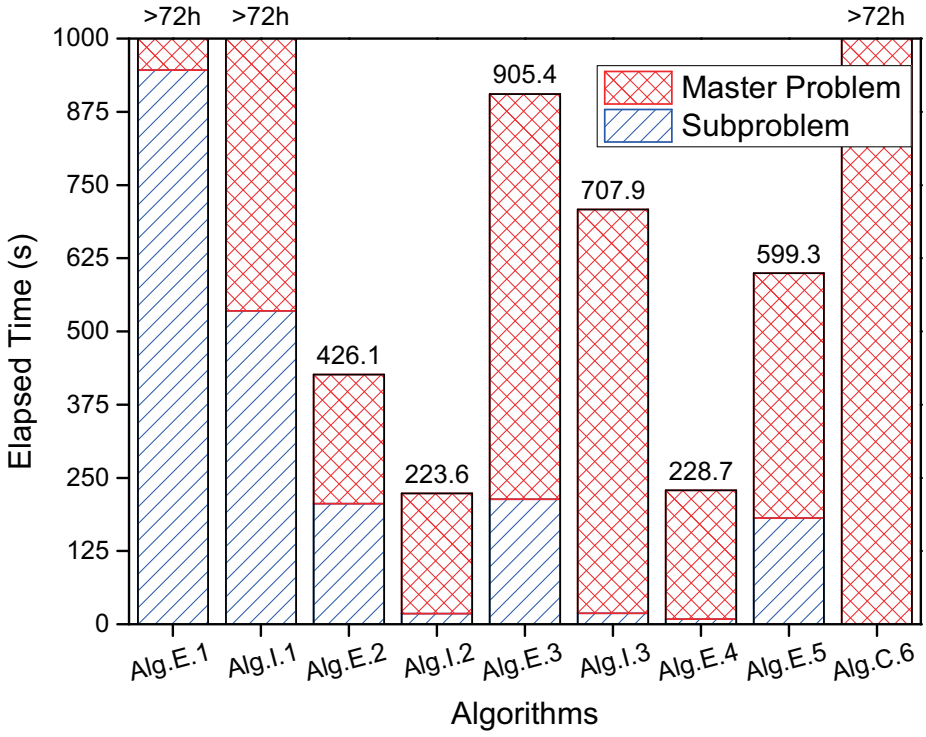


Figure 4.3: Behavior of time consumption for different algorithms.

Table 4.2: Alternative versions of algorithms.

Algorithms	Description
Alg.E.1/***Alg.I.1***:	Explicit/Implicit method with Benders cuts only.
Alg.E.2/***Alg.I.2***:	Explicit/Implicit method with constraint sets only.
Alg.E.3/***Alg.I.3***:	Explicit/Implicit method with both constraint sets and Benders cuts.
Alg.E.4:	***Alg.E.2*** with its subproblems solved in parallel by 24 threads.
Alg.E.5:	***Alg.E.2*** with multi-cut strategy.
Alg.C.6:	Commercial MILP solver CPLEX 12.6.3.

4.2.3.2 Performance Evaluation of Benders Cuts and Constraint Sets

According to Fig. 4.3, both ***Alg.E.1*** and ***Alg.I.1*** cannot terminate after a run time of 1000s, while ***Alg.E.2*** and ***Alg.I.2*** converge at 426.1s and 223.6s, indicating that CCG is more efficient than Benders cuts. On the other hand, ***Alg.E.3*** and ***Alg.I.3*** are even slower than ***Alg.E.2*** and ***Alg.I.2*** with both CCG and Benders cuts involved, which means the introduction of Benders cuts even drags the solution efficiency. The reason is that a slack variable $r_i^{(s)}(t)$ is required during the solution process of Benders decomposition, which expands the scale for both master problem and subproblems.

4.2.3.3 Performance Evaluation of Parallel Implementation

It is noticeable in Fig. 4.3 that the time consumed by subproblem is much higher in the explicit method, while the master problem consumes almost the same amount of time for both methods, especially for ***Alg.E.2*** and ***Alg.I.2***. Therefore, the parallel implementation is introduced in ***Alg.E.4*** based on ***Alg.E.2***. The solution time is reduced from 426.1s to 228.7s, which is comparable to ***Alg.I.2*** with 223.6s; thus the performance improvement gained by parallel computing is promising.

4.2.3.4 Performance Evaluation of Multi-cut Strategy

In ***Alg.E.5***, two sets of constraints are added in each iteration, i.e., $\pi = 2$. However, it spends more time than ***Alg.E.2*** although their subproblems spend the same time. Which means adding one more set of constraints only increases the size of the master problem, but does not enhance the convergence, i.e., the benefit of multi-cut strategy is not significant. One reason is that the two most violated scenarios are similar and one of them is redundant. Another reason lies in the small number of iterations before termination, which can be seen from Fig. 4.2.

Table 4.3: Computational results for different K^G and K^L values.

K^G	K^L	Cost ($)	Time (s)	K^G	K^L	Cost ($)	Time (s)
0	0	948,582	2.328	2	0	953,380	267.375
0	1	948,932	11.939	2	1	956,259	4,336.830
0	2	Infeasible	———	2	2	Infeasible	———
1	0	949,810	58.266	3	0	958,028	3,655.560
1	1	952,164	223.592	3	1	Infeasible	———
1	2	Infeasible	———	4	0	Infeasible	———

4.2.3.5 Performance Comparison between Explicit and Implicit Methods with MILP Solver

If the cutoff time of 1000s is replaced by 72h, the algorithms ***Alg.E.1*** and ***Alg.I.1*** end with a gap of 8.3% and 1.6% respectively; however, no valuable solution was reported by ***Alg.C.6***. Therefore, two advantages can be drawn from decomposition methods in comparison to direct MILP solution: 1) the solution process is observable, i.e., each intermediate solution can be output and its quality can also be identified by gap, and 2) the decomposition strategy makes the large-scale problem tractable in terms of execution time and memory resources.

4.2.3.6 Sensitivity Analysis for K^G and K^L

Table 4.3 summarizes the total cost and execution time in terms of different values of K^G and K^L. As can be seen, the cost goes higher as K^G and K^L increase, since more units should be turned on to compensate for the failure of components. Comparing the most severe contingency with the sufficient one, only 1% additional cost is introduced; however, the solution time increases heavily from 2s to more than 1h, showing that the complexity of the problem is exponentially related with K^G and K^L. It is also noticeable that the system is more capable of surviving the loss of generator than the outage of circuits. The reason is that generator is much easier to be substituted by others if the network is still sufficient, while the loss of circuits usually results in node isolation.

4.2.3.7 Potential Exploration for Large-scale Implementation

The comprehensive case study and discussion on IEEE 24-bus test system reveals that the implicit method ***Alg.I.2*** and parallel explicit method ***Alg.E.4*** are superior to other algorithms and solvers. Their potential on the large-scale instance is validated by the IEEE 118-bus test system in this section. Fig. 4.4 depicts the convergence behavior of ***Alg.I.2***, which terminates at 3h after 3 iterations, while ***Alg.C.6*** runs out of memory at 1.2h. The

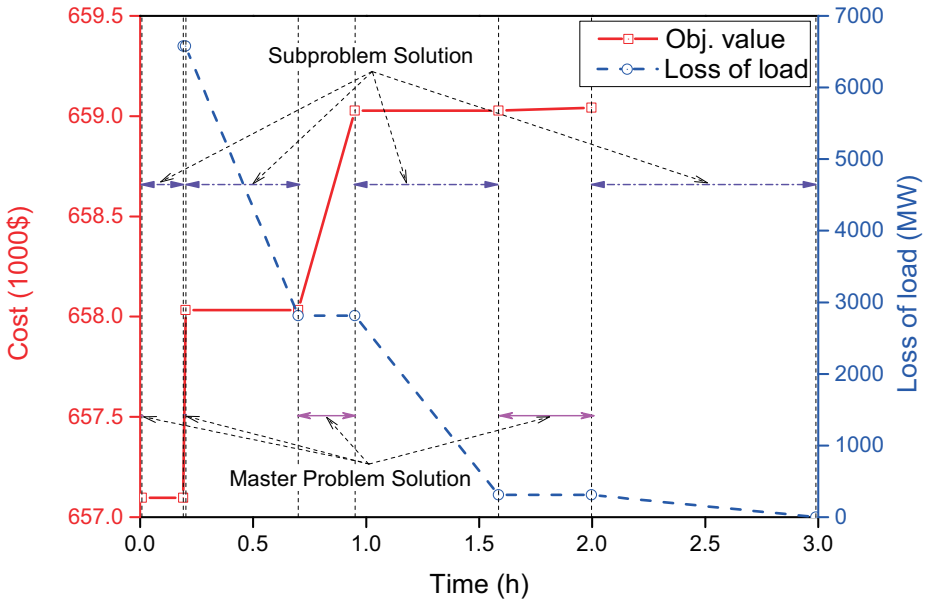


Figure 4.4: Behavior of convergence of ***Alg.I.2*** for IEEE 118-bus test system.

solution process of ***Alg.E.4*** is similar with that of ***Alg.I.2*** illustrated in Fig. 4.4 except for the solution time of subproblems. Finally, ***Alg.E.4*** takes 3.79h in total for solution. Although ***Alg.I.2*** is faster, it may be intractable for larger systems since all the contingency scenarios are included in one MILP subproblem. On the other hand, the solver's limits on the solution of ***Alg.E.4***'s individual LP subproblem are far from being reached.

4.3 Real-Time Optimal Power Flow

As shown in Fig. 1.4, the forecasting error is inevitable during the traditional implementation framework of RTOPF. In order to enhance the accuracy, a three-stage solution framework is developed in this section based on the PDIMP. Mathematical formulation of RTOPF is presented in section 4.3.1. PDIPM is reviewed in section 4.3.2 as a preliminary for the description of parallel computing. Section 4.3.3 illustrates the implementation details of PDIPM with GPU heterogeneous computing. Case studies with both CPU and GPU platforms are reported in section 4.3.4.

4.3.1 Mathematical Formulation

This section briefly introduces the definition and formulation of classical RTOPF, where all the utilized variables and parameters are defined in the nomenclature. To achieve the efficiency and accuracy, a three-stage process is developed for the solution of RTOPF with

REGs, where scenario reduction algorithm and hot-start strategy will be employed.

4.3.1.1 Optimization Model

The RTOPF is expressed as follows: "determining the thermal generator output to minimize the total operation cost based on the forecasted power output of REGs, demand loads, and current status of thermal units".

i. Objective Function

The OPF can be designed to optimize a wide range of goals [19], such as total generation cost, system loss, and emission, etc. The operation cost of the generator is determined for optimization in this work, which is expressed by a quadratic function as follows:

$$\min F = \sum\nolimits_{g\in\mathcal{N}_g} \left[a_g \left(P_g^G\right)^2 + b_g P_g^G + c_g\right]. \tag{4.41}$$

ii. Nodal Power Balance Constraints

The active and reactive power balance on bus $i \in \mathcal{N}_b$ can be written as:

$$P_i^G + P_i^R - P_i^D = V_i \sum_{j\in\mathcal{N}_b} V_j(G_{ij}\cos\theta_{ij} + B_{ij}\sin\theta_{ij}), \tag{4.42}$$

$$Q_i^G + Q_i^R - Q_i^D = V_i \sum_{j\in\mathcal{N}_b} V_j(G_{ij}\sin\theta_{ij} - B_{ij}\cos\theta_{ij}). \tag{4.43}$$

If $i \notin \mathcal{N}_g$ or $i \notin \mathcal{N}_r$, then $P_i^G = Q_i^G = 0$ and $P_i^R = Q_i^R = 0$, respectively. It should be noted that the intersection of $\mathcal{N}_g$ and $\mathcal{N}_r$ may not be empty.

iii. Generator Capacity Constraints

For generator $g \in \mathcal{N}_g$, its active and reactive power output are limited, which can be expressed as:

$$P_g^{G,min} \le P_g^G \le P_g^{G,max}, \tag{4.44}$$

$$Q_g^{G,min} \le Q_g^G \le Q_g^{G,max}. \tag{4.45}$$

iv. Ramp Rate Constraints

The ramp rate limit of generator $g \in \mathcal{N}_g$ is considered as follows:

$$P_g^{G0} - R_g^{G,down} \le P_g^G \le P_g^{G0} + R_g^{G,up}. \tag{4.46}$$

v. Voltage Deviation Constraints

The voltage magnitude at bus $i \in \mathcal{N}_b$ is restricted by its lower and upper limits:

$$V_i^{min} \le V_i \le V_i^{max}. \tag{4.47}$$

vi. Line Security Constraints

The maximum power flow on line $ij \in \mathcal{N}_l$ should not exceed its capacity for security concerns:

$$f_{ij}^{min} \leq f_{ij} \leq f_{ij}^{max}. \tag{4.48}$$

In [148], the constraints (4.48) have been proven to be practically equivalent to (4.49).

$$\theta_{ij}^{min} \leq (\theta_{ij} = \theta_i - \theta_j) \leq \theta_{ij}^{max}. \tag{4.49}$$

4.3.1.2 Uncertainty Management

It should be noted that in the above equations (4.42)–(4.43), the terms P_i^R, P_i^D, Q_i^R, and Q_i^D are fixed forecast values of the next interval. Practically, the output of REG is greatly dependent on the meteorological data, such as the daily solar irradiation and wind speed. Fig. 4.5 gives an example at NREL Solar Radiation Research Laboratory (latitude 39.74°N, longitude 105.18°W, elevation 1829m, and time zone GMT-7) on May 2, 2017 [2]. It can be seen that, for hours resolution, the fluctuation is large and random, i.e., the prediction accuracy is not guaranteed. In the sub-figure, the variation is depicted in every minute, which is more stable and easier to forecast. In this work, although the decision interval is determined as 30s, the prediction can only be performed with significant uncertainty. Therefore, additional enhancement might be required.

Consider the forecasted active power output of REG r is P_r^R, probable scenarios around P_r^R are also likely to be realized, such as $(1 \pm q)P_r^R$, where q can be 5% or 10%. Take Fig. 4.6 as an example, picking one value of these 3 high probable estimations of each REG, a scenario can be generated. It is obvious that the total number of scenarios is 3^{n_r} if there are n_r REGs, which will increase sharply if the dynamics of loads and more values of q are considered. For each scenario, there will be one RTOPF to be built and addressed. More scenarios represent higher accuracy, but the computation may be excessive. Thus, scenario reduction strategies should be employed to strike a balance between the heavy computational burden and prediction accuracy.

Fig. 4.7 demonstrates the stages of the whole solution process of RTOPF with REGs integrated. Stage 1 is devoted to scenario preparation, where the utilized prediction algorithm and scenario reduction strategy are out of the scope of this thesis, for more details please refer to [96, 149, 150]. RTOPF formulation and solution is performed in Stage 2, where the heaviest computation resources are consumed. To achieve high solution efficiency, GPU-based acceleration techniques are introduced in this work. Stage 3 is designed to refine the solution specified from the lookup table and eliminate the forecast error.

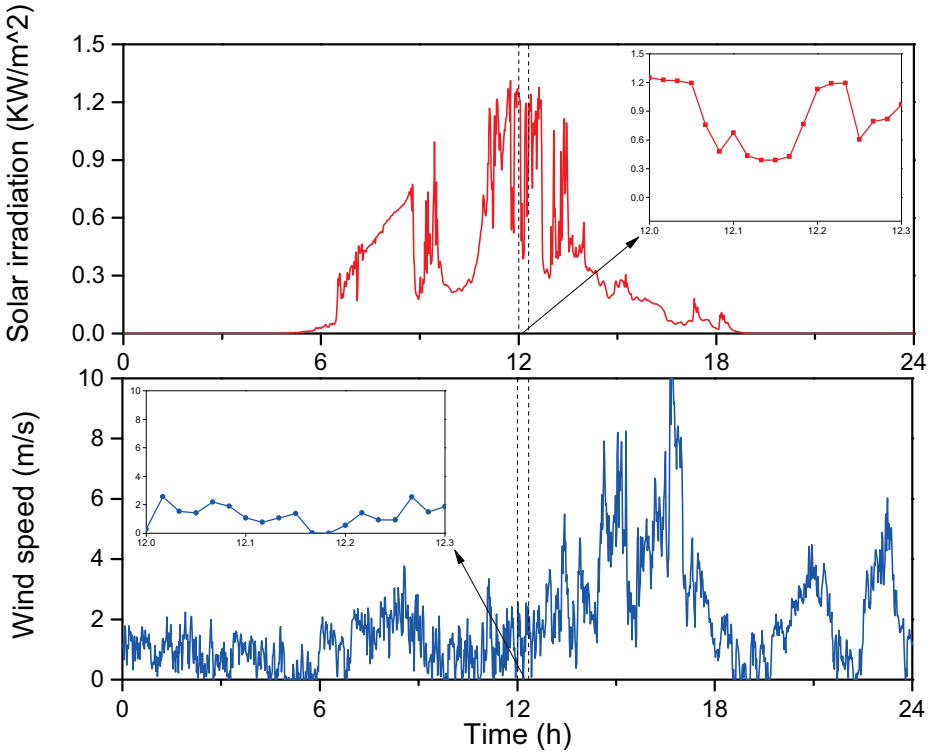


Figure 4.5: Daily solar irradiation (global CMP22) and wind speed (height 19ft) at NREL Solar Radiation Research Laboratory on May 2, 2017 [2].

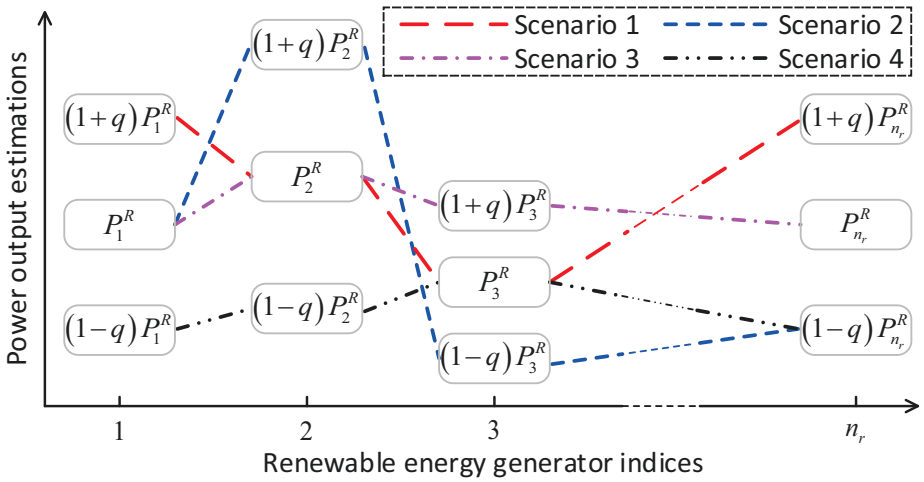


Figure 4.6: Illustration of scenarios for the power output of REGs.

4.3.2 Primal-Dual Interior Point Method

This section reviews the steps of PDIPM [6] for the solution of nonlinear programming problem. The notations are summarized at the beginning, then the derivations are presented.

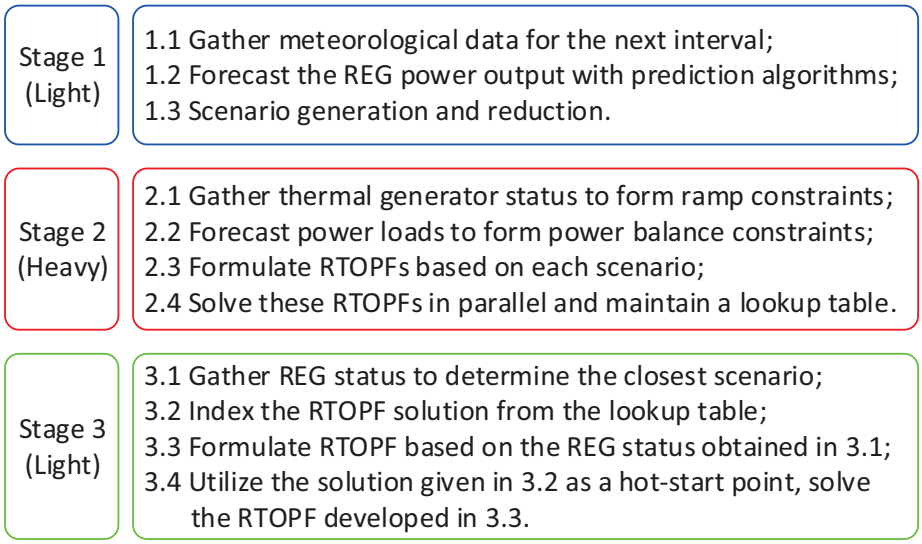


Figure 4.7: Three-stage solution process of the hot-start RTOPF with REGs.

4.3.2.1 Notations

Given a real vector $X = [x_1, x_2, \ldots, x_n]^T$, the first (transpose of the gradient) and second partial (Hessian matrix) derivatives of a scalar function $f(X) : \mathbb{R}^n \to \mathbb{R}$ is given as:

$$f_X = \frac{\partial f}{\partial X} = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}\right], \tag{4.50}$$

$$f_{XX} = \frac{\partial^2 f}{\partial X^2} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}. \tag{4.51}$$

The vector function $F : \mathbb{R}^n \to \mathbb{R}^m$ is stated as $F(X) = [f_1(X), f_2(X), \ldots, f_m(X)]^T$, whose first derivatives (Jacobian matrix) is:

$$F_X = \frac{\partial F}{\partial X} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}. \tag{4.52}$$

A matrix of partial derivatives of $F(X)$ based on the vector λ is:

$$F_{XX}(\lambda) = \frac{\partial({F_X}^T \lambda)}{\partial X}. \tag{4.53}$$

Additionally, $[X]$ represents a diagonal matrix whose diagonal elements are valued by vector X, and e is the vector with all ones.

4.3.2.2 Derivations

For simplicity, the OPF problem (4.41)–(4.48) stated in Section 4.3.1.1 can be rewritten into a compact form as follows:

$$\min \quad f(X), \tag{4.54}$$

$$s.t. \quad G(X) = 0, \tag{4.55}$$

$$H(X) \leq 0. \tag{4.56}$$

where X is the vector $[P_1^G, \ldots, P_n^G, Q_1^G, \ldots, Q_n^G, V_1, \ldots, V_m, \theta_1, \ldots, \theta_m]$; functions $f : \mathbb{R}^n \to \mathbb{R}$, $G : \mathbb{R}^n \to \mathbb{R}^p$, and $H : \mathbb{R}^n \to \mathbb{R}^q$ corresponds to (4.41), (4.42) – (4.43), and (4.44) – (4.47) and (4.49), respectively.

By introducing a barrier function, a perturbation parameter γ, and a positive slack vector Z, the problem is evolved into:

$$\min \quad \left[f(X) - \gamma \sum_{i=1}^{q} \ln(Z_i) \right], \tag{4.57}$$

$$s.t. \quad G(X) = 0, \tag{4.58}$$

$$H(X) + Z = 0, \tag{4.59}$$

$$Z > 0. \tag{4.60}$$

The Lagrangian function of this equality constrained problem (4.57)–(4.60) is:

$$L^{\gamma} = f(X) + \lambda^T G(X) + \mu^T (H(X) + Z) - \gamma \sum_{i=1}^{q} \ln(Z_i). \tag{4.61}$$

The first and second particle derivatives of (4.61) over X is given as:

$$L_X^{\gamma} = f_X + G_X \lambda + H_X \mu, \tag{4.62}$$

$$L_{XX}^{\gamma} = f_{XX} + G_{XX}(\lambda) + H_{XX}(\mu). \tag{4.63}$$

Based on the Karush-Kuhn-Tucker conditions and Newton's method, the following iterative updating procedure can be performed with initiated X, Z, λ, μ, and γ:

- **Step 1:** Compute ΔX and $\Delta \lambda$ according to:

$$\begin{bmatrix} M & {G_X}^T \\ G_X & 0 \end{bmatrix} \begin{bmatrix} \Delta X \\ \Delta \lambda \end{bmatrix} = \begin{bmatrix} -N \\ -G(X) \end{bmatrix}, \tag{4.64}$$

  where

$$M = L_{XX}^{\gamma} + {H_X}^T [Z]^{-1} [\mu] H_X, \tag{4.65}$$

$$N = {L_X^{\gamma}}^T + {H_X}^T [Z]^{-1} (\gamma e + [\mu] H(X)). \tag{4.66}$$

$$\max\left\{\frac{\max\{||G(X)||_\infty, ||H(X)||_\infty\}}{1+\max\{||X||_\infty, ||Z||_\infty\}}, \frac{||L_X^\gamma||_\infty}{1+\max\{||\lambda||_\infty, ||\mu||_\infty\}}, \frac{Z^T\mu}{1+||X||_\infty}, \frac{|f-f_0|}{1+|f_0|}\right\} < \epsilon. \tag{4.76}$$

- **Step 2:** Compute ΔZ and $\Delta\mu$ from:

$$\Delta Z = -H(X) - Z - H_X\Delta X, \tag{4.67}$$
$$\Delta\mu = -\mu + [Z]^{-1}(\gamma e - [\mu]\Delta Z). \tag{4.68}$$

- **Step 3:** Update variables X, Z, λ, and μ according to:

$$X = X + \alpha_p\Delta X, \tag{4.69}$$
$$Z = Z + \alpha_p\Delta Z, \tag{4.70}$$
$$\lambda = \lambda + \alpha_d\Delta\lambda, \tag{4.71}$$
$$\mu = \mu + \alpha_d\Delta\mu. \tag{4.72}$$

  where the primal and dual scale factors α_p and α_d is derived by the following equations with constant parameter $\xi = 0.99995$:

$$\alpha_p = \min\left(\xi \min_{\Delta Z_i<0}\left(-\frac{Z_i}{\Delta Z_i}\right), 1\right), \tag{4.73}$$
$$\alpha_d = \min\left(\xi \min_{\Delta \mu_i<0}\left(-\frac{\mu_i}{\Delta \mu_i}\right), 1\right). \tag{4.74}$$

- **Step 4:** Update perturbation parameter γ by the following mechanism, where σ is set to 0.1:

$$\gamma = \sigma\frac{Z^T\mu}{q}. \tag{4.75}$$

- **Step 5:** Terminate the solution process if convergent criteria (4.76) is met, where f and f_0 are the objective function value of the current and previous iterations respectively; ϵ is valued as 1.0×10^{-6}. Otherwise, go to **Step 1**.

4.3.3 Parallel Implementation on GPU

This section summarizes general features of Nvidia® GPU architecture and heterogeneous computing, with processing flow and instruction rules are described. Based on these regulations, a detailed parallel implementation scheme is elaborated. Finally, the batched linear solver is presented.

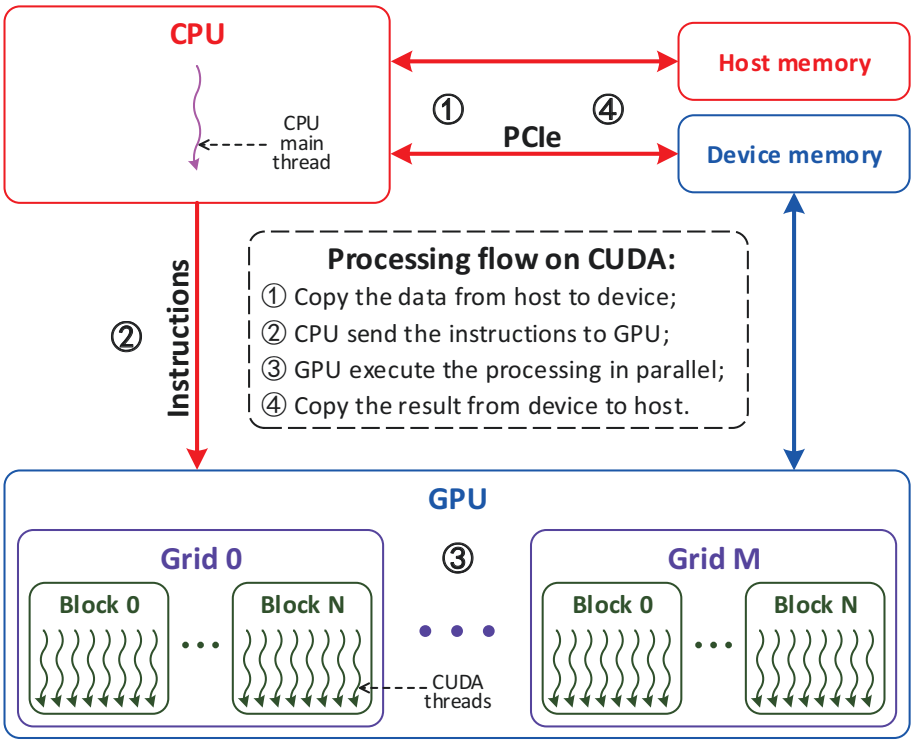


Figure 4.8: CUDA thread hierarchy and processing flow.

4.3.3.1 GPU and Compute Unified Device Architecture

Initially designed for graphic and image processing, GPU has been introduced for the general-purpose scientific computing in recent years [151]. Its popularity on HPC community was soared by the release of Nvidia® CUDA [8]. In CUDA, the function designed for GPU implementation is named as *kernel*. Different from regular C function where one call means one execution, the *kernel* can be executed simultaneously by multiple *CUDA threads* with one call. As shown in Fig. 4.8, the threads are organized in a three-level hierarchy, i.e., each *thread* is contained in a *block* and many *block* consist of a *grid*. During execution, the dimensions of *grid* and *block* are specified, as well as their indices, thus each *thread* can be uniquely identified and controlled. The CUDA programming model is heterogeneous, i.e., all the *kernels* execute on GPU while the rest of the C program run on CPU. In addition, the host and device memories are logically separated. *CUDA threads* can access the device memory during execution, including local memory, shared memory, global memory, etc. *CPU thread* manages the bilateral data transfer between the host and device memory via the API function *cudaMemcpy()*, which is physically performed by PCIe interface. Based on these restrictions, a brief processing flow is given in Fig. 4.8.

In GPU, each *thread* is executed on one *CUDA core*, an arithmetic unit contained in the *Streaming Multiprocessor* (SM). The SM receives computation task in the unit of *block*. Each

thread in one *block* can be concurrently executed in one SM, and one SM can also take care of multiple *blocks* at the same time. Different blocks are independent with each other, while threads within one block can cooperate via shared memory and barrier.

4.3.3.2 Rules for Heterogeneous Computing

Although GPU has far more cores than CPU, its frequency is significantly lower than CPU core and several important features are missing, such as interrupts and virtual memory; therefore, GPU may not always outperform CPU [152]. Actually, the GPU is productive in the manipulation of large amounts of data with many streams, while the CPU excels at doing complex operations on a small set of data.

The SM organizes threads in groups of 32 and called *warps*. If all 32 threads in a *warp* are on the same path, the execution is coherent and efficient [8]. On the other hand, if divergence occurred on the data-dependent conditional branch, each thread will be checked individually and sequentially, i.e., parallelism is limited. Similarly, the *warp* memory access share the same characteristic. If the target address of all 32 threads is successive, the coalesced memory request will be performed; otherwise, up to 32 requests may be launched. The memory access pattern consists one of the most important obstacles for the performance of GPU computing [75]. The GPU-accelerated libraries provided by Nvidia® contribute to a potential. With minimal changes, one can integrate them into the domestic code. Since these libraries are highly optimized, the performance is usually remarkable [153]. It should also be noted that the bandwidth of PCIe is relatively slow and might be the bottleneck for data-intensive problems [8].

Based on the above analysis, three rules for heterogeneous computing is generated as follows:

- **Rule 1:** Allocate computational intensive task to GPU, and leave data-dependent conditional branches for CPU;
- **Rule 2:** Utilize the coalesced memory access pattern on device memory, and introduce the highly tuned and optimized libraries if it is possible;
- **Rule 3:** Minimize the amount of data transferred between CPU and GPU.

4.3.3.3 Implementation Flowchart

According to Fig. 4.7, a set of RTOPFs corresponding to different scenarios should be solved in Stage 2. Since they are mutually independent, parallel implementation is introduced for concurrent solution in this subsection. Fig. 4.9 illustrates the proportion of

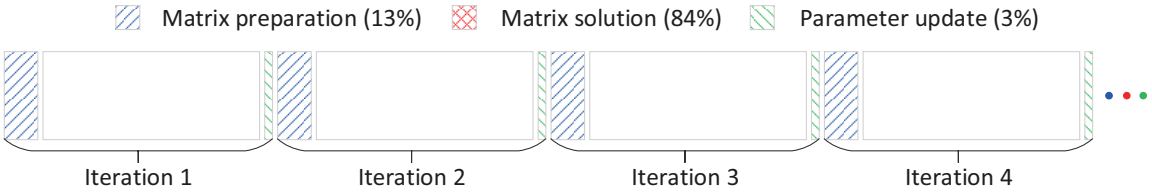


Figure 4.9: The execution time proportion of different operations of PDIPM.

execution time for different operations in a single PDIPM. It is obvious that the linear system (4.64) solution is the most intensive computational task, which consumes more than 80% of the whole execution time. Therefore, based on **Rule 1**, this process should be distributed to GPU. Kernel 2 in Fig. 4.10 achieved this goal. Although the runtime of matrix preparation and parameter updating in Fig. 4.9 are less, and few data-dependent conditional branches are involved, they are executed on GPU with kernels 1 and 3 respectively in Fig. 4.10 to minimize the data exchange between host and device (**Rule 3**), resulting in that the transferred data in each iteration is only the binary indicator for termination.

At the beginning, data preparation will be conducted on CPU. After data copying, kernel 1 will be launched by the instructions from CPU main thread. The execution of kernel 1 is simultaneous with blocks, where each block corresponds to one scenario. Within one block, a maximum number of 1024 threads can be called for parallel processing. In CUDA, there is an implicit barrier between different kernels. Therefore, kernel 2 will not be executed until the matrix preparation processes for all scenarios are finished. This barrier facilitates the utilization of batched solver for the solution of linear systems in kernel 2, which will be exemplified in the next subsection. Similarly, the barrier between kernels 2 and 3 guarantees all intermediate results from linear system solution have been updated for each scenario. The thread organization pattern in kernel 3 is the same with kernel 1. The termination judgment of (4.76) will be performed in kernel 3 and an indicator will be generated. Finally, CPU will copy that indicator from device to host memory when all blocks finished the calculation. The CPU will terminate the whole solution process if the termination criteria has been met; otherwise, execution instructions will be sent to kernel 1 by CPU to start a new iteration. It should be noted that all the intermediate data produced or updated by kernels are stored in device memory, thus the data exchange has been minimized and the barrier is required.

4.3.3.4 Kernels Design

In this work, all vectors are dense, whereas all matrices are sparse and stored with

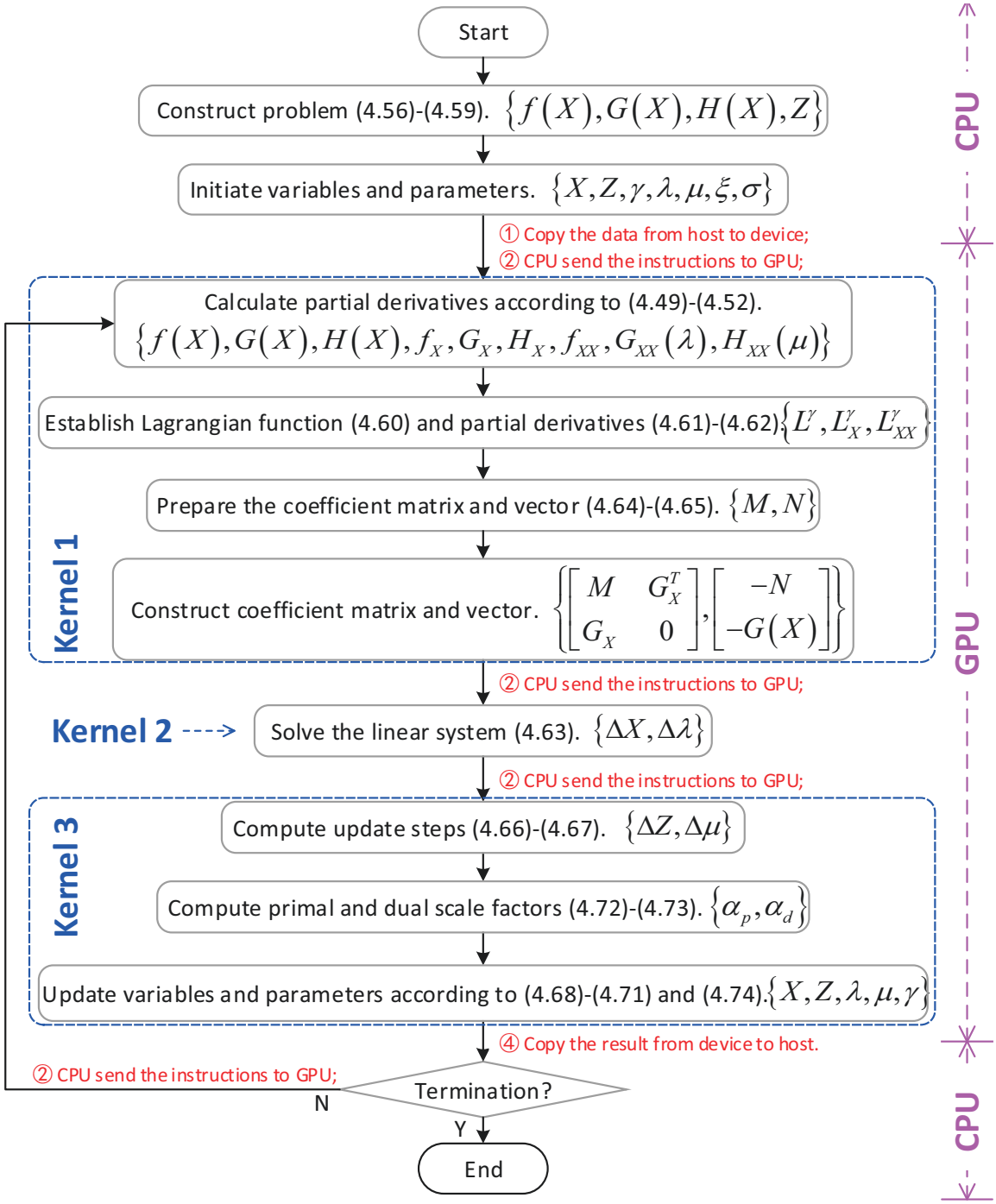


Figure 4.10: Parallel implementation flowchart of concurrent PDIPM with heterogeneous architecture.

Compressed Sparse Row (CSR) format as follows:

$$A = (P_A, I_A, X_A, m_A, n_A, z_A),$$

where m_A, n_A, and z_A are the numbers of rows, columns, and nonzero elements; P_A, I_A,

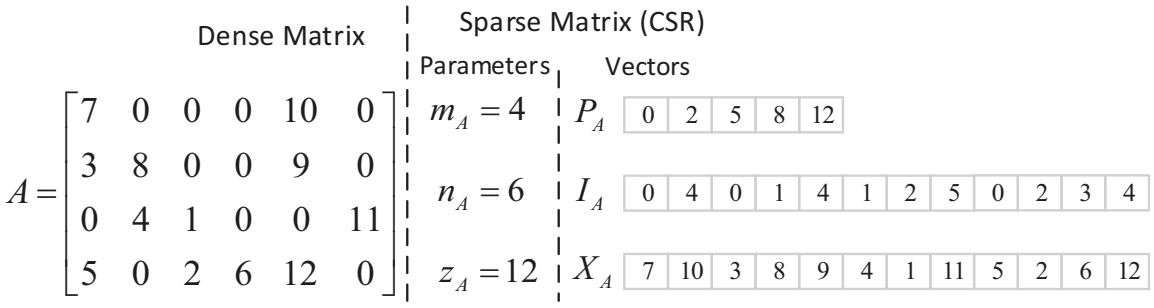


Figure 4.11: Demonstration of CSR format.

Algorithm 4.1 Horizontally concatenating of two matrices $C = [A \mid B]$ in CSR format (P, I, X, m, n, z)

if $m_A \neq m_B$ **then**
  Return "dimension mismatch" message.
else
  Let $m_C = m_A$, $n_C = n_A + n_B$, $z_C = z_A + z_B$;
  Set an indicator $ind = 1$;
  for $j = 1 \cdots m_A$ **do**
    for $p = P_A(j) + 1 \cdots P_A(j+1)$ **do**
      Let $I_C(ind) = I_A(p)$, $X_C(ind) = X_A(p)$;
      Update $ind = ind + 1$;
    end for
    for $p = P_B(j) + 1 \cdots P_B(j+1)$ **do**
      Let $I_C(ind) = I_B(p) + n_A$, $X_C(ind) = X_B(p)$;
      Update $ind = ind + 1$;
    end for
    Let $P_C = P_A + P_B$ (this is a vector addition).
  end for
end if

Algorithm 4.2 Vertically concatenating of two matrices $C = \left[\frac{A}{B}\right]$ in CSR format (P, I, X, m, n, z)

if $n_A \neq n_B$ **then**
  Return "dimension mismatch" message.
else
  Let $n_C = n_A$, $m_C = m_A + m_B$, $z_C = z_A + z_B$;
  Let $I_C = [I_A, I_B]$, $X_C = [X_A, X_B]$;
  Let $P_C = [P_A(1 \cdots m_A), P_A(m_A + 1) + P_B]$.
end if

and X_A are vectors with sizes of $m_A + 1$, z_A, and z_A, respectively. An illustrative example is given in Fig. 4.11

i. Kernels 1 and 3

Data preparation and updating are the main tasks for kernels 1 and 3, where a lot of ba-

sic operations are involved, including matrix/vector element updating, matrix/vector addition, matrix-matrix/matrix-vector multiplication, matrix transposing, minimizing, and summation, etc. In order to achieve the best performance, a lot of investigation has been conducted to achieve the coalesced access pattern based on thread origination and shared memory utilization [154, 155]. In this work, mature strategies reported in the literature are widely consulted and employed.

In addition to the basic operations, horizontally and vertically matrices concatenating are also involved in kernel 1, e.g., $\left[M \mid G_X^T\right]$ and $\left[\frac{M}{G_X}\right]$. **Algorithms 4.1** and **4.2** illustrate the detailed steps of these two operations in the CSR format, whose parallel implementation is also conducted within each block based on the aforementioned techniques. Line 6 in **Algorithm 4.2** consists of two steps: 1) $P_A(m_A+1)+P_B$: add the (m_A+1)th element of vector P_A into each entry of vector P_B; 2) $[P_A(1 \cdots m_A), P_A(m_A+1)+P_B]$: combine the $(1 \cdots m_A)$ elements of vector P_A with the updated vector P_B into one whole vector.

ii. Kernel 2

As shown in Fig. 4.10, a set of linear systems with different coefficient matrices and vectors should be solved by kernel 2. For simplicity, they are denoted as $A_j x_j = b_j$ $(j = 1, 2, ..., N)$ in the following, where N is the number of linear systems. In order to follow the **Rule 2**, *cuSolver* library [156] is introduced for the solution of $A_j x_j = b_j$ on GPU. Table 4.4 summarizes different types of factorization methods provided by *cuSolver* in various modes. In CPU and GPU regular mode, $A_j x_j = b_j$ will be solved one-by-one in *cuSolver* kernels. Although these kernels are highly optimized, the task switching between different linear systems is inevitable, resulting the whole solution efficiency is limited. On the other hand, the GPU batched mode can solve all the N linear systems $A_j x_j = b_j$ at the same time with only one call, thus full potential of GPU resources are utilized and the solution efficiency is highly improved. Based on the above analysis, the QR factorization is determined in this work for the linear equations solution since it is the only method that supported by *cuSolver* in the GPU batched mode. The batched mode requires all the coefficient matrices have the same sparsity pattern, thus adjusting has been made on different matrices A_j in the same iteration as well as different iterations by adding zero values. By doing that, $P_{A_1} = P_{A_2} = ... = P_{A_N}$ and $I_{A_1} = I_{A_2} = ... = I_{A_N}$ can be achieved. The basic steps of batched QR solution is depicted as follows:

- **Prepare A_{batch}:** Summarize all the individual matrices A_j into one batched matrix A_{batch},

$$A_{batch} = \{P_{A_1}, I_{A_1}, X_{batch}, m_{A_1}, n_{A_1}, z_{A_1}\},$$

  where vector V_{batch} stores the nonzero elements of A_j one after another, i.e., $X_{batch} = [X_{A_1}, X_{A_2}, \ldots, X_{A_N}]$.

Table 4.4: Accessible factorization methods of *cuSolver* in various modes.

Methods	CPU	GPU	
		regular mode	batched mode
Cholesky Factorization	√	√	×
LU Factorization	√	×	×
QR Factorization	√	√	√

- **Generate permutation array q:** Generally, there will be extra nonzero entries named fill-ins appeared after the factorization, which demands extra memory space and more arithmetic operations. Fortunately, its number can be greatly reduced by matrix reordering, where a permutation array q is required. In this work, RCM [157] and AMD [79] are considered. The AMD is finally utilized since the it is faster for all tested systems.

- **Reorder A_{batch}:** The fill-ins reduction is performed by reordering A_{batch} into

$$B_{batch} = Q A_{batch} Q^T, \tag{4.77}$$

  where Q is derived from q.

- **Symbolic analysis of B_{batch}:** This process is utilized to determine the sparsity pattern of lower and upper triangle matrices of QR factorization, which will be applied for the parallelism extraction and working space allocation.

- **Numerical factorization for B_{batch}:** This step is performed by all *CUDA cores* in the GPU with intensive parallelism. The generated solution x_{batch} should be reordered according to q before utilization in the following steps.

4.3.4 Case Studies

4.3.4.1 Network and Input Data

The case studies are carried out on four benchmark systems modified from Matpower [143], including IEEE 14-bus, IEEE 57-bus, IEEE 118-bus, and IEEE 300-bus test cases, where 2, 4, 10, and 25 REGs are integrated to substitute thermal generators. For each system, 1024 scenarios are generated at the beginning of each interval based on the meteorological data [158], REG parameter [100], and load profile [99], where time granularity adjusting, data normalization, and scenario reduction processes are employed. All of these 1024 RTOPFs are tackled by a PC equipped with Intel Xeon E5-2620 CPU and Nvidia® GeForce GTX 1080 GPU. The programming environment is Visual Studio 2015. For simplicity, only one interval is conducted for comparison.

Table 4.5: Execution time of RTOPF with different platforms (s).

Cases	14-bus	57-bus	118-bus	300-bus
Regular CPU	20.51	106.95	890.95	2,125.40
Parallel CPU	2.29	11.38	91.91	208.47
Regular GPU	1.51	5.88	43.54	85.05
Batched GPU	0.51	2.20	17.19	37.80

Table 4.6: Speedup of different methods over regular CPU implementation.

Cases	14-bus	57-bus	118-bus	300-bus
Regular CPU	1.00	1.00	1.00	1.00
Parallel CPU	8.95	9.40	9.69	10.20
Regular GPU	13.60	18.19	20.47	24.99
Batched GPU	40.22	48.61	51.83	56.23

In order to validate the performance of the framework developed in section 4.3.3, four types of implementations are compared:

- Regular CPU: All scenarios are solved one-by-one, and the steps shown in Fig. 4.10 is sequentially executed for each scenario. In terms of linear equations solution, the sparse solver *Csparse* developed in [79] is utilized.
- Parallel CPU: All scenarios are solved in parallel based on the OpenMP API with each thread corresponding to one scenario. Within each scenario, the solution process is the same with regular CPU implementation, i.e., sequential and *Csparse*. In the case study, 12 threads are launched.
- Regular GPU: Parallel implementation on GPU according to the flowchart given in Fig. 4.10 except that the kernel 2 is performed by *cuSolver* QR factorization without batched mode enabled.
- Batched GPU: Parallel implementation on GPU with all the proposals shown in section 4.3.3 employed, including the batched QR factorization of *cuSolver*. The batch size is set at 1024 in case studies.

Tables 4.5 and 4.6 summarize the main results for the solution time and speedup, where 1024 scenarios are considered for each test system.

4.3.4.2 Results on CPU Platform

In [23], the execution for a 41-bus system with 49 scenarios is about 80s on CPU. Although the computation platforms are different, the results reported in Table 4.5 with regular CPU

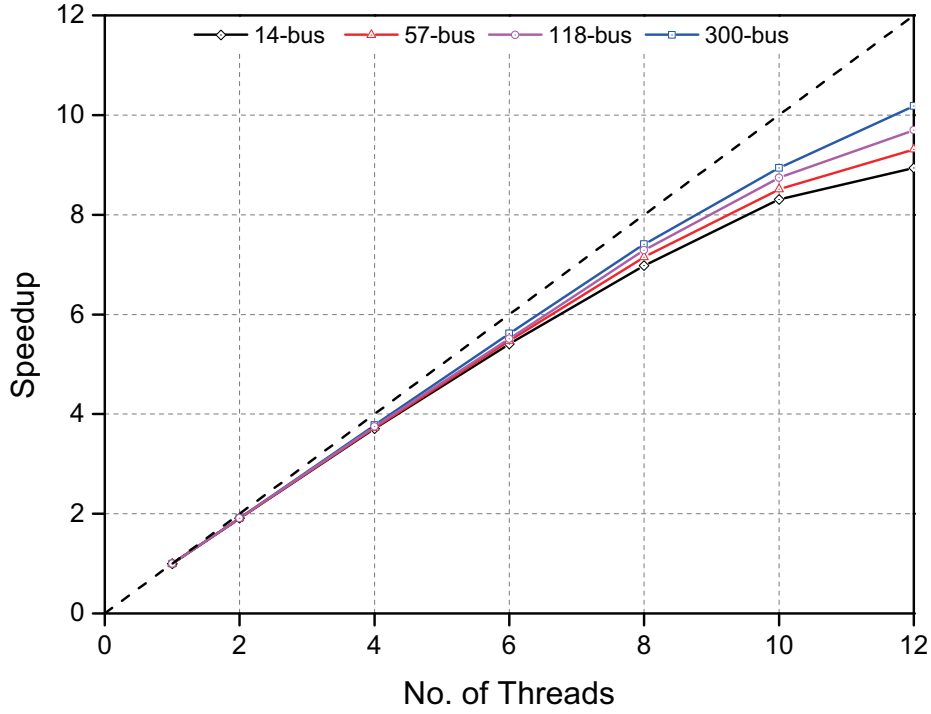


Figure 4.12: Achieved speedups by OpenMP on various test systems with different numbers of threads enabled.

implementation are comparable, e.g., 106.95s is consumed for 57-bus system with 1024 scenarios. Nevertheless, it is far away from real-time application. Therefore, parallel computing with OpenMP is carried out on CPU with 12 threads enabled, resulting in the speedups from 8.95 to 10.20 for various test systems. It can be seen from Table 4.6 that the speedup is higher for larger system, which is partially due to the contradiction between overhead of thread switching and numerical calculation (larger system has heavier computation load). Fig. 4.12 demonstrates the achieved speedups for various systems with different numbers of threads launched. It is observable that the marginal profit gained by thread addition is diminishing, e.g., a speedup of 1.91 can be obtained by 2 threads for 14-bus system, whereas that number is only 8.95 for 12 threads. The reason is still related to the workloads of each scenario.

4.3.4.3 Results on GPU Platform

Although GPU has a smaller frequency than CPU, the number of concurrent threads is much larger, thus the execution time is shorter than parallel CPU as shown in Table 4.5. In order to further improve the solution efficiency, batched mode is introduced, whose solution process is illustrated in Fig. 4.13 as well as the regular QR factorization. Both modes take 5 steps summarized in section 4.3.3.4 to solve a single or one bunch scenarios in iteration 1. Since the sparsity pattern of different coefficient matrices is tuned as the

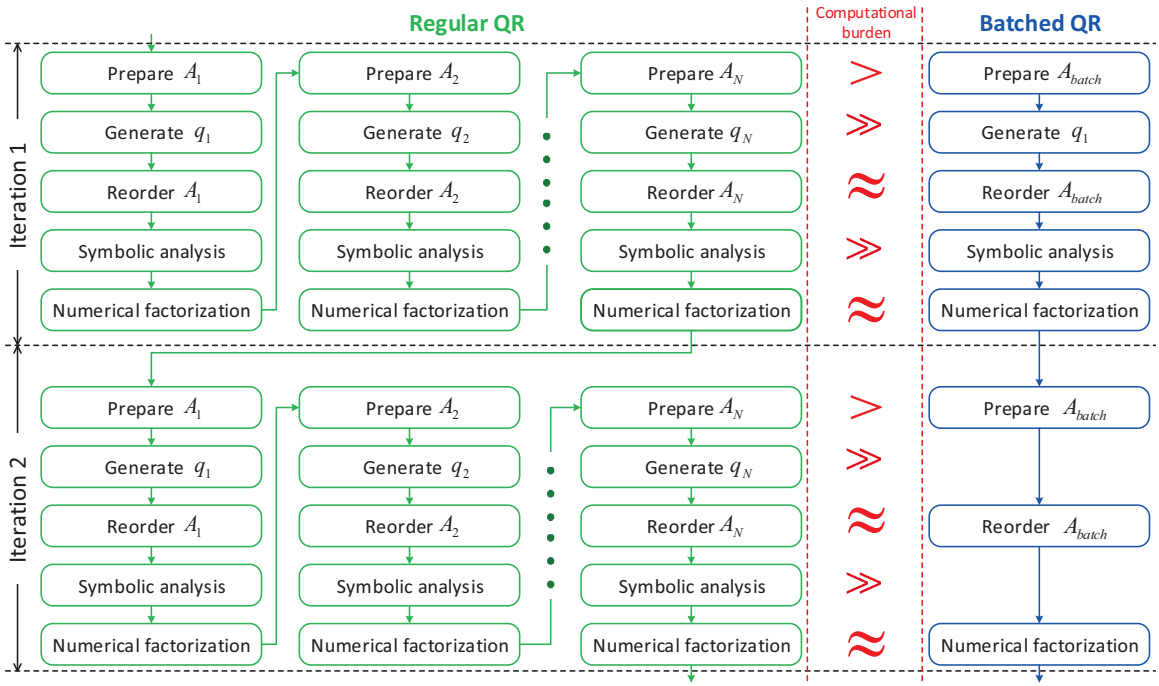


Figure 4.13: Solution process comparison between the regular and batched QR factorization.

same on the batched QR, the results of reorder vector q and symbolic analysis are reusable, thus a lot of effort has been saved as highlighted with $\gg$ in Fig. 4.13. As reported in Table 4.6, the utilization of batched mode has doubled the speedup obtained by regular GPU implementation.

4.3.4.4 Discussions

Apart from the above results on the solution time and speedup, the following discussions are given on two different topics.

i. Batch Sizes

To intensively investigate the performance, different batch sizes are utilized for the solution of case 300-bus. The result is depicted in Fig. 4.14. The total execution time reduced quickly from 85.05s to 42.21s with the batch size increased from 1 to 256, validating the superiority of the batched mode. If the batch size keeps increasing, the runtime still decreases, but the rate is limited, indicating that the full potential of GPU is approaching. Since the fastest improvement occurs at the beginning rather than the later stage, one can use smaller batch size to increase the capability for larger systems with little sacrifice of efficiency. For example, the solution time increased 11.67% by reducing the batch size from 2014 to 256 in Fig. 4.14.

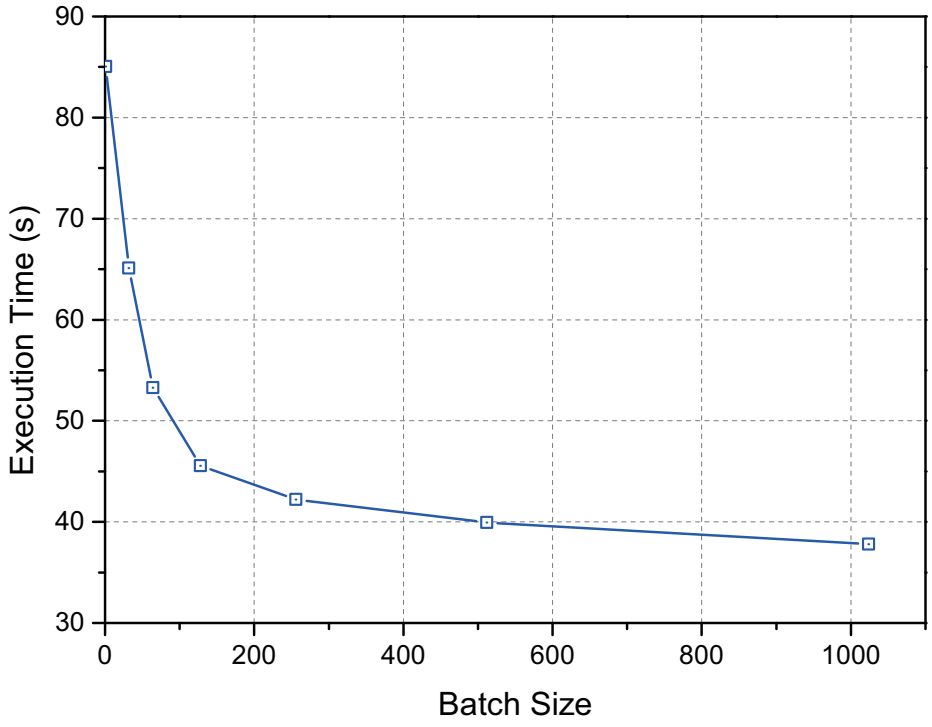


Figure 4.14: Execution time of case 300-bus with different batch sizes (1, 32, 64, 128, 256, 512, 1024).

Table 4.7: Effectiveness of the initiated solution based on different numbers of scenarios for the hot start linear system.

No. of scenarios	32	64	128	256	512	1024
No. of iter. for the final hot-start linear system	8	5	3	2	1	1

ii. Hot Start

Table 4.7 summarizes the numbers of iterations for the final hot start linear system of case 118-bus on the basis of different numbers of scenarios. It is observable that the quality of the initiated solution is higher with more scenarios since the numbers of iterations before convergence is less. On the other hand, fewer iterations means that the initial solution is close to the final one, which can also be explained that the requirement for hot start strategy is more urgent for the circumstance with fewer scenarios. In order to gain better performance, one can increase the number of scenarios; however, there is a saturation point, after which the performance cannot be advanced. In the 118-bus case, that point is 512.

4.4 Summary

Off-line and real-time decision processes of generator optimal operation are investigated in this chapter, where discrete and continuous variables are included respectively.

Both explicit and implicit decomposition frameworks have been investigated for the solution of $n - K^G - K^L$ SCUC problem. Except for the validation of conventional finding that the CCG dominates on Benders cuts, several other conclusions are made: 1) the introduction of Benders cuts may even drag the solution efficiency of CCG; 2) the parallel implementation of explicit method is proportionate with the implicit method; 3) the benefit of multi-cut strategy is not significant; 4) the decomposition framework is superior over commercial solver for this kind of problem; and 5) the system is more capable of surviving the loss of generator than the outage of circuits.

In order to minimize the forecasting error related to REGs and demand loads during the solution of RTOPF, a three-stage framework is employed in this chapter. Scenarios are generated and filtered in Stage 1 to characterize the uncertainty. Stage 2 tries to minimize the prediction interval by GPU acceleration, where heterogeneous computing with batched linear solver is implemented. Hot-start strategy is introduced in Stage 3 to eliminate the prediction error. Comparison between CPU and heterogeneous CPU-GPU platforms are implemented on the IEEE 14-bus, 57-bus, 118-bus, and 300-bus systems, where both regular and batched solution schemes are included. The results validate the superiority of batched GPU over regular GPU, parallel CPU, and sequential CPU.

5 Distribution System Optimal Operation: DNRC and RTVVO

5.1 Introduction

Distribution network is the final stage in the power delivery to bridge individual consumers with the transmission system [19]. Due to the low voltage levels, significant power loss is encountered. Two problems are investigated in this chapter to minimize the active power loss.

At the planning stage, it is designed as interconnected with switches, while during operation it is arranged as radial tree configuration. Therefore, the active power loss can be minimized by changing the open/close status fo switches, i.e., DNRC. Meta-heuristic PSO is utilized to address DNRC due to its MINLP property. In terms of power flow calculation, the DA is employed since it works better than NR and FD in the distribution network. In order to accelerate the solution efficiency, two improvements are proposed on the individual representation and fitness evaluation.

One of the major responsibilities of distribution network is the voltage and reactive power (var) management, i.e., achieving high efficiency, reliability, and quality on the power supply. A lot of control devices are available to fulfill that goal, such as OLTC transformer, voltage regulator, and SC, etc. The RTVVO is designed to minimize system active power losses while satisfying equality constraints to node active and reactive power balances, as well as lower/upper bounds of node voltages. Similar with DNRC, PSO and DA are employed in the solution process. In addition, the GPU is introduced for acceleration in order to achieve the possibility for real-time application. All the solution process is

executed by GPU with the well-established data structure and thread organization pattern, resulting in high efficiency by guaranteeing coalesced access within each warp.

5.2 Distribution Network Reconfiguration

In order to accelerate the DNRC solution process, two major concerns are addressed in this section. In each meta-heuristic algorithm, the individual representation is of great significance for the solution efficiency, which is analyzed and improved in section 5.2.1. Section 5.2.2 is devoted to the enhancement on fitness evaluation process, where the DA is revised based on graph theory. Case studies and discussions are presented in section 5.2.3.

5.2.1 Solution Encoding and Decoding Strategy

The solution of DNRC consists of a series of open branches, which can be intuitively represented with binary and integer numbers. Although both of them are straightforward and easy to implement, the radial topology of the network cannot be maintained efficiently, which results in a large number of infeasible solutions during the evolutionary process; therefore, the convergence property is limited and the capability for the large-scale system solution is weak. To alleviate these concerns, a novel encoding technique was developed in [27], where each branch corresponds to one element in the solution vector and assigned with a real number. When decoding, the real number corresponding to each branch is regarded as the weight, thus a weighted undirected graph was established. By doing an MST computation, the radial topology can be uniquely determined. This method guarantees all solutions are feasible, i.e., the RTS is always preserved.

Although there are several advanced algorithms for MST computing, such as Prim's, Kruskal's, and Boruvka's algorithms, the computational complexity is still high. In addition, the MST calculation is required at each candidate generation. To alleviate the computational burden and improve the solution efficiency, a two-stage probability-based encoding and decoding process is developed in this chapter:

5.2.1.1 Stage 1: Network Analysis

This stage is a preliminary process for solution decoding, which is independent with all encoded solutions and will be executed for only once. The main objective is to find out the shortest cycle for each tie switch and organize them in a specific order. Based on this order, Stage 2 is designed to break these cycles and generate the RTS. For simplicity, a small-scale distribution system is introduced for illustration, which is shown in Fig. 5.1 and consists

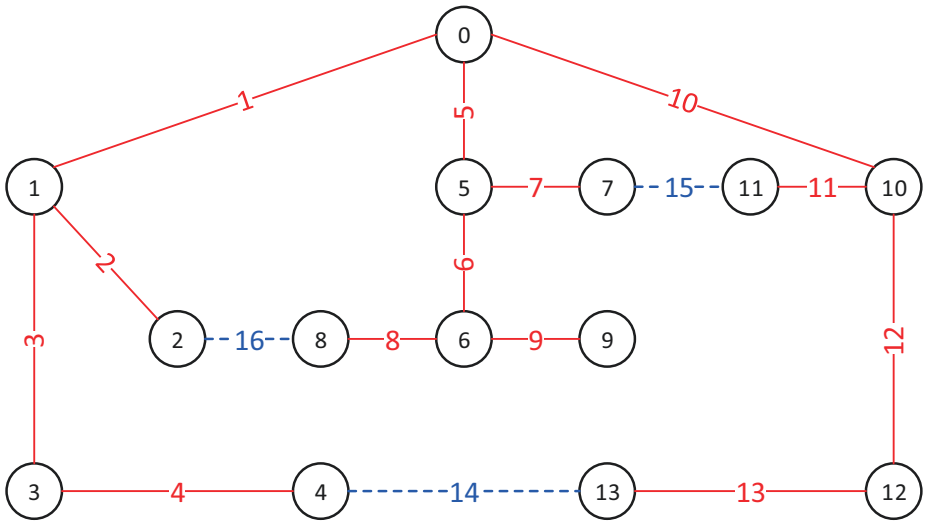


Figure 5.1: Configuration of the target distribution network.

of $n_d = 14$ nodes, $n_b = 13$ branches, and $n_s = 3$ tie switches (indicated by dashed lines). The following steps are responsible for network analysis, where step-by-step temporary results corresponding to Fig. 5.1 are also revealed.

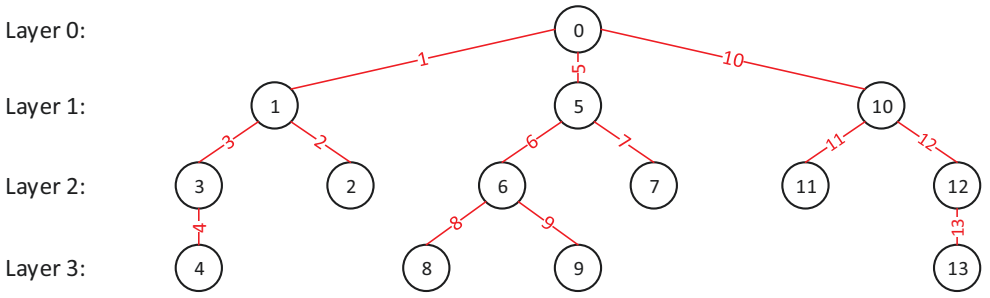
- **Step 1:** Open all the switches to generate a radial tree. The result is indicated by Fig. 5.2(a).
- **Step 2:** Close each tie switch to find a corresponding cycle. This step can be fulfilled by calculating the shortest path between two nodes of each tie switch in the radial tree. Fig. 5.2(b) illustrates these cycles with responding to both branch and node ID.
- **Step 3:** Join the cycles with branch ID into one whole vector $\boldsymbol{C}$, and determine the index of each switch in $\boldsymbol{C}$ to generate another vector $\boldsymbol{D}$. Finally, insert a 0 at the beginning of $\boldsymbol{D}$. Fig. 5.2(c) demonstrates the ultimate results of Stage 1.

Obviously, the number of cycles is n_s; therefore, $\boldsymbol{D}$ is of length $n_s + 1$. On the other hand, the length of $\boldsymbol{C}$ is problem dependent due to branch reputation at different cycles. To sum up, the input of network analysis is the initial configuration, and the outputs are vectors $\boldsymbol{C}$ and $\boldsymbol{D}$.

5.2.1.2 Stage 2: Solution Representation

Based on vectors $\boldsymbol{C}$ and $\boldsymbol{D}$, this stage illustrates how to encode and decode decimal solution vector $\boldsymbol{R}$.

(a)



(b)

Switch ID	Circle (with branch ID)	Circle (with node ID)
14	4 - 3 - 1 - 10 - 12 - 13 - **14**	4 - 3 - 1 - 0 - 10 - 12 - 13 - 4
15	7 - 5 - 10 - 11 - **15**	7 - 5 - 0 - 10 - 11 - 7
16	2 - 1 - 5 - 6 - 8 - **16**	2 - 1 - 0 - 5 - 6 - 8 - 2

(c)

$\boldsymbol{C}$: 4 - 3 - 1 - 10 - 12 - 13 - **14** - 7 - 5 - 10 - 11 - **15** - 2 - 1 - 5 - 6 - 8 - **16**

$\boldsymbol{D}$: **0** **7** **12** **18**

Figure 5.2: Intermediate results of network analysis.

i. Encoding

In order to determine whether a branch should be open or close, a probability value $r_i \in [0, 1]$ is granted for each branch in this work; therefore, the $1 \times (n_b + n_s)$ decimal solution vector $\boldsymbol{R}$ can be encoded intuitively as,

$$\boldsymbol{R} = \{r_1, r_2, \ldots, r_{n_b}, \ldots, r_{n_b+n_s}\}.$$

ii. Decoding

Given a real number encoded solution $\boldsymbol{R}$ as shown in Fig. 5.3 (a), the PLD decoding processes are developed as follows:

- **Step 1:** Generate a $1 \times n_s$ temporary vector $\boldsymbol{T}$, whose elements are corresponding to the probability of each tie switch in $\boldsymbol{R}$.
- **Step 2:** Sort $\boldsymbol{T}$ in ascending order and store the $1 \times n_s$ indexes vector as $\boldsymbol{P}$. Set $k = 0$.
- **Step 3:** Let $k = k + 1$, find out the $\boldsymbol{P}_k$-th cycle based on $\boldsymbol{C}$ and $\boldsymbol{D}$.
- **Step 4:** Look up the probability value in $\boldsymbol{R}$ for each branch of the $\boldsymbol{P}_k$-th cycle.
- **Step 5:** Select the branch with the largest probability value as the one for breaking, which is marked as $\boldsymbol{S}_k$.
- **Step 6:** Update the probability value of the $\boldsymbol{P}_k$-th cycle in $\boldsymbol{R}$ as -1.00.

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
(a)	$\boldsymbol{R}$:	0.44	0.41	0.25	0.71	0.78	0.24	0.81	0.94	0.89	0.07	0.99	0.12	0.74	0.63	0.12	0.73
(b)	Swithch ID:				14	15	16								Sorted $\boldsymbol{T}$:	0.12	0.63	0.73
	$\boldsymbol{T}$:				0.63	0.12	0.73								$\boldsymbol{P}$:	2	1	3
(c)	$k=1$	$\boldsymbol{P}_k=2$	**Circle:** 7 – 5 – 10 – 11 – 15						**Prob:** 0.81, 0.78, 0.07, 0.99, 0.12						**Max:** 0.99	$\boldsymbol{S}_k=11$	
	$\boldsymbol{R}$:	0.44	0.41	0.25	0.71	**-1.00**	0.24	**-1.00**	0.94	0.89	**-1.00**	**-1.00**	0.12	0.74	0.63	**-1.00**	0.73
	$k=2$	$\boldsymbol{P}_k=1$	**Circle:** 4 – 3 – 1 – 10 – 12 – 13 – 14						**Prob:** 0.71, 0.25, 0.44, -1.00, 0.12, 0.74, 0.63						**Max:** 0.74	$\boldsymbol{S}_k=13$	
	$\boldsymbol{R}$:	**-1.00**	0.41	**-1.00**	**-1.00**	**-1.00**	0.24	**-1.00**	0.94	0.89	**-1.00**	**-1.00**	**-1.00**	**-1.00**	**-1.00**	**-1.00**	0.73
	$k=3$	$\boldsymbol{P}_k=3$	**Circle:** 2 – 1 – 5 – 6 – 8 – 16						**Prob:** 0.41, -1.00, -1.00, 0.24, 0.94, 0.73						**Max:** 0.94	$\boldsymbol{S}_k=8$	
	$\boldsymbol{R}$:	**-1.00**	**-1.00**	**-1.00**	**-1.00**	**-1.00**	**-1.00**	**-1.00**	**-1.00**	0.89	**-1.00**	**-1.00**	**-1.00**	**-1.00**	**-1.00**	**-1.00**	**-1.00**

Figure 5.3: Encoded real number solution vector and its decoding.

- **Step 7:** If $k < n_s$, go back to **Step 3**; otherwise, output the decoded solution $\boldsymbol{S}$.

Temporary results from **Step 1** and **2** are illustrated in Fig. 5.3 (b), while other results are demonstrated in Fig. 5.3 (c).

5.2.1.3 Supplementary Explanation

The basic idea of PLD is *branch exchange*: Stage 1 is utilized to find the cycle formulated by closing one switch, and that loop is destroyed in Stage 2 by opening one branch based on the probability. All the cycles generated from Stage 1 are stored in two vectors $\boldsymbol{C}$ and $\boldsymbol{D}$ with respect to the specific order. Intuitively, these circles can be broken with the same order for all decoding process. Nevertheless, this will destroy the randomness and restrict the solution space. For example, if the destruction of cycle $\{4-3-1-10-12-13-14\}$ is always earlier than cycle $\{7-5-10-11-15\}$, then the branch 10 in the second cycle will never be broken since its probability is updated into -1.00 after the destruction of the first cycle. To address this concern, the breaking of different cycles should be conducted in a random order, that is the reason to introduce vector $\boldsymbol{P}$ in **Steps 3 – 6**. In terms of how to determine different $\boldsymbol{P}$ for various decoding process, the easiest method is random generation. However, this will result in diversity during the convergence process since one $\boldsymbol{R}$ may be decoded into different $\boldsymbol{S}$ if different $\boldsymbol{P}$ is utilized. In order to guarantee that one $\boldsymbol{R}$ can only be decoded into a unique $\boldsymbol{S}$, the $\boldsymbol{P}$ should be the same for each decoding. One possible strategy is directly deducing $\boldsymbol{P}$ from $\boldsymbol{R}$, which is fulfilled by **Steps 1 – 2**.

Although Stage 1 is straightforward, it is more computationally intensive than Stage 2 due to the shortest path searching. Similarly, the MST calculation also comes with heavy computation. According to the implementation frameworks given in Fig. 5.4, the heavy computation is involved for all N times of decoding in the MST method, while only 1

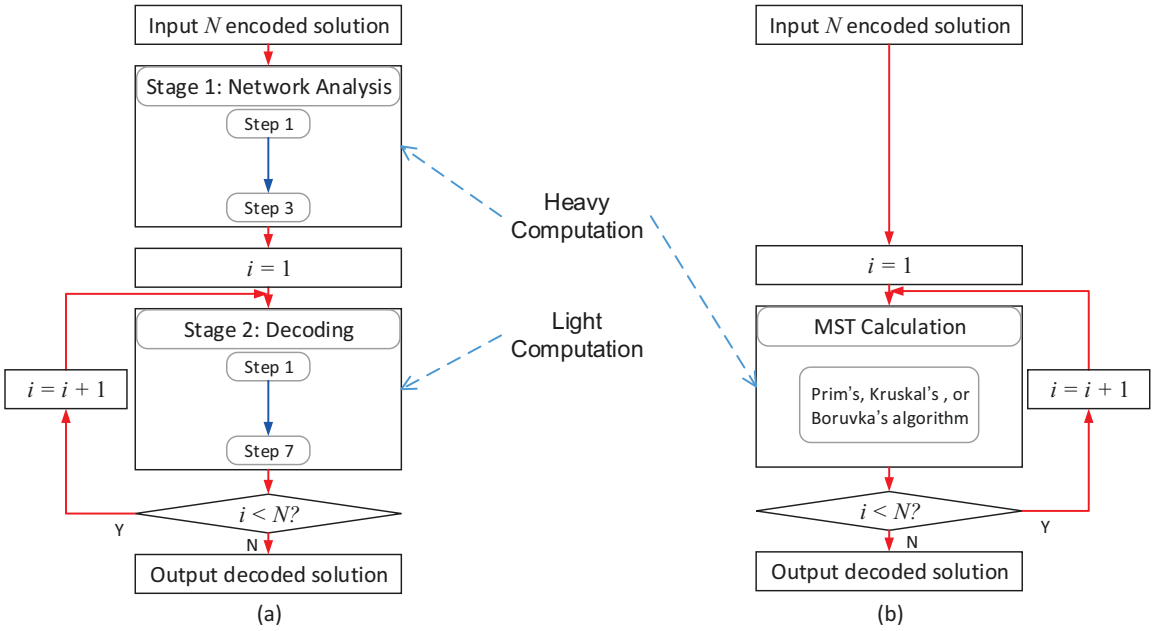


Figure 5.4: Implementation frameworks of different decoding techniques: (a) PLD method; (b) MST method.

network analysis is required for the PLD method; therefore, the computational complexity of PLD is lighter. Solid quantitative validation will be given in the case studies.

5.2.2 Distribution Network Power Flow Analysis

As indicated in the Introduction, the DST proposed by [119] is advantageous, therefore its framework is determined for the solution of DNPF. The DST is dominated by BIBC and BCBV matrices, which are generated by BRD in [119]. After a brief introduction of the solution process of DST, this section intends to propose a novel and efficient MRD for the substitution of BRD when formulating BIBC and BCBV.

5.2.2.1 Solution Process of the DST

Given a distribution network with n_d nodes, the equivalent current injection for node i at the k-th iteration is given as,

$$I_i^k = \left(\frac{P_i + jQ_i}{V_i^k}\right)^* \quad i \in [1, n_d], \tag{5.1}$$

where V_i^k is the voltage of bus i at the k-th iteration; P_i and Q_i are real and reactive power injection on node i, respectively; * is the conjugate operator. Consider $[V^k]$ and $[I^k]$ are vectors of V_i^k and I_i^k without reference node. Then the voltage update steps at the k-th

Algorithm 5.1 Iterative solution process of the DST

1: Initialize $[V^0]$, set iteration $k = 1$ and $[V^1] = [V^0] + 2\epsilon$.
2: Generate matrices $[BIBC]$, $[BCBV]$, and $[DLF]$.
3: **while** $\max\left\{|V^k - V^{k-1}|\right\} > \epsilon$ **do**
4: Calculate $[I^k]$ according to (5.1).
5: Compute $[\Delta V^k]$ based on (5.2).
6: Update $[V^{k+1}]$ just as (5.3), and set $k = k + 1$.
7: **end while**
8: Calculate the power losses based on $[V^k]$ and $[I^{k-1}]$.

iteration is given as,

$$[\Delta V^k] = [BCBV][BIBC][I^k] = [DLF][I^k]. \tag{5.2}$$

Therefore, the voltage vector can be updated as,

$$[V^{k+1}] = [V^0] + [\Delta V^k], \tag{5.3}$$

where $[V^0]$ is a vector whose all elements are the voltage of reference bus.

Based on the above preliminary description and definition, the iterative solution process is summarized in **Algorithm 5.1**.

5.2.2.2 Matrix Generation

It can be seen from **Algorithm 5.1** and (5.1)–(5.3) that matrices $[BIBC]$ and $[BCBV]$ are essential for the iterative procedure. In order to illustrate how to generate $[BIBC]$, the distribution system shown in Fig. 5.1 is utilized as an example, where the tie switches $4-13$, $7-11$, and $2-8$ are open to formulate a radial tree. Consider B_i as the current for branch i. The objective is finding matrices $[BIBC]$ and $[BCBV]$ such that,

$$[B] = [BIBC][I], \tag{5.4}$$

$$[\Delta V] = [BCBV][B]. \tag{5.5}$$

At first, the formulation of $[BIBC]$ is illustrated as follows:

- **Step 1: Reorganize the branch data.** Suppose the input data is the one shown in Fig. 5.5(a), then a radial tree can be generated as in Fig. 5.2(a). This step is exchanging the 'from' and 'to' ends of each branch such that the 'from' has a smaller layer number than the 'to'. The intermediate result is shown in Fig. 5.5(b).

- **Step 2: Rank the branch data.** For any distribution network with n_d nodes, there are $n_d - 1$ branches. After the reorganization, the 'to' nodes of branches are different from each other, which corresponds to $n_d - 1$ non-reference buses. This step ranks

From	To	Z
0	1	$0.075 + j\,0.10$
4	3	$0.04 + j\,0.04$
0	5	$0.11 + j\,0.11$
8	6	$0.11 + j\,0.11$
12	10	$0.08 + j\,0.11$
10	0	$0.11 + j\,0.11$
3	1	$0.09 + j\,0.18$
7	5	$0.11 + j\,0.11$
13	12	$0.04 + j\,0.04$
6	9	$0.08 + j\,0.11$
1	2	$0.08 + j\,0.11$
6	5	$0.08 + j\,0.11$
10	11	$0.09 + j\,0.12$

(a)

From	To
0	1
3	4
0	5
6	8
10	12
0	10
1	3
5	7
12	13
6	9
1	2
5	6
10	11

(b)

ID	From	To	Z
1	0	1	$0.075 + j\,0.10$
2	1	2	$0.08 + j\,0.11$
3	1	3	$0.09 + j\,0.18$
4	3	4	$0.04 + j\,0.04$
5	0	5	$0.11 + j\,0.11$
6	5	6	$0.08 + j\,0.11$
7	5	7	$0.11 + j\,0.11$
8	6	8	$0.11 + j\,0.11$
9	6	9	$0.08 + j\,0.11$
10	0	10	$0.11 + j\,0.11$
11	10	11	$0.09 + j\,0.12$
12	10	12	$0.08 + j\,0.11$
13	12	13	$0.04 + j\,0.04$

(c)

	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}	I_{11}	I_{12}	I_{13}
B_1	1	1	1										
B_2		1											
B_3			1	1									
B_4				1									
B_5					1	1	1						
B_6						1		1	1				
B_7							1						
B_8								1					
B_9									1				
B_{10}										1	1	1	
B_{11}											1		
B_{12}												1	1
B_{13}													1

(d)

	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}	I_{11}	I_{12}	I_{13}
B_1	1	1	1	1									
B_2		1											
B_3			1	1									
B_4				1									
B_5					1	1	1	1	1				
B_6						1		1	1				
B_7							1						
B_8								1					
B_9									1				
B_{10}										1	1	1	1
B_{11}											1		
B_{12}												1	1
B_{13}													1

(e)

Figure 5.5: Intermediate results of matrix BIBC generation.

these branches in an ascending order of their 'to' nodes, and then assigns ID to them. Fig. 5.5(c) illustrates the temporary result.

- **Step 3: Construct the adjacency matrix.** In order to describe the direct relationship between $[B]$ and $[I]$, an adjacency matrix $[A]$ is defined. Its size is $(n_d - 1) \times (n_d - 1)$, containing all branches and non-reference buses. Each element is filled with a binary number, where $A_{ij} = 1$ means that I_j can be directly accessed by B_i, and vice versa. The construction process is as follows: 1) since B_i is identical with I_i, the diagonal of $[A]$ are all ones; 2) if there is a branch from non-reference node i to j, set $A_{ij} = 1$. Fig. 5.5(d) demonstrates $[A]$, where 10 off-diagonal elements are corresponding to 10 branches without reference node.

- **Step 4: Calculate the path matrix.** According to [119], $[BIBC]$ is a binary matrix, and $BIBC_{ij} = 1$ represents that I_j can be accessed by B_i either directly or indirectly. According to the graph theory, this definition is similar to the path matrix, thus the

$[BIBC]$ is generated as

$$[BIBC] = f([A]^{n_d-1}), \tag{5.6}$$

where $f(\ )$ is a function that converts any nonzero elements into 1 and keeps zeros constant. The final obtained result $[BIBC]$ is shown in Fig. 5.5(e).

Based on (5.4), $[BIBC]$ in Fig. 5.5(e) can be interpreted as,

$$\begin{cases} B_1 = I_1 + I_2 + I_3 + I_4, \\ B_5 = I_5 + I_6 + I_7 + I_8 + I_9, \\ \quad \vdots \end{cases} \tag{5.7}$$

which is identical with Fig. 5.1. It was stated in [119] that $[BIBC]$ is an upper triangular matrix. We intend to claim that this is not always true although Fig. 5.5(e) shows an upper triangular pattern. In Fig. 5.5(c), if the 'from' is larger than 'to', matrices $[A]$ and $[BIBC]$ are both not upper triangular. For example, replace the branch $4-3$ with $4-13$, the resulting $[BIBC]$ is not triangular.

As indicated in [119] that the construction processes of $[BIBC]$ and $[BCBV]$ are similar, thus the above process can be reused with minor revision. Actually, these two matrices were built in the same subroutine in [119] to save computation resources and time. In this work, the $[BCBV]$ is generated by the following simple equation,

$$[BCBV] = [BIBC]^T[Z], \tag{5.8}$$

where ^{T} is the transpose operator; $[Z]$ is the matrix whose diagonal is the line impedance shown in Fig. 5.5(c). Combining equations (5.5) and (5.8), we get,

$$\begin{cases} V_0 - V_4 = Z_1B_1 + Z_3B_3 + Z_4B_4, \\ V_0 - V_9 = Z_5B_5 + Z_6B_6 + Z_9B_9, \\ \quad \vdots \end{cases} \tag{5.9}$$

which is identical with Fig. 5.1.

5.2.2.3 Supplementary Explanation

It should be noted that the $[A]$ generated in **Step 3** is not the adjacency matrix according to the definition of graph theory [26] due to the non-zero diagonal. Regard the tree as a directed graph (each branch is directed from the higher layer to the lower layer) and let $[Y] = [A] - [I]$, then $[Y]$ is the adjacency matrix coordinated with the definition. According

to [26], the path matrix can be deduced by,

$$[\tilde{Y}] = [I] + [Y] + [Y]^2 + \cdots + [Y]^{n-1} + [Y]^n, \tag{5.10}$$

where n is the size of $[Y]$, i.e., $n = n_d - 1$ in this work. The (i,j) entry of $[Y]^k$ is equal to the number of walks from node i to j that use exactly k edges. On the other hand, according to the binomial expansion theorem:

$$[A]^n = ([I] + [Y])^n = [I] + C_n^1[Y] + C_n^2[Y]^2 + \cdots + [Y]^n, \tag{5.11}$$

where C_n^r are constant numbers valued as $\frac{n!}{r!(n-r)!}$.

Since there are no cycles in the tree, the maximum number of paths between any two nodes is one; thus, the elements of $[Y]^k$ as well as $[\tilde{Y}]$ are zeros and ones. Therefore, (5.6) can be rewritten as:

$$\begin{aligned}
[BIBC] &= f\left([A]^n\right) = f\left(([I] + [Y])^n\right) = f([I] + C_n^1[Y] + C_n^2[Y]^2 + \cdots + [Y]^n) \\
&= f([I]) + f(C_n^1[Y]) + f(C_n^2[Y]^2) + \cdots + f([Y]^n) \\
&= f([I]) + f([Y]) + f([Y]^2) + \cdots + f([Y]^n) \\
&= [I] + [Y] + [Y]^2 + \cdots + [Y]^n = [\tilde{Y}].
\end{aligned} \tag{5.12}$$

Although equation (5.6) is identical with (5.10), the difference on the computational burden is large. In (5.10), a lot of matrix power should be calculated, such as $[Y]^n$ and $[Y]^{n-1}$, while there is only one matrix power $[A]^n$ executed in (5.6). The complexity of $f(\,)$ is equivalent with the matrix addition. Thus (5.6) is much more efficient than (5.10).

Furthermore, based on the above analysis and (5.11), we obtain:

$$f([A]^{n+k}) = f([A]^n) + [Y]^{n+1} + \cdots + [Y]^{n+k},\ k \geq 0. \tag{5.13}$$

Since the longest walk in the tree with n nodes is $n-1$,

$$[Y]^n = [Y]^{n+1} = \cdots = [Y]^{n+k} = \mathbf{0},\ k \geq 0. \tag{5.14}$$

Thus,

$$f([A]^{n+k}) = f([A]^n),\ k \geq 0. \tag{5.15}$$

This property can be utilized to further improve the efficiency of (5.6), i.e., to reduce the number of matrix multiplications from $n-1$ to $\lceil \log_2 n \rceil$. In this example, $n = n_d - 1 = 13$, therefore $f([A]^{13}) = f([A]^{16})$, and $[A]^{16}$ can be generated by $\lceil \log_2 n \rceil = 4$ times of matrix multiplications as follows:

$$\begin{aligned}
[A]^2 &= [A] \times [A], & [A]^4 &= [A]^2 \times [A]^2, \\
[A]^8 &= [A]^4 \times [A]^4, & [A]^{16} &= [A]^8 \times [A]^8.
\end{aligned}$$

Table 5.1: Scales of the benchmark systems.

Systems	Buses	Feeders	Branches	Tie Switches
14-bus	13	3	16	3
33-bus	32	1	37	5
70-bus	68	2	79	11
83-bus	83	11	96	13
136-bus	135	1	156	21

5.2.3 Numerical Experiments

As demonstrated above, two methods are proposed in this work. In order to evaluate their performance, three types of numerical experiments are implemented in this section. In the beginning, the comparison between the proposed PLD method and its advanced counterpart MST method is carried out for the solution decoding. Then, the MRD method is compared with the BRD method on the DNPF solution. Finally, these methods are integrated into a standard PSO framework for the solution of DNRC. The former two tests are the partial validation of the effectiveness of the proposed methods for the DNRC solution, while the last one is a complete performance evaluation.

Five benchmark systems generated from [167] are introduced as the testbed. Table 5.1 summarizes the scales of these systems. All tests are implemented on a PC equipped with Intel Xeon E5-2620 CPU and Windows 8.1 operating system. Matlab 2017a is employed for programming and execution.

5.2.3.1 PLD Method vs. MST Method for Solution Decoding

Within the evolutionary computing algorithm framework, the solution decoding process is executed in each iteration by a number of times; its efficiency is of great significance for the whole execution. On the other hand, the representation methodology should not contain any bias, i.e., the randomness is valid. This subsection is devoted to the performance evaluation of the PLD method in terms of two different aspects.

i. Randomness

Without supplementary information, the global optimal may lie anywhere in the solution space. Therefore, the evolutionary computing algorithm always demands an evenly initial population to cover the solution space as large as possible. Both the PLD and MST methods are designed for decimal encoded solutions, whose randomness is guaranteed by a lot of software, such as Matlab. Theoretically, the randomness of the decoded integer solution is dominated by the transformation methodology.

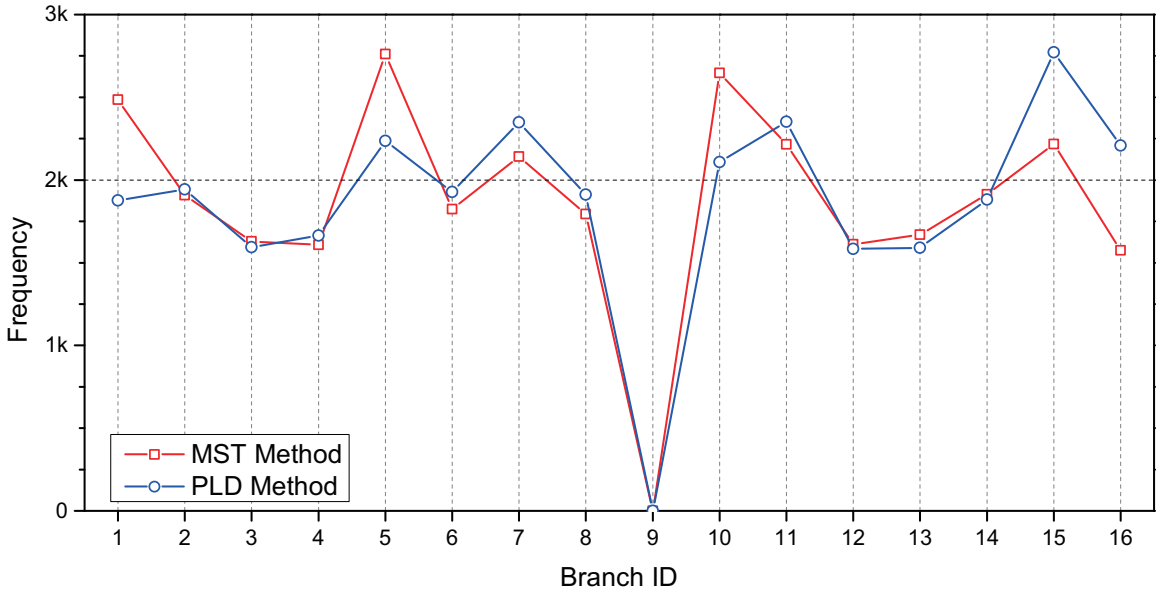


Figure 5.6: Frequency of branches chosen for breaking in the 14-bus system.

For validation and comparison, 10,000 real number encoded solutions are randomly generated for the 14-bus system. After decoding, 10,000 integer solutions with sizes of $1 \times n_s$ are obtained, which means there will be 30,000 branches to be chosen for breaking. Since the number of candidate branches for breaking is only 15 (branch $6-9$ cannot open as it is not included in any cycles), the frequency of each branch should be 2,000. Fig. 5.6 illustrates the results of both methods, it can be seen that the randomness of the PLD and the MST methods are similar and satisfactory. For the MST method, branches close to the root node have a high probability to be chosen for breaking, such as branches 1, 5, and 10. While for the PLD method, branches contained by the shortest cycle are likely to be determined for loop destruction, e.g., branches 7, 11, and 15.

ii. Efficiency

In order to compare the execution efficiency, both methods are implemented to decode N={1, 10, 100, 1000, 10000} solutions for the 14-bus system. Table 5.2 summarizes the comparative results. Fig. 5.7 demonstrates the relationship between the execution time and the population size in the double logarithmic coordinate system. For the MST method, decoding each solution involves one whole MST calculation, thus its execution has a linear relationship with N. On the other hand, the PLD method invokes 2 stages for solution decoding, where Stages 1 and 2 are executed for 1 and N times. In this example, the execution time for a single run of Stage 1 and 2 are 5.95ms and 25.72us respectively. Due to the light computational burden of Stage 2, the total execution time of the PLD method experiences a slow increase. Based on different properties of execution time, the speedup keeps increasing as N increases, but the increase rate is reducing.

Table 5.2: Execution time of the PLD and MST methods to decode N solutions for the 14-bus system.

N	Execution Time (ms)		Speedup
	PLD Method	MST Method [27]	
1	5.981	11.424	**1.91**×
10	6.244	49.031	**7.85**×
100	8.840	426.557	**48.25**×
1,000	34.549	4,176.972	**120.90**×
10,000	289.553	41,735.488	**144.14**×

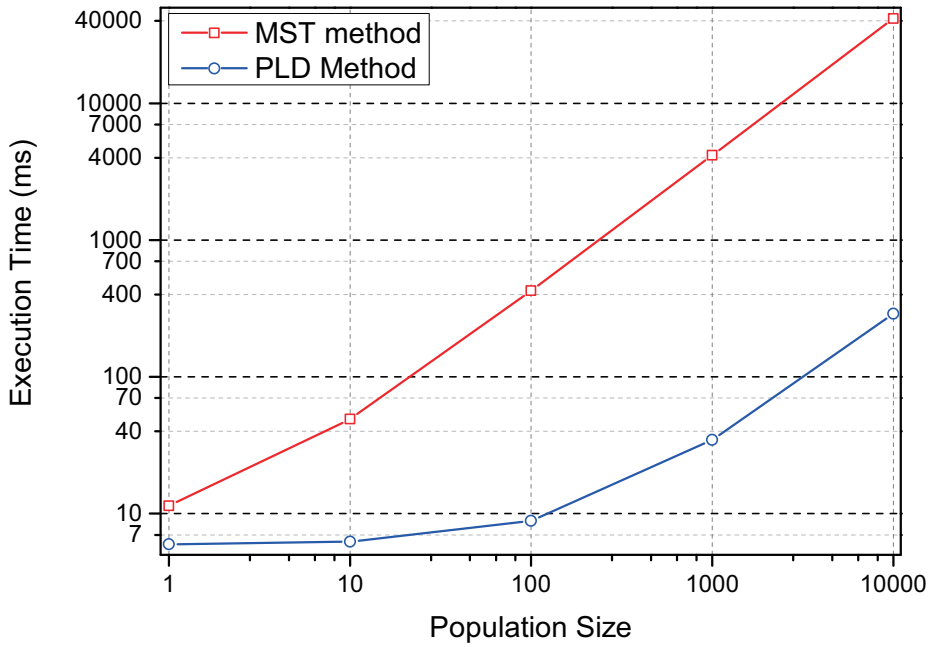


Figure 5.7: Execution time of the PLD and MST methods in double logarithmic coordinate system.

Table 5.3: Execution time of the PLD and MST methods to decode N =10,000 solutions for different systems.

Systems	Execution Time (s)		Speedup
	PLD Method	MST Method [27]	
14-bus	0.290	41.735	**144.14**×
33-bus	0.439	41.905	**95.46**×
70-bus	0.874	42.344	**48.45**×
83-bus	1.006	42.704	**42.45**×
136-bus	1.592	43.012	**27.02**×

Comparisons between the PLD and MST methods on other systems were also implemented. Table 5.3 demonstrates the results, where N is fixed as 10000. The advantage of the PLD over MST method is established by the gained speedup. It should be noted that the execution time of the PLD method goes longer as the system size increases. While the running time of MST method for different systems seems to be constant. The reason is that the utilized method for MST calculation is the Matlab built-in Kruskal's algorithm, which is highly optimized such that the execution time is insensitive to system scales when the sizes are moderate.

5.2.3.2 MRD Method vs. BRD Method for DNPF Solution

In this work, the DST framework given in **Algorithm 5.1** is utilized for DNPF solution, where $[BIBC]$, $[BCBV]$, and $[DLF]$ in lines 1 – 2 can be generated by either MRD or BRD method. In order to compare the efficiency with more details, the solution process of **Algorithm 5.1** is divided into three parts:

- **Part I:** Data preparation and matrix generation of $[BIBC]$, including line 1 and part of line 2;
- **Part II:** Matrix generation of $[BCBV]$ and $[DLF]$, which is described in line 2;
- **Part III:** Iterative DNPF solution, consisting of lines 3 – 8.

Comparison is implemented according to two aspects:

i. Accuracy

In this work, the BRD method reported in [119] is utilized for accuracy validation of the MRD method. Since both methods are integrated into **Part I** of the DST framework, the final difference on the active power losses is fully dependent on the intermediate results provided by the BRD and MRD methods, i.e., the $[BIBC]$ matrix. Since $[BIBC]$ consists of zeros and ones, the difference is easy to identify. Based on the test results for various systems, the $[BIBC]$ generated by the MRD method is exactly the same as the one formulated by the BRD method. Accordingly, the final power losses are the same. Therefore, the accuracy of the MRD method is preserved to be the same with the BRD method.

ii. Efficiency

In this subsection, full DST method is implemented to calculate the DNPF for various systems. Due to alternative methods utilized in **Part I**, the partial as well as total execution times are different, which are all summarized in Table 5.4. As shown in the above, **Parts II** and **III** are the same for these two implementations, thus the execution time is similar and the speedup is close to 1.00. For **Part I**, the MRD method gains a speedup from 1.75 to 4.85

Table 5.4: Execution times of DNPF solution based on the MRD and BRD methods.

Syst.	Execution Time (ms)								Speedup				Exe.
	MRD Method				BRD Method [119]								Time
Parts	I	II	III	Sum	I	II	III	Sum	I	II	III	Sum	Redu.
14-bus	0.23	0.02	0.05	0.29	0.40	0.02	0.05	0.46	**1.75**	0.98	0.98	**1.57**	**36.31%**
33-bus	0.29	0.04	0.08	0.41	0.76	0.04	0.08	0.88	**2.61**	0.98	0.98	**2.13**	**53.15%**
70-bus	0.42	0.15	0.13	0.69	1.53	0.15	0.13	1.81	**3.68**	1.01	1.01	**2.61**	**61.74%**
83-bus	0.44	0.18	0.16	0.78	1.81	0.18	0.16	2.15	**4.12**	0.99	1.00	**2.76**	**63.71%**
136-bus	0.72	0.58	0.33	1.62	3.47	0.57	0.33	4.37	**4.85**	0.99	1.00	**2.70**	**62.93%**

Table 5.5: Configuration of different algorithms.

Algorithms	Configurations				
	PSO	PLD	MST	MRD	BRD
Alg1	•		•		•
Alg2	•		•	•	
Alg3	•	•			•
Alg4	•	•		•	

for different systems, which means that the method is advantageous. Since the improvement is only valid for **Part I**, the value of speedup on the total execution time is reduced to some extent. Take the BRD method as a basis, the last column of Table 5.4 shows the execution time reduction gained by the MRD method. It is observable that the amount of reduction is larger than 50% for the majority of systems, and it goes larger as the system scale increases.

5.2.3.3 Performance Evaluation with Full DNRC Solution

Based on the above results, the superiority of the PLD and MRD over their counterparts is established. This section intends to evaluate their performance on the full solution of DNRC. For consistency, different methods are integrated into the same PSO framework, which is provided by the Matlab optimization toolbox. Therefore, four algorithms are separated, whose configurations are illustrated in Table 5.5. Each system is tested by all four algorithms for 20 trials. It should be noted that all the settings of PSO are kept as default from Matlab except for the population size, which is valued as 256 and 512 for the 14-bus and all the other systems.

i. Quality

It can be seen from Table 5.6 that the average active power loss is the same for different algorithms. The reason is that the global optimum is achieved by these algorithms for all

Table 5.6: Average active power losses by 20 trials (kW).

Systems	**Alg1**	**Alg2**	**Alg3**	**Alg4**
14-bus	**84.880**	**84.880**	**84.880**	**84.880**
33-bus	**139.551**	**139.551**	**139.551**	**139.551**
70-bus	203.384	202.929	**201.412**	**201.412**
83-bus	469.923	470.182	**469.878**	**469.878**
136-bus	285.395	286.412	280.954	**280.877**

Table 5.7: Average execution time by 20 trials (s)

Systems	**Alg1**	**Alg2**	**Alg3**	**Alg4**
14-bus	17.722	16.912	2.230	**1.436**
33-bus	20.122	18.021	4.955	**2.730**
70-bus	99.887	82.303	33.627	**16.009**
83-bus	110.360	89.776	42.422	**17.830**
136-bus	262.537	190.515	122.709	**46.509**

20 trials. As discussed in the above subsection, the MRD shares the same accuracy with the BRD, thus **Alg1** and **Alg2** (**Alg3** and **Alg4**) should terminate with the same quality of results. This prediction has been validated by Table 5.6. On the other hand, **Alg3** (**Alg4**) presents less power loss than **Alg1** (**Alg2**) for all tests, indicating that the PLD method outperforms the MST method in the convergent property.

ii. Efficiency

Table 5.7 summarizes the average execution time of different algorithms for various systems, which is also illustrated in Fig. 5.8. It is obvious that **Alg4** performs better than all the other three algorithms. The difference between **Alg1** (**Alg3**) and **Alg2** (**Alg4**) is marked as DIF_1, which is due to the alternative DNPF solution methods, i.e., MRD and BRD. On the other hand, the difference between **Alg1** (**Alg2**) and **Alg3** (**Alg4**) is marked as DIF_2, which is due to the alternative solution decoding methods, i.e., PLD and MST. It can be seen from Fig. 5.8 that DIF_2 is larger than DIF_1, which means the improvement on the solution decoding is more significant on the DNRC solution. It is also identical with the results reported in Table 5.3 and 5.4 that the speedup gained by PLD is larger than the MRD. However, DIF_1 and DIF_2 are not as large as the difference demonstrated in Table 5.3 and 5.4, the reason is that the PLD and MRD are just parts of the DNRC solution process.

Fig. 5.9 depicts the speedup gained by **Alg4** over the other algorithms. Compared with **Alg1**, both MST and BRD are replaced in **Alg4**, thus the speedup is the largest. According to Table 5.3, the speedup obtained by PLD is decreasing as the system scale increases. While the circumstance is reverse in Table 5.4 for MRD. In addition, the speedup value

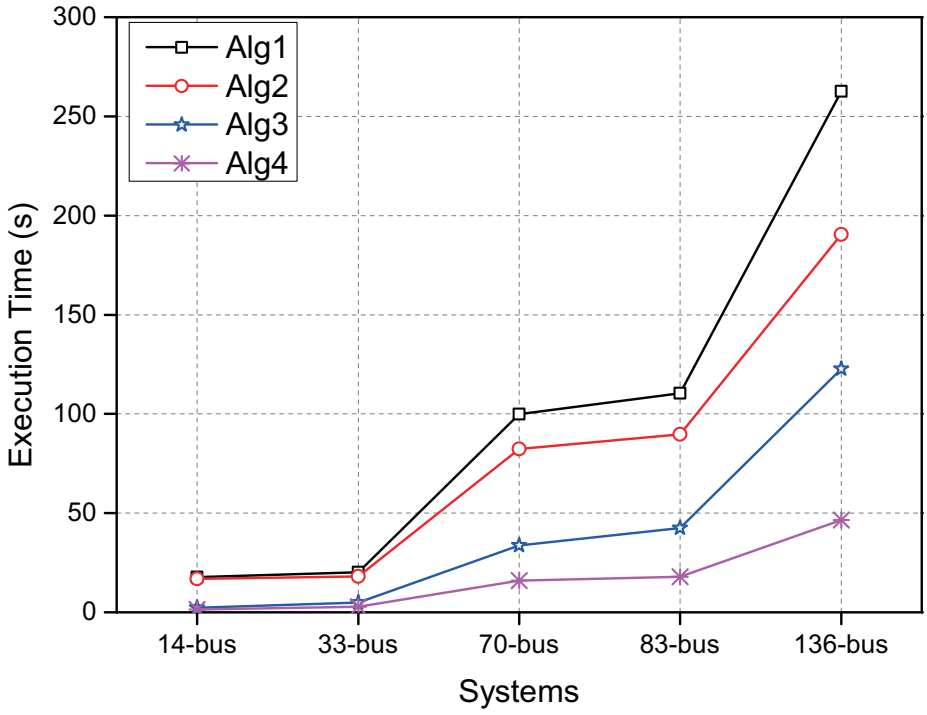


Figure 5.8: Execution time of different algorithms for the DNRC solution.

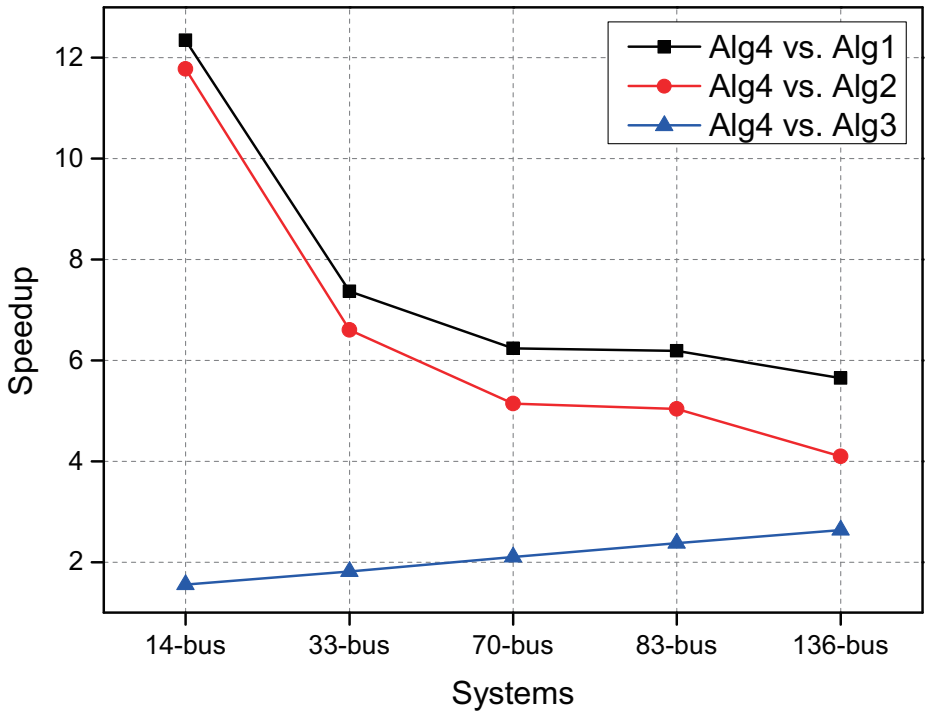


Figure 5.9: Speedup gained by **Alg4** for the DNRC solution.

and decreasing rate in Table 5.3 are larger. Thus, the total speedup of **Alg4 vs. Alg1** is decreasing. Finally, it will end up with a compromise between the increasing and decreasing tendency. Since the MST is updated by PLD, the speedup of **Alg4 vs. Alg2** is also significant. However, without the updating for the DNPF solution, the speedup decrease rate is sharper than that of **Alg4 vs. Alg1**. Both **Alg3** and **Alg4** utilize PLD for solution decoding, the only difference is BRD versus MRD, thus the obtained speedup of **Alg4 vs. Alg3** in Fig. 5.9 is similar with Table 5.4. The DNPF is much more time-consuming than the solution decoding in each iteration. Taking the 136-bus system as an example, they are 1.620*ms* and 0.159*ms*, respectively. Thus, the total speedups demonstrated in Fig. 5.9 are approaching the data reported in Table 5.4.

5.3 Real-Time Volt/Var Optimization

This section illustrates the parallel implementation of RTVVO on GPU with PSO and DA. Section 5.3.1 is devoted to the problem formulation, including the iterative process of DA method, mathematical model of components, and the whole MINLP optimization model of RTVVO. PSO solution framework for this problem is briefly introduced in section 5.3.2. Implementation details related to parallel computing on GPU are revealed in section 5.3.3, such as the design of data structure and organization of threads. Section 5.3.4 validates the solution accuracy and efficiency of the proposed implementation scheme with case studies on four test systems.

5.3.1 Problem Formulation

In this work, three types of control devices are considered for VVO with the objective of minimizing the total active power loss. Based on the integration of AMI, a general implementation scheme of RTVVO for distribution network is given in Fig. 5.10. At the beginning, information related to the current and historical status of SC, OLTC, DG, and load is sampled and collected through smart meters and AMI. Based on these data as well as the parameters of all components (including short/medium-length line and other devices), the RTVVO module performs the fast computing and outputs the optimal coordinated schedule for different types of equipment within a fixed time interval, e.g., 10 seconds. Finally, these instructions are distributed to the corresponding control devices via AMI. Except for the information collection and instruction distribution, the solution of MINLP RTVVO problem turns to be the most important task, which is addressed with RTVVO module in Fig. 5.10. Within RTVVO module, the ACPF calculation will be intensively involved, thus its solution efficiency is of great significance for the whole decision-making process. In this work, the DA proposed in [119] is utilized for distribution network ACPF calculation.

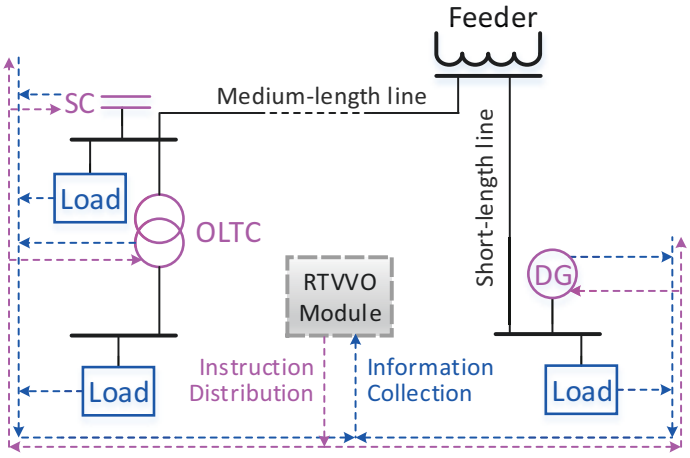


Figure 5.10: Schematic framework of RTVVO for the distributed network.

5.3.1.1 Direct Approach Power Flow Method

Given a distribution network with n_b nodes (where node 1 is regarded as the reference bus), the equivalent current injection for node i at the k-th iteration can be described as:

$$I_i^k = \left(\frac{P_i^d + jQ_i^d}{V_i^k} \right)^*, \quad i \in [2, n_b], \tag{5.16}$$

where * is the conjugate operator. Accordingly, the branch current vector can be obtained:

$$[Bl]_{n_c \times 1} = [BIBC]_{n_c \times (n_b - 1)} [I]_{(n_b - 1) \times 1}. \tag{5.17}$$

Subsequently, the vector for voltage updating can be generated:

$$[\Delta V^k]_{(n_b - 1) \times 1} = [BCBV]_{(n_b - 1) \times n_c} [Bl]_{n_c \times 1}. \tag{5.18}$$

Finally, node voltage vector can be updated:

$$[V^{k+1}]_{(n_b - 1) \times 1} = [V^0]_{(n_b - 1) \times 1} + [\Delta V^k]_{(n_b - 1) \times 1}, \tag{5.19}$$

where $[V^0]$ is a vector with all elements valued as the voltage of reference bus.

Given an initial flat voltage profile, equations (5.16)–(5.19) can be solved sequentially and iteratively until the system reaches a steady state, i.e., the node voltage difference between two successive iterations is less than the specified threshold. Ultimately, the total active power loss can be obtained:

$$P_{loss} = \sum_{l=1}^{n_c} z_l |B_l|^2. \tag{5.20}$$

5.3.1.2 Mathematical Formulation of Components

The above formulation is the basic version of DA, where the transformers and lines are formulated as simple series impedances. Although this is acceptable for short-length lines and untapped transformers, modifications are required to deal with other common devices shown in Fig. 5.10, such as DG, SC, medium-length line, and OLTC transformer.

i. Constant power factor model of DG

DG can be installed by the combination of different energy sources (fuel, wind, and solar) and conversion devices (induction generator, static power converter, and synchronous generator), resulting in various output characteristics [40]. In order to formulate them, three models are developed [168]: constant power factor model, constant voltage model, and variable reactive power model, of which the first one is utilized in this work due to its great popularity.

With specified active power output and power factor, the reactive power output can be calculated [168]:

$$Q_i^g = P_i^g \tan\left(\cos^{-1}\left(f_i^g\right)\right), \tag{5.21}$$

then the equivalent node current injection given in (5.16) should be updated as:

$$I_i^k = \left(\frac{(P_i^d - P_i^g) + j(Q_i^d - Q_i^g)}{V_i^k}\right)^*, \quad i \in [2, n_b]. \tag{5.22}$$

ii. Discrete steps of SC

Given an SC with step s_i^c, its reactive power injection can be given as:

$$Q_i^c = s_i^c \Delta Q_i^c. \tag{5.23}$$

Accordingly, based on (5.22), the node current injection need to be updated:

$$I_i^k = \left(\frac{(P_i^d - P_i^g) + j(Q_i^d - Q_i^g - Q_i^c)}{V_i^k}\right)^*, \quad i \in [2, n_b]. \tag{5.24}$$

iii. Pi-equivalent model of medium-length line

Given a medium-length line between nodes i and j with series impedance z_{ij} and total lumped shunt admittance Y_{ij}, a pi-equivalent model can be formulated as Fig. 5.11. Take the shunt admittance $Y_i = Y_i + 0.5Y_{ij}$ into consideration, the node current injection given in (5.22) should be updated as:

$$I_i^k = \left(\frac{(P_i^d - P_i^g) + j(Q_i^d - Q_i^g - Q_i^c)}{V_i^k}\right)^* + Y_i V_i^k, i \in [2, n_b]. \tag{5.25}$$

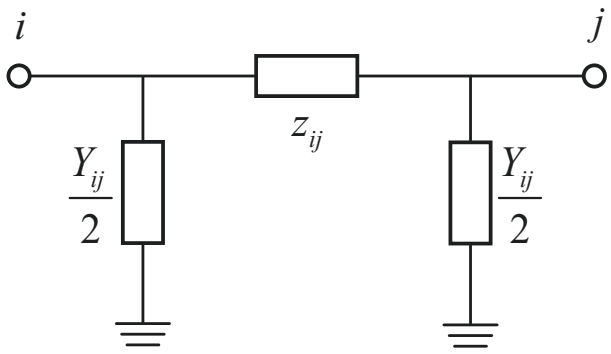


Figure 5.11: Pi-equivalent model of the medium-length line.

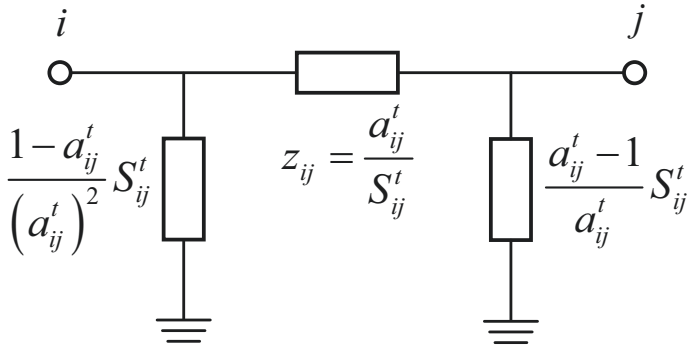


Figure 5.12: Pi-equivalent model of the OLTC transformer.

iv. Pi-equivalent model of OLTC transformer

Given an OLTC transformer between nodes i and j with short circuit admittance S_{ij}^t and regulation tap a_{ij}^t, a pi-equivalent model can be formulated as Fig. 5.12. Similar to Fig. 5.11 and Eq. (5.25), the node current injection needs to be updated with the following shunt admittances:

$$Y_i = Y_i + \frac{1 - a_{ij}^t}{(a_{ij}^t)^2} S_{ij}^t, \qquad Y_j = Y_j + \frac{a_{ij}^t - 1}{a_{ij}^t} S_{ij}^t. \tag{5.26}$$

In addition to the shunt admittances, the variation on the series impedance shown in (5.27) also needs to be integrated into the DA iterative solution process.

$$z_{ij} = \frac{a_{ij}^t}{S_{ij}^t}. \tag{5.27}$$

Initially, z_{ij} is implicitly included in (5.18) according to the following relationship:

$$[BCBV]_{(n_b-1)\times n_c} = [BIBC]^T_{n_c\times(n_b-1)}[z]_{(n_b-1)\times(n_b-1)}, \tag{5.28}$$

where $[z]$ is a diagonal matrix with all elements are corresponding z_{ij}. It should be noted

that, the radial distribution network always has $n_c = n_b - 1$, thus the matrix multiplication in (5.28) is established; while for the meshed networks, as reported in [119] and [169], similar processes are also possible. According to (5.17), (5.18), and (5.28), a rewirtten of (5.18) can be given as:

$$[\Delta V^k] = [BIBC]^T[z][BIBC][I] = [DLF][I]. \tag{5.29}$$

It should be noted that $[BIBC]$ is constant since it is determined by the network topology, whereas $[DLF]$ may change as the variation of parameter a_{ij}^t.

5.3.1.3 Mathematical Formulation of RTVVO

Based on the DA power flow method and component modeling, the mathematical formulation of RTVVO can be given as follows.

i. Objective Function

The objective of RTVVO is to minimize the total active power loss:

$$\min_{[P^g],\,[f^g],\,[s^c],\,[a^t]} \quad \left\{ P_{loss} = \sum_{l=1}^{n_c} z_l |B_l|^2 \right\}. \tag{5.30}$$

ii. Constraints

For this practical problem, the constraints include:

- Distribution network power flow equations:

$$\text{steady state of (5.16) – (5.29).} \tag{5.31}$$

- Active power constraints of DG:

$$P_i^{g,min} \le P_i^g \le P_i^{g,max}, \quad P_i^g \text{ is continuous.} \tag{5.32}$$

- Power factor constraints of DG:

$$f_i^{g,min} \le f_i^g \le f_i^{g,max}, \quad f_i^g \text{ is continuous.} \tag{5.33}$$

- Switch step of SC:

$$s_i^{c,min} \le s_i^c \le s_i^{c,max}, \quad s_i^c \text{ is discrete.} \tag{5.34}$$

- Tap of OLTC transformer:

$$a_{ij}^{t,min} \le a_{ij}^t \le a_{ij}^{t,max}, \quad a_{ij}^t \text{ is discrete.} \tag{5.35}$$

- Bus voltage magnitude limits:

$$V^{min} \leq |V_i| \leq V^{max}. \tag{5.36}$$

- Distribution line thermal limits:

$$\sqrt{P_{ij}^2 + Q_{ij}^2} \leq S_{ij}^{max}. \tag{5.37}$$

- Reactive power overcompensation limits:

$$\sum_{i=1}^{n_b} Q_i^c + \sum_{i=1}^{n_b} Q_i^g \leq \sum_{i=1}^{n_b} Q_i^d. \tag{5.38}$$

It is noticeable that the practical operation limits for SC and OLTC are usually included in DAVVO, e.g., there are maximum allowable daily operating times for each device. Since the RTVVO is designed for a specified time point rather than one whole day, these constraints cannot be directly added. Instead, it can be replaced by another constraint in this work: a component is available for adjusting new commands only when it has been working at a fixed status for a specified number of time intervals. Accordingly, if OLTC mk is not available for adjusting at the current RTVVO, then the values of $a_{mk}^{t,min}$ and $a_{mk}^{t,max}$ in constraint (5.35) should be adjusted into the former status of a_{mk}^{t}, thus OLTC mk will still working at a constant status in the next time interval. That is the reason why historical operation data is required in RTVVO module. Similar operations can also be conducted on (5.34) for SC.

5.3.2 Solution Framework

In this work, the PSO framework is utilized for the solution of RTVVO, where DA is integrated for the fitness evaluation of each particle. A general flowchart is given in Fig. 5.13, where the data exchanging with Fig. 5.10 is highlighted with dashed lines. Since the PSO has been fully established in the literature, only a few basic details are introduced here:

- **Solution encoding:** The decision variables in RTVVO are P_i^g, f_i^g, s_i^c, and a_{ij}^t, where the former two are encoded as continuous numbers and the rest are regulated as integer. A single particle is the vector $x_i = [P_i^g, f_i^g, s_i^c, a_{ij}^t]$ containing all the decision variables.

- **Solution initialization:** All particles are uniformly sampled within the solution space shaped by constraints (5.32)–(5.35). Rounding process will be carried out for the integer decision variables s_i^c and a_{ij}^t.

- **Fitness evaluation:** Penalty factor will be added in objective function (5.30) to punish the violations of constraints (5.36)–(5.38). On the other hand, constraint (5.31) will be

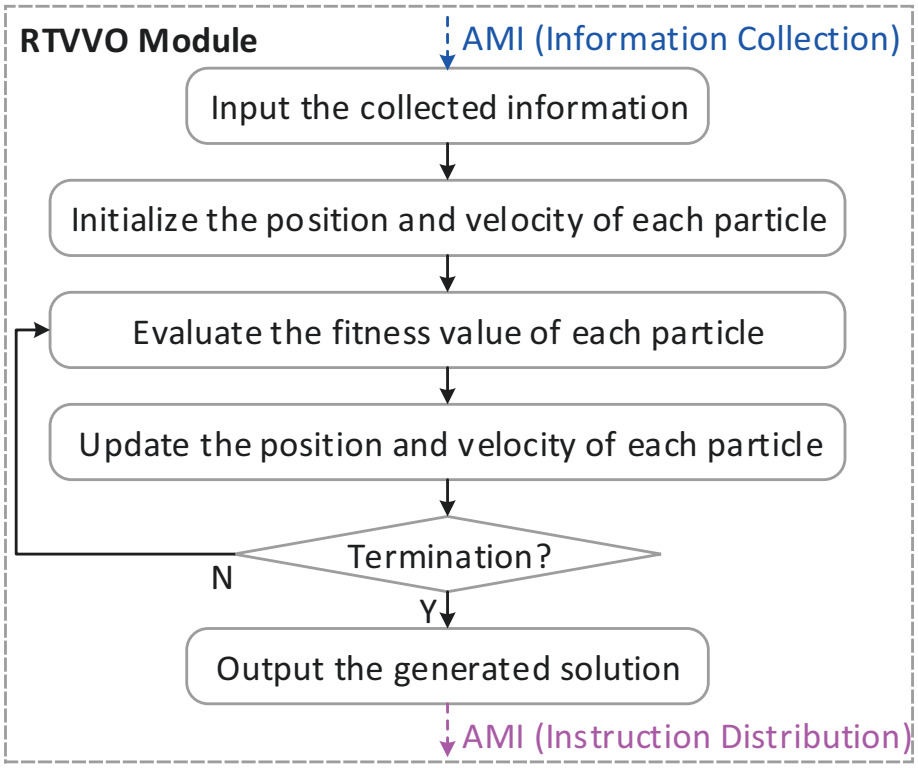


Figure 5.13: General flowchart of the PSO utilized in the RTVVO module.

satisfied by DA power flow, and (5.32)–(5.35) will be addressed in solution updating process.

- **Velocity updating:** Mechanism reported in [48] is utilized for the velocity updating. Equation (5.39) and (5.40) corresponds to continuous and discrete decision variables, respectively.

$$v_i^{k+1} = w_0 v_i^k + c_1 w_1 (p_i^k - x_i^k) + c_2 w_2 (g^k - x_i^k), \tag{5.39}$$

$$v_i^{k+1} = round\left(w_0 W_0 v_i^k + c_1 W_1 (p_i^k - x_i^k) + c_2 W_2 (g^k - x_i^k)\right), \tag{5.40}$$

where $w_0 = 1 - 0.6k/N$, $c_1 = 2$, $c_2 = 2$ are fixed parameters; k is the current iteration order; N is the maximum number of iterations; w_1 and w_2 are uniformly random numbers within $[0, 1]$; W_0 is random number taking discrete values of 0, -1, or 1; W_1 and W_2 are random discrete numbers of either 0 or 1.

- **Solution updating:** The solution updating is performed with (5.41). It can be seen from the initialization process and (5.40) that the discrete decision variables are always kept in integer form. If the updated decision variable is out of the range regulated by (5.32)–(5.35), it will be forced to the nearest boundary.

$$x_i^{k+1} = x_i^k + v_i^{k+1}. \tag{5.41}$$

Algorithm 5.2 Particle swarm optimization framework for RTVVO

Prepare matrix $[BIBC]$ based on graph searching.
Input state variables P_i^d, Q_i^d, z_{ij}, Y_{ij}, ΔQ_i^c, and S_{ij}^t.
Initialize the population.
for each iteration **do**
  for each particle **do**
    Read decision variables P_i^g, f_i^g, s_i^c, and a_{ij}^t.
    Update network configurations Q_i^g, Q_i^c, Y_i, and z_{ij} according to (5.21), (5.23), (5.26), and (5.27).
    Update matrix $[DLF] = [BIBC]^T[z][BIBC]$.
    Initialize a flat node voltage vector $[V^0]$.
    while $\max\left\{|V^k - V^{k-1}|\right\} > \epsilon$ **do**
      Calculate $[I^k]$ according to (5.25).
      Compute $[\Delta V^k]$ based on (5.29).
      Update $[V^{k+1}]$ based on (5.19), and set $k = k + 1$.
    end while
    Calculate the active power loss based on (5.17) and (5.20).
    Check the constraints (5.36)–(5.38) and update the fitness value.
  end for
  Determine the local and global particles, update the velocity and position $[P_i^g, f_i^g, s_i^c, a_{ij}^t]$ based on (5.39)–(5.41).
end for
Output the global best particle.

- **Termination criteria:** In order to guarantee fair comparison in case studies, i.e., the computation load for different runs are the same and insensitive to random numbers, the PSO will terminate at a fixed number of iteration in this work. It is also easy to be extended to other criteria, such as terminating if the global best has not been updated for a specified number of iterations.

5.3.3 Parallel Implementation

For the purpose of facilitating description, the solution process given in section 5.3.2 is summarized as **Algorithm 5.2** with the detailed data flow from DA and PSO. Since all particles are mutually independent, each particle at a specified iteration can be manipulated with one thread or block on GPU, therefore, the parallel implementation seems to be trivial. However, in order to gain superior performance, the design of data structure and thread organization require further investigation.

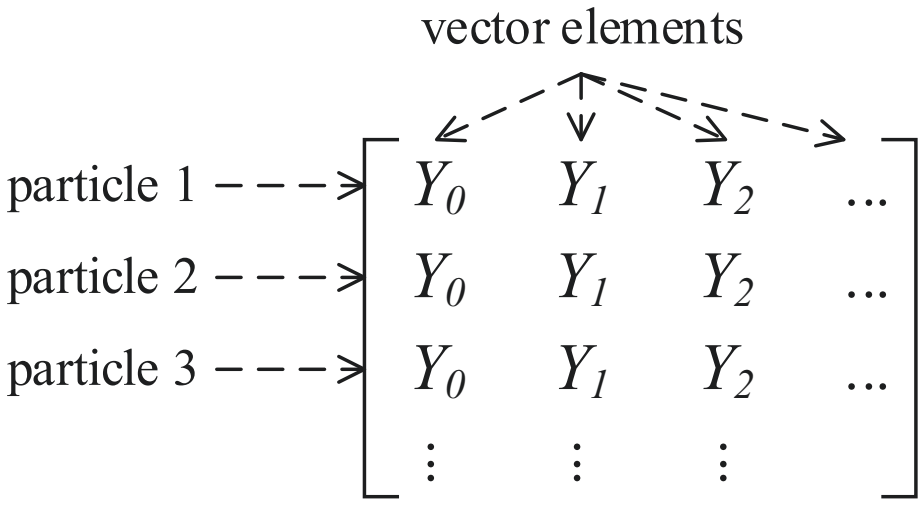


Figure 5.14: Unified vector storage structure across different particles.

Algorithm 5.3 DLF translating from $\{p, i, xp, xx\}$ to $\{p, i, x\}$

```
Initialize all x(i) into 0 (suppose x has n_x elements).
for i = 1 ··· n_x do
    for j = xp(i) ··· (xp(i + 1) − 1) do
        x(i) = x(i) + z(xx(j)).
    end for
end for
```

5.3.3.1 Data Structure

As can be seen from **Algorithm 5.2**, each particle maintains a copy of all intermediate vectors and matrix $[DLF]$, whereas the matrix $[BIBC]$ is constant and the same for all particles. For different vector copies across particles, they are stored as a unified matrix with each row corresponds to one particle. An illustrative example for vector $[Y]$ is given in Fig. 5.14. In this work, both $[DLF]$ and $[BIBC]$ are stored with the compressed sparse row (CSR) format [79] since it enables fast row access and matrix-vector multiplications. As the CSR structure $\{p, i, x\}$ can be decomposed into three arrays p, i, and x, the unified vector storage pattern shown in Fig. 5.14 is also valid for matrix storage. Since the $[DLF]$ across different particles have the same pattern, vectors p and i are the same for each particle, thus only vector x is required in different particles for distinction, i.e., the storage space for p and i is saved.

Although the sparse technique has been utilized, the calculation of $[DLF]$ by two times of matrix multiplication $[DLF] = [BIBC]^T[z][BIBC]$ is still computationally intensive. In Fig. 5.15, at least 27 atom operations are involved. As the matrix $[BIBC]$ is fixed and all its elements are either 0 or 1, we intend to improve the computation efficiency by translating the matrix multiplication into the point-wise substitution. Fig. 5.15 gives an illustrative

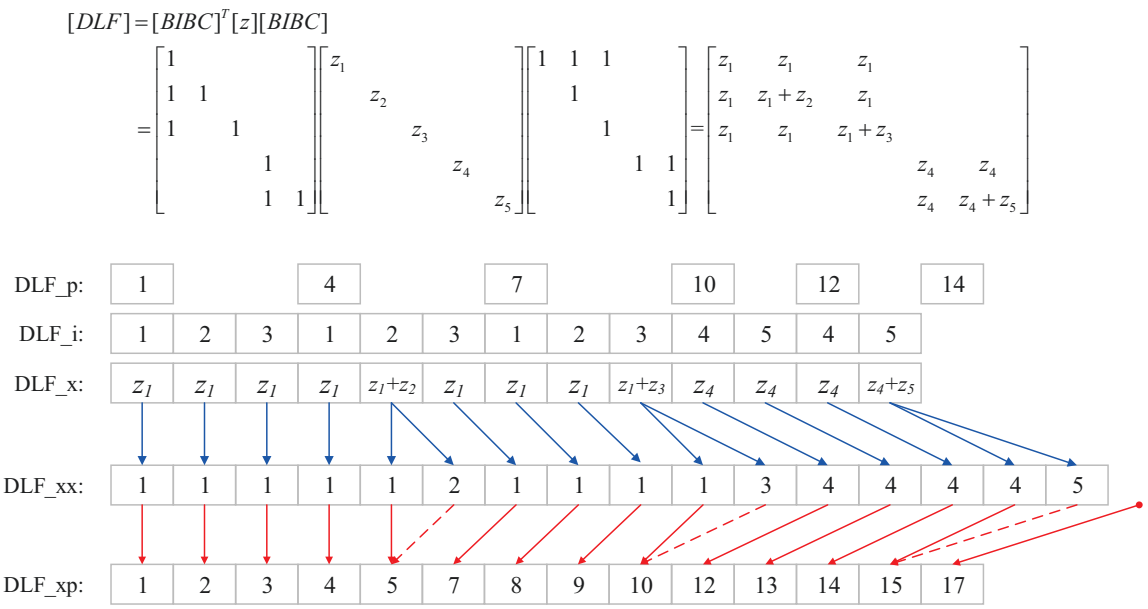


Figure 5.15: Sparse storage patterns of DLF matrix in CSR format with $\{p, i, x\}$ and $\{p, i, xp, xx\}$.

example. First, we change the CSR format of $[DLF]$ from $\{p, i, x\}$ into $\{p, i, xp, xx\}$. The array xx is derived from x by decomposing all atom operations into successive entries, i.e., expanding one element $z_1 + z_2$ into two elements 1 and 2. For each original element of x, its new order in xx is recorded as the array xp. Just like the array p, one more element is supplemented at the end. Given an array $[z]$, it is easy to obtain $\{p, i, x\}$ from $\{p, i, xp, xx\}$ by Algorithm 5.3 with only 16 atom operations. The performance difference between matrix multiplication and point-wise substitution is much larger for bigger systems, which will be demonstrated in case studies.

5.3.3.2 Thread Organization

CUDA provides a three-level parallelism based on thread hierarchy [8]: different streaming multiprocessor (SM) in GPU is possible for concurrent execution; each SM can launch multiple blocks at the same time; the threads within one block will be executed simultaneously. Different blocks are independent with each other, while threads within one block can cooperate via shared memory or barrier. The barrier synchronizes all threads in one block by forcing them to wait at a specified point until everyone has reached. The shared memory is possessed by each block visible to all threads within that block, but the size is limited. Since the access latency of shared memory is low, it should be fully utilized when it is possible. In addition, each thread can access the much bigger global memory of GPU with a higher latency.

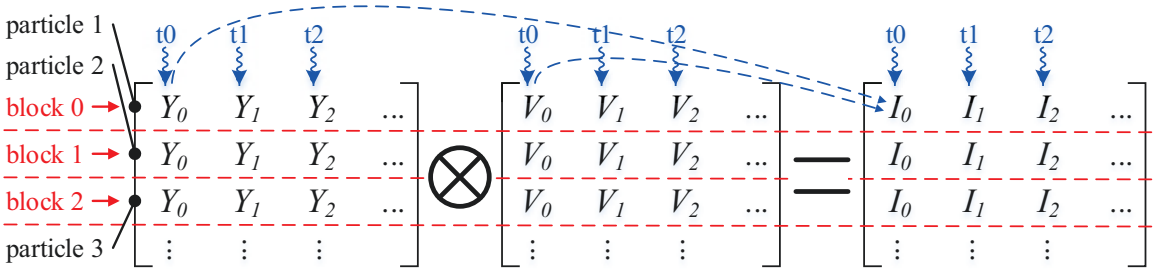


Figure 5.16: Thread organization for parallel regular mapping.

Although SM receives computation task in the unit of block, it creates, manages, schedules, and executes threads in groups of 32 and named warps [8]. A warp executes one common instruction at a time. If the target address for all 32 threads is successive, the access can be fulfilled with full efficiency (called coalesced access); otherwise, the divergence (scattered access) will occur and result into the low bandwidth. Since the efficiency of warps dominates the whole solution performance, how to guarantee successive data access is of high priority when organizing threads [75].

In the following subsections, key steps of **Algorithm 5.2** are tuned for coalesced access with thread organization.

i. Parallel Regular Mapping

There are a lot of element-wise calculations and updating operations in Algorithm 5.2, such as lines 7, 9, 11, 13, and 16, which can be regarded as regular mapping. Take the line 11 as an example, the last term of (5.25) is $I_i = Y_i V_i$, Fig. 5.16 illustrates the thread organization pattern, where $\otimes$ represents the element-wise mapping process. In alliance with Fig. 5.14, vectors $[Y]$ and $[V]$ across particles are stored in one matrix. It can be seen that there is no leap during the access, thus we naturally distribute each particle (one row) into one block. In Fig. 5.16, all particles are concurrently executed since multiple blocks can be launched at the same time. Within all blocks, coalesced access is fulfilled by each warp, e.g., the successive address from Y_0 to Y_{31} are accessed by threads 0 to 31.

ii. Parallel Reduction

Different with regular mapping process, where the length of the input vector is the same as the output vector, the reduction process takes a vector as input but output only one value. This type of operation is also involved in Algorithm 5.2, such as the maximizing in line 10, the minimizing in line 18, and the summation in line 15. It should be noted that the reduction operation in line 15 is due to (5.20), where the term $z_l|B_l|^2$ is calculated with parallel regular mapping and stored as an intermediate vector. Similar to Fig. 5.16, each particle (vector) is handled with one block. The thread organization for parallel reduction

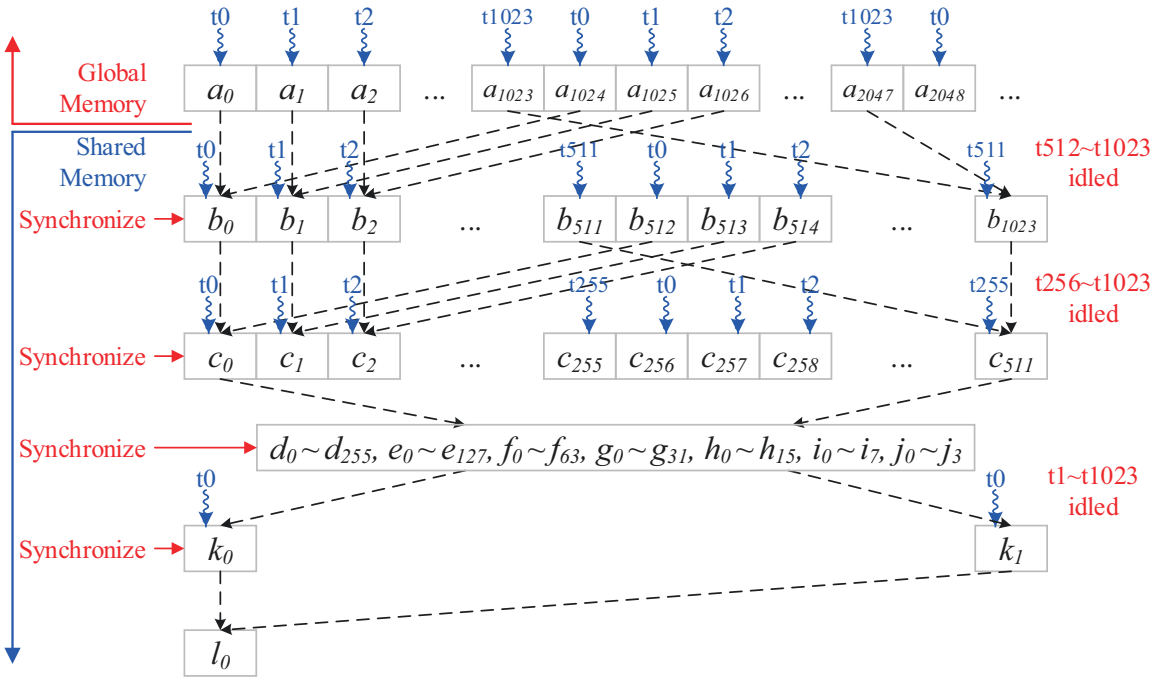


Figure 5.17: Thread organization for parallel reduction within each block.

within each block is illustrated in Fig. 5.17. The first step is copying the original vector $[a]$ stored in global memory into $[b]$ in shared memory. During this step, reduction process may be required, e.g., reducing a_0 and a_{1024} to b_0. There should be a barrier at the end of this step to guarantee all shared memory has finished data updating, otherwise, dirty data will appear and result into a wrong output. The following steps are familiar with the first one, except that both input and output data are stored in shared memory and the number of active thread is reduced into half of the former. It can be seen that the coalesced access is guaranteed at each step, i.e., there is no leap within each warp.

iii. Parallel Matrix-Vector Multiplication

Although the matrix-matrix multiplication in $[DLF] = [BIBC]^T[z][BIBC]$ has been eliminated by data structure design in the above, the matrix-vector multiplication is inevitable in line 12 of Algorithm 5.2. In alliance with the previous process, each block is assigned to one particle, i.e., one matrix-vector multiplication process will be fulfilled within one block. Fig. 5.18 demonstrates two types of thread organization for parallel matrix-vector multiplication within each block. After the updating in line 8 of Algorithm 5.2, the matrix $[DLF]$ is stored in CSR format, which is shown in the bottom of Fig. 5.18. The naive parallel implementation of matrix-vector multiplication is to assign each row into one thread as shown in Fig. 5.18 (a). However, the scattered access will appear when the CSR format is utilized for matrix storage. For example, thread 0 to 3 need to access the first non-zero number in the corresponding row at the same time, but these numbers are discontinuous

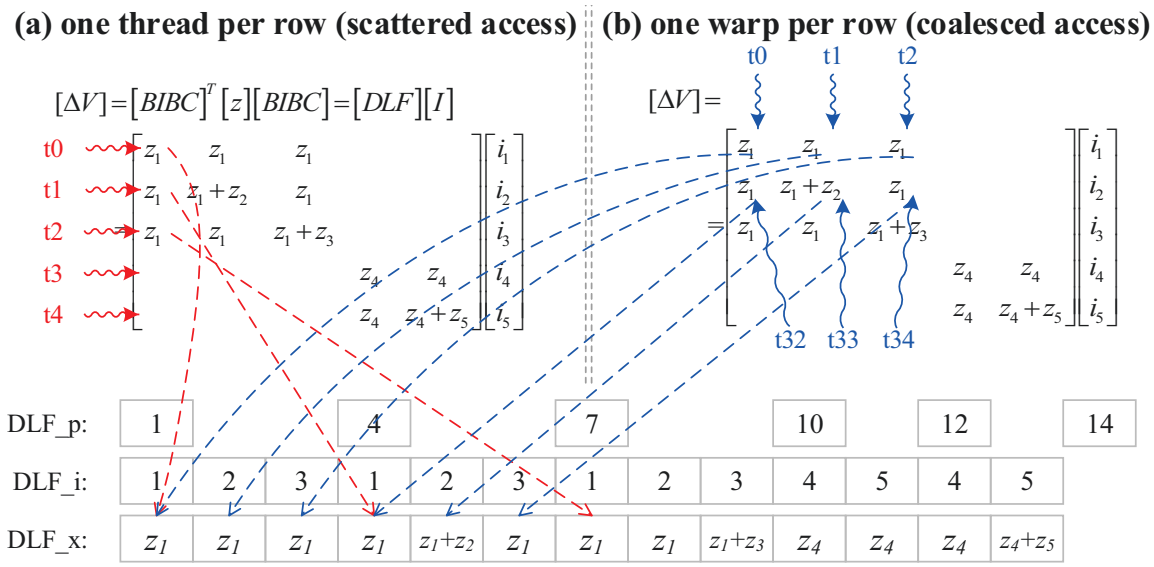


Figure 5.18: Thread organization for parallel matrix-vector multiplication within each block.

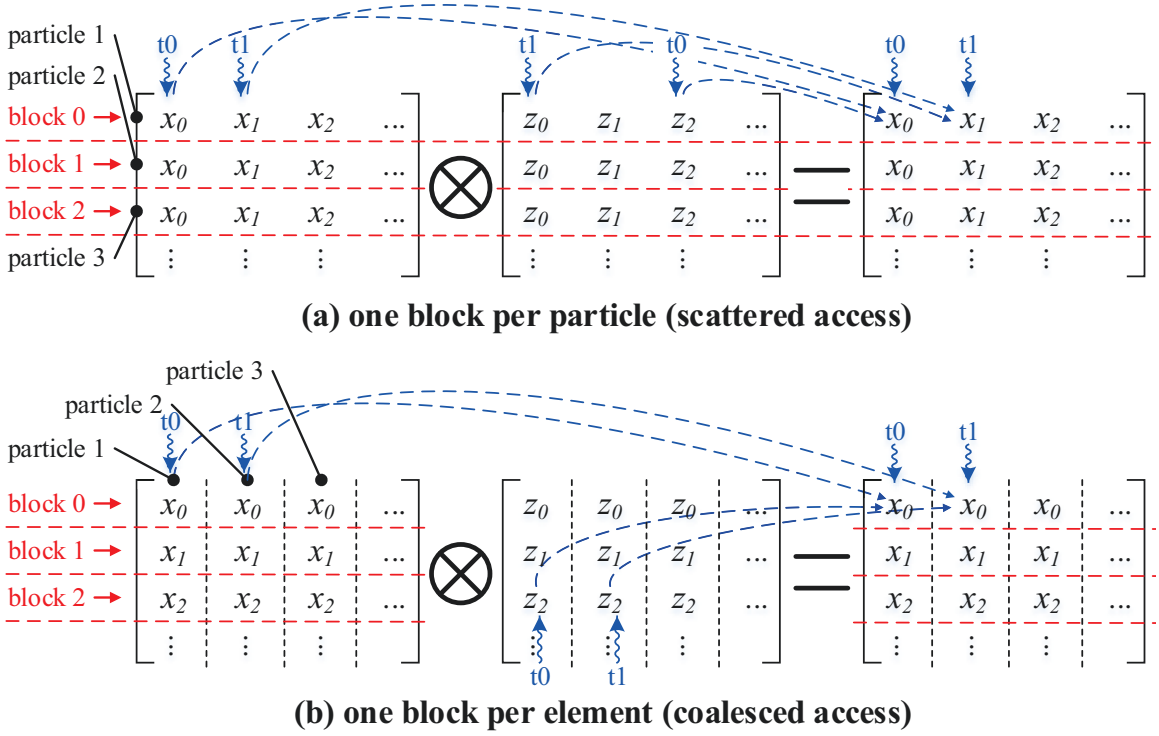


Figure 5.19: Thread organization for parallel irregular mapping.

in the vector x. On the other hand, we assign each row into one warp as shown in Fig. 5.18 (b), where the coalesced access has been achieved with illustrative dashed arrows. It should be noted that the parallel reduction for summation in each row is required.

iv. Parallel Irregular Mapping

Based on the above parallel processing, each step in Algorithm 5.2 can be efficiently executed except for line 8, where irregular mapping is confronted. Actually, line 8 of Algorithm 5.2 is realized with Algorithm 5.3. Conventionally, if each block is assigned to one particle, the scattered access will appear as shown in Fig. 5.19 (a). The reason lies in line 4 of Algorithm 5.3. During the process to access $z(xx(j))$, although j is in successive for each thread, the result of $xx(j)$ is discontinuous, resulting in irregular access of vector $[z]$. In this example, $x(0)$ and $x(1)$ correspond to $z(2)$ and $z(1)$, respectively. In order to achieve the coalesced access, matrices $[x]$ and $[z]$ shown in Fig. 5.19 (a) are transposed, and we distribute each element to one block as shown in Fig. 5.19 (b). For the new scheme, thread 0 and 1 in block 0 are launched to update $x(0)$ for particle 1 and 2. Since $x(0)$ corresponds to $z(2)$ holds for all particles, these two threads should access $z(2)$ of particle 1 and 2, thus the successive addresses are accessed.

v. Parallel Matrix Transpose

In the process of parallel irregular mapping, matrix transpose is involved. In order to coordinate with other processes, parallel matrix transpose should be executed. Fig. 5.20 illustrates two kinds of implementation. As shown in Fig. 5.20 (a), the naive matrix transpose will result in scattered access. Nevertheless, the transposing process is divided into two steps in Fig. 5.20 (b) with the help of shared memory. In step 1, a small piece of the original matrix is copied into shared memory with one warp for each row. Step 2 is comping that piece from shared memory to target address with one warp for each column of the original matrix (when transposed, it appears as one row in the new matrix). It can be seen that in each step, the coalesced access is obtained. The reason for transposing piece by piece is that the size of shared memory is limited.

5.3.4 Case Studies

In this section, four distribution networks retrieved from [167] are employed for the case studies. Based on the original network topology and component data, a lot of VVO devices are added. For simplicity, the parameter of each type of component added into different networks is identical. For each system, a maximum of 30% of the total active demand can be provided with DGs. Each DG has a capacity of 0.5 MW, with a power factor between -0.9 (lagging) and 0.9 (leading). A maximum of 10% of the total reactive demand can be supported by SC. The number of switchable stages for each SC is 5, and the stage step is 0.04 MVar. The number of OLTC and medium-length line equal to 10% and 1% of the total number of branches, respectively. Each OLTC has a tap ratio from 0.9 to 1.1 and can be divided into 21 steps. Table 5.8 summarizes the basic configuration of each system with the number of components. It should be noted that all the detailed data is randomly gen-

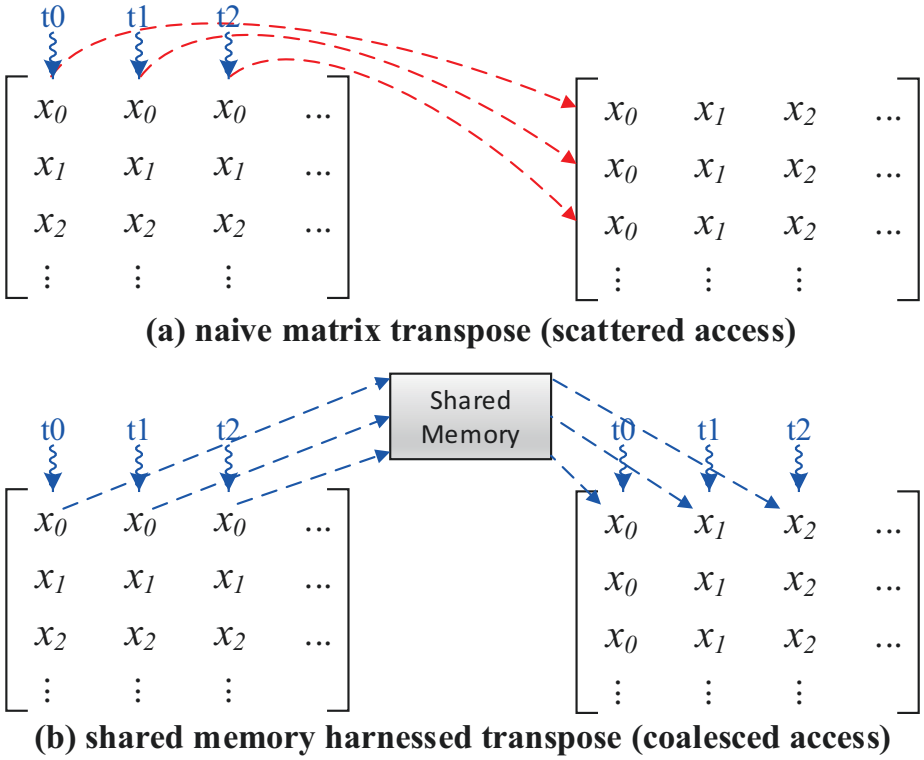


Figure 5.20: Thread organization for parallel matrix transpose.

erated according to Table 5.8, such as where to install these components. Since different types of implementation utilize the same randomly generated input data, the comparison on solution accuracy and efficiency is reasonable.

In order to validate the performance of the GPU-based parallel RTVVO, four types of implementation have been carried out for comparison:

- **CPU_M:** PSO framework given in Fig. 5.13, where the fitness evaluation of each particle is performed by Matpower with the built-in Newton-Raphson method;
- **CPU_S:** sequential version of **Algorithm 5.2** in CPU with C++;
- **CPU_P:** parallel version of **Algorithm 5.2** in CPU with OpenMP, where 12 threads are launched;
- **GPU_P:** parallel version of **Algorithm 5.2** in GPU with CUDA.

For each implementation, the population size of PSO is set as 512, and the maximum number of iteration is 200. All tests are implemented on the same platform including: Intel Xeon E5-2620 CPU with 32GB RAM, Nvidia GeForce GTX 1080 GPU, Matlab version 2017a, CUDA version 8.0, and Visual Studio 2015.

Table 5.8: The number of components for different test systems.

Cases	Branches	DG	SC	OLTC	Medium-length line
136-bus	135	12	4	14	1
415-bus	415	86	52	42	4
880-bus	873	74	37	87	9
1760-bus	1746	150	74	175	17

Table 5.9: The average active power loss error between different implementations for the 1760-bus system in 20 trials.

Methods	CPU_M	CPU_S	CPU_P	GPU_P
CPU_M	0	4.91×10^{-7}	4.56×10^{-7}	9.16×10^{-7}
CPU_S	4.91×10^{-7}	0	6.21×10^{-9}	8.13×10^{-9}
CPU_P	4.56×10^{-7}	6.21×10^{-9}	0	2.79×10^{-9}
GPU_P	9.16×10^{-7}	8.13×10^{-9}	2.79×10^{-9}	0

5.3.4.1 Solution Validation

Although the main objective of this work is achieving high solution efficiency from parallel processing with GPU, the accuracy and convergence property of RTVVO should be guaranteed. Without loss of generality, the 1760-bus system is employed for validation.

i. Accuracy

The DA developed in section 5.3.1 is intensively integrated into **Algorithm 5.2**, thus its accuracy dominates the quality of final results. In order to validate its accuracy, 20 PSO particles are randomly generated and solved with either DA or Matpower in all four implementation environments. The average difference on the obtained active power loss for different methods is reported in Table 5.9. It can be seen that the error between Matpower and DA is smaller than 10^{-6}, which means the accuracy of the developed DA is acceptable. On the other hand, the difference of DA running on different platforms is smaller than 10^{-8}, indicating that parallel implementation does not spoil the accuracy.

ii. Convergence property

According to the above accuracy analysis, it can be concluded that the difference in the power flow calculation is negligible. Thus the convergence test is devoted to the validation of the PSO framework. Based on the same input data and PSO parameters, the convergence property of different types of implementation corresponding to iteration is demonstrated in Fig. 5.21. Although there are fluctuations in each line, the convergence property is similar: 1) the objective value drops very fast in the first 40 iterations, indicating that the PSO is effective to find a high-quality solution for this problem; 2) the improvement during the next 40 iteration is still observable, which is due to the disturbance

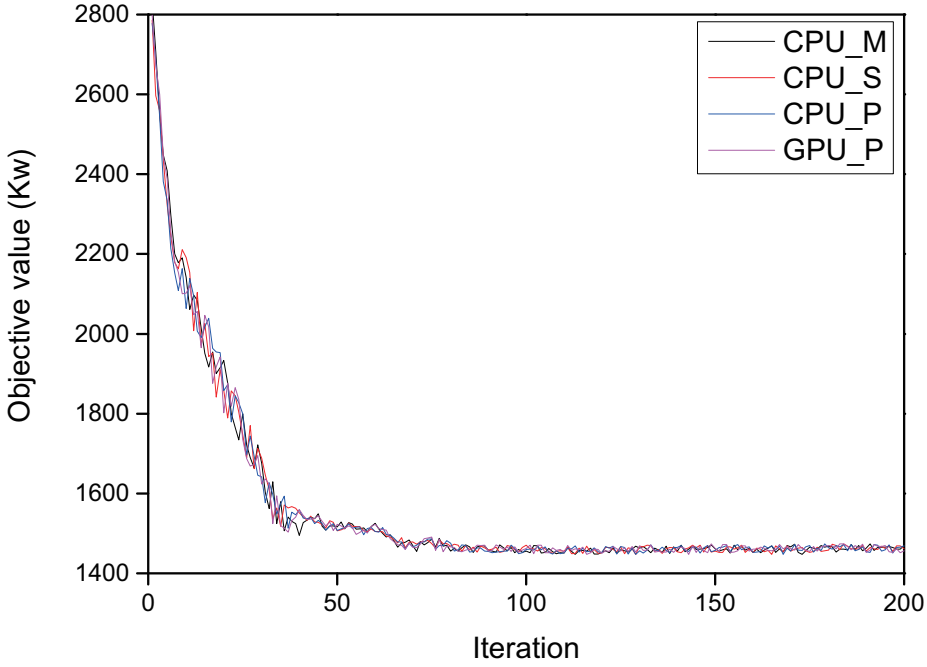


Figure 5.21: Convergence property of different types of implementation.

Table 5.10: The number of atom operations for the updating of $[DLF]$ with two different methods.

Cases	Dim.	$[BIBC]$		$[DLF]$		Atom operations		Imp.
		Nnz	Sparsity	Nnz	Sparsity	M-M mult.	**Alg. 5.3**	
136-bus	135	976	5.36%	2,465	13.53%	26,504	12,746	51.91%
415-bus	415	1,950	1.13%	3,585	2.08%	28,010	13,030	53.48%
880-bus	873	23,147	3.04%	125,285	16.44%	4,220,153	2,098,503	50.27%
1760-bus	1746	46,294	1.52%	250,570	8.22%	8,440,306	4,197,006	50.27%

introduced in (5.40); 3) finally, the population turns to be stable in the remaining iterations, showing that the particles searching around the obtained global best optimal.

5.3.4.2 Solution Efficiency

In this subsection, the improvement on the solution efficiency gained by two proposals given in section 5.3.3 will be exemplified on four test systems.

i. Performance improvement gained by data structure design

As shown in Fig. 5.15, the number of atom operations required by **Algorithm 5.3** is smaller than the matrix-matrix multiplication. Quantitative results on four large systems are given in Table 5.10. It can be seen that the data structure redesign is beneficial to reduce over

Table 5.11: Execution time of the RTVVO for various implementation schemes.

Cases	Total RTVVO execution time (s)				Avg. exe. time for single ACPF (ms)			
	CPU_M	CPU_S	CPU_P	GPU_P	CPU_M	CPU_S	CPU_P	GPU_P
136-bus	574.46	47.34	4.64	2.37	5.61	0.46	0.045	0.023
415-bus	1,295.36	96.36	8.95	3.06	12.65	0.94	0.087	0.030
880-bus	2,744.32	184.88	16.77	3.80	26.80	1.81	0.164	0.037
1760-bus	5,149.69	336.92	29.55	5.41	50.29	3.29	0.289	0.053

Table 5.12: Achieved speedup over CPU_M for various methods.

Cases	CPU_M	CPU_S	CPU_P	GPU_P
136-bus	1.00	12.13	123.81	242.39
415-bus	1.00	13.44	144.73	423.32
880-bus	1.00	14.84	163.64	722.19
1760-bus	1.00	15.28	174.27	951.88

50% of the atom operations for all test systems although the sparsity of $[BIBC]$ and $[DLF]$ are different. In addition to the superiority of fewer atom operations, the number of index calculation when traversing CSR matrices is smaller for **Algorithm 5.3**. Therefore, the **Algorithm 5.3** is at least two times faster than the matrix-matrix multiplication.

ii. Performance improvement gained by GPU parallel implementation

In order to validate the performance of various methods, all of them are executed for 20 times with the same input data. One of them has been demonstrated with Fig. 5.21. The average execution time for different methods is collected in Table 5.11. It can be seen that the CPU_M takes much longer time than the other methods for all test systems. The reason is two-fold: 1) the complexity of Newton-Raphson method is higher than DA; 2) the execution efficiency of C++ code is higher than the Matlab code. Although the CPU_S is much faster than CPU_M, its application in real-time circumstance is still questionable since minutes of time is required. According to the data reported in Table 5.11, the capability of CPU_P and GPU_P for practical application is promising. A similar conclusion can also be drawn if the average ACPF execution time is considered.

Taking the execution time of CPU_M for different systems as the basis, the achieved speedup can be obtained as shown in Table. 5.12. The superiority of DA over Matpower in this kind of problem is established. Consider CPU_S as the benchmark in Table 5.13, the parallel efficiency of OpenMP and CUDA are satisfactory with ranges from 10.20 to 11.40 and from 19.97 to 62.28, respectively. In addition, the speedup goes higher as the system scale increases.

Table 5.13: Achieved speedup over CPU_S for various methods.

	CPU_M	CPU_S	CPU_P	GPU_P
136-bus	0.082	1.00	10.20	19.97
415-bus	0.074	1.00	10.77	31.49
880-bus	0.067	1.00	11.02	48.65
1760-bus	0.065	1.00	11.40	62.28

5.4 Summary

With the objective of minimizing total active power loss of distribution network, acceleration strategies are investigated in this chapter on two problems: DNRC and RTVVO.

Two main concerns are intensively involved in the solution of DNRC: preserving the RTS and calculating the DNPF. In this chapter, an effective decimal encoding and decoding technique are proposed, which guarantees the RTS with the least effort when compared with other state-of-the-art methods. In addition, the solution process of DNPF is enhanced by the introduction of adjacency and path matrices from graph theory. Case studies are conducted on five benchmark systems. For individual comparison, the developed methods outperform their advanced counterparts respectively. When integrated into the standard PSO framework, the superiority of the proposals is still established.

PSO is also employed as the solution framework for RTVVO, where the full AC power flow is tackled with the DA method. In the iterative process of DA, the detailed mathematical models for DG and other VVO components are integrated. Although PSO and DA are suitable for parallel processing, the best performance is far away to be reached by the naive implementation, therefore designs on data structure and thread organization are proposed. After tuning, all threads in one warp are assigned to access the successive address, i.e., coalesced access is achieved. In the case study, four systems with the size ranging from 136-bus to 1760-bus are introduced. The results indicate that the accuracy and convergence property is satisfactory, and the parallel efficiency is suitable for practical application.

6 Conclusions and Future Works

Although electricity is ubiquitous in modern life, its availability and reliability are not granted. Actually, they are highly dependent on the robust planning and stable operation of power system, which consists of generation, transmission, and distribution systems. In addition to the security concern, economic profit/cost is another important issue that receives great attention. Therefore, a lot of optimization problems are proposed to minimize the cost with respect to various security constraints. Due to non-convex properties, discrete variables, and large system scales, the problem solution process presents great challenges for various algorithms, solvers, and platforms. In order to address these concerns, eight problems are investigated in this thesis for the purpose of performance acceleration via algorithm customization, framework development, and parallel processing, based on interdisciplinary backgrounds of power systems, operations research, and computer engineering.

6.1 Contributions of Thesis

The main contributions of this thesis can be summarized as follows:

- **MGPSO and LU Decomposition for TEP**
  The MGPSO algorithm framework is proposed based on the discrete PSO and several beneficial enhancements, such as Sobol sequence initialization method, multi-group co-evolution strategy, and mutation mechanism. Specifically, the fitness evaluation of MGPSO is accelerated by the substitution of LP with LES, whose solution process is very efficient with optimized LU decomposition approach. Superiority over com-

mercial software Lingo 11.0 is established with case studies.

- **Branch-and-Cut Benders Decomposition for SCTEP**
  The BCBD algorithm is developed with the integration of BD into the B&C framework, where the original MILP master problem of BD is relaxed into LP, resulting in lower complexity and shorter execution time. In addition, four acceleration strategies have been employed to enhance the performance, including two-phase method, multi-cut strategy, valid inequality, and optimal preconditioning. Better performance of BCBD over commercial software ILOG-Cplex 12.5.1 is validated with numerical experiments.

- **Fast Decoupled Method and Direct Linear Solver for ACPF with GPU**
  GPU implementation of FD for the solution of ACPF with direct linear solver is carried out with both Matlab and CUDA. Implementation platforms (CPU and GPU), linear equations solution strategies (LU decomposition and left division), data storage formats (dense and sparse), and fill-in reduction algorithms (RCM and AMD) are compared and discussed, indicating that sparse LU decomposition with AMD running on GPU is promising for practical application.

- **Compensation Method for RTCA with GPU**
  Instead of the FD method, CM is determined for the solution of RTCA. Detailed GPU implementation schemes are elaborated, including various strategies and principles for data structure definition, kernel function design, and memory management. Comprehensive comparisons with open-source package Matpower and state-of-the-art parallel computing methods are provided, where the advantage of parallel CM with GPU is revealed.

- **Robust Optimization Frameworks for SCUC**
  In order to reveal the capability of different methods on the solution of SCUC, both explicit and implicit decomposition frameworks are investigated, as well as their inner feedback strategies, such as BD and CCG algorithm. In addition, sensitivity analysis, multi-cut strategy, and parallel implementation are analyzed and discussed. Results indicate that 1) the introduction of Benders cuts may even drag the solution efficiency of CCG; 2) the parallel implementation of explicit method is proportionate with the implicit method; 3) the decomposition framework is superior over commercial solver for this kind of problem.

- **Primal-Dual Interior Point Method for RTOPF with GPU on Batched Mode**
  To improve the solution efficiency and accuracy of RTOPF, a three-stage framework for parallel processing is employed. In Stage 1, uncertainties from renewable generators and demand loads are characterized with scenarios. Large numbers of RTOPFs corresponding to each scenario are formulated and addressed in Stage 2 with PDIPM, where the linear systems are regulated into the same sparsity pattern and then tackled in a batched style with GPU. Results from Stage 2 are utilized in Stage 3 to perform a hot-start RTOPF, where the forecasting error can be minimized. The superiority of batched GPU solution is validated by comparisons with regular GPU, parallel CPU, and sequential CPU implementations.

- **Direct Approach and Graph Theory for DNRC**
  Due to mixed-integer and non-linear properties, DNRC problem has been widely addressed with meta-heuristic algorithms. In order to accelerate the solution process, two essential components of meta-heuristic algorithms are investigated: solution representation and fitness evaluation. Instead of the popular binary and integer numbers, decimal encoding is employed. An efficient decoding process is proposed and validated. In addition, acceleration on the DA is also developed based on the adjacency matrix and path matrix from graph theory. Results indicate that the proposals significantly improve the solution efficiency without the loss of accuracy.

- **PSO and GPU for RTVVO**
  The full AC RTVVO is formulated based on PSO framework and DA power flow method, where all components, such as DG and OLTC transformer, are formulated and integrated into the iterative DA process. Since both PSO and DA are suitable for parallel implementation, the GPU is introduced for acceleration in order to achieve the possibility for real-time application. Schemes on the design of data structure and thread organization pattern are developed to achieve coalesced access within each warp of threads. Based on the results from solution efficiency comparison between CPU and GPU parallel programs, the promise of the proposed implementation scheme for practical application is established.

6.2 Directions for Future Work

The following topics are proposed for future work:

- The DC and disjunctive model are utilized for TEP and SCTEP, respectively. Al-

though they are sufficient for long-term planning, more practical but complicated AC model is demanded for short-term planning.

- In realistic power systems, the generation and transmission sectors are highly interrelated. Therefore, SCTEP can be performed with the association of generation expansion planning.
- The full potential of Matlab has not been explored with GPU for ACPF since there is no sparse LU decomposition solver available. It is believed that an advanced solver will be provided shortly, and the ACPF can be efficiently addressed with GPU and Matlab.
- RTCA is employed to recognize the possible outages of power system, providing instructions for post-contingency corrective transmission switching. Therefore, further utilization and analysis of RTCA results can be fulfilled to achieve higher reliability standards.
- Proposed methods for RTVVO can be expanded to include more details in network or generator modeling. It is also possible to change the number of processor nodes or GPUs which are running in parallel.
- It is possible to implement other parallel processing based techniques to investigate higher speed-ups. It is predicted that if a method accelerates the CPU-based program, it would also accelerate the GPU-based model if that approach is efficiently implemented on the GPU.
- The application of the proposed method is not limited to the power system analysis, so future research can also be done to develop GPU-based algorithms for other optimization problems, such as operations research.
- The majority of investigated problems are static, in the future, extensions will be put on more practical concerns, such as dynamics and uncertainties of power systems.

Bibliography

[1] United States Department of Energy, "Simple diagram of electricity grids in North America," [Online]. Available:, https://commons.wikimedia.org/wiki/File:Electricity_grid_simple-_North_America.svg.

[2] NREL Solar Radiation Research Laboratory, "Daily plots and raw data files (January 1, 2015 to November 30, 2017)," [Online], available: http://midcdmz.nrel.gov/srrl_bms/.

[3] U.S.-Canada Power System Outage Task Force, "Final report on the august 14, 2003 blackout in the United States and Canada: causes and recommendations," Tech. Rep., Apr. 2004.

[4] J. F. Benders, "Partitioning procedures for solving mixed-variables programming problems," *Numer. Math.*, vol. 4, no. 1, pp. 238–252, 1962.

[5] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski, *Robust optimization*. Princeton University Press, 2009.

[6] H. Wang, C. E. Murillo-Sanchez, R. D. Zimmerman, and R. J. Thomas, "On computational issues of market-based optimal power flow," *IEEE Trans. Power Syst.*, vol. 22, no. 3, pp. 1185–1193, Aug. 2007.

[7] T. Cui, "Power system probabilistic and security analysis using commodity high performance computing systems," Ph.D. dissertation, Carnegie Mellon University, 2013.

[8] NVIDIA, "CUDA C programming guide 8.0," Santa Clara, USA, 2017.

[9] L. L. Garver, "Transmission network estimation using linear programming," *IEEE Trans. Power Appar. Syst.*, vol. PAS-89, no. 7, pp. 1688–1697, Sept. 1970.

[10] A. Moreira, A. Street, and J. M. Arroyo, "An adjustable robust optimization approach for contingency-constrained transmission expansion planning," *IEEE Trans. Power Syst.*, vol. 30, no. 4, pp. 2013–2022, Jul. 2015.

[11] E. A. M. Cesea, T. Capuder, and P. Mancarella, "Flexible distributed multienergy generation system expansion planning under uncertainty," *IEEE Trans. Smart Grid*, vol. 7, no. 1, pp. 348–357, Jan. 2016.

[12] A. J. Wood, B. F. Wollenberg, and G. B. Sheble, *Power generation, operation, and control*, 3rd ed. New York, NY, USA: John Wiley & Sons, 2013.

[13] I. de J. Silva, M. J. Rider, R. Romero, A. V. Garcia, and C. A. Murari, "Transmission network expansion planning with security constraints," *IEE Proc. - Gener. Transm. Distrib.*, vol. 152, no. 6, pp. 828–836, Nov. 2005.

[14] K. W. Hedman, M. C. Ferris, R. P. O'Neill, E. B. Fisher, and S. S. Oren, "Co-optimization of generation unit commitment and transmission switching with N-1 reliability," *IEEE Trans. Power Syst.*, vol. 25, no. 2, pp. 1052–1063, May 2010.

[15] G. Latorre, R. D. Cruz, J. M. Areiza, and A. Villegas, "Classification of publications and models on transmission expansion planning," *IEEE Trans. Power Syst.*, vol. 18, no. 2, pp. 938–946, May. 2003.

[16] R. Hemmati, R.-A. Hooshmand, and A. Khodabakhshian, "Comprehensive review of generation and transmission expansion planning," *IET Gen., Transm., Distrib.*, vol. 7, no. 9, pp. 955–964, Sept. 2013.

[17] R. Romero, A. Monticelli, A. Garcia, and S. Haffner, "Test systems and mathematical models for transmission network expansion planning," *Proc. Inst. Elect. Eng., Gen., Transm., Distrib.*, vol. 149, no. 1, pp. 27–36, Jan. 2002.

[18] X. Li, P. Balasubramanian, M. Sahraei-Ardakani, M. Abdi-Khorsand, K. W. Hedman, and R. Podmore, "Real-time contingency analysis with corrective transmission switching," *IEEE Trans. Power Syst.*, vol. 32, no. 4, pp. 2604–2617, Jul. 2017.

[19] J. Zhu, *Optimization of power system operation*, 2nd ed. New York, NY, USA: John Wiley & Sons, 2015.

[20] Q. Wang, J. P. Watson, and Y. Guan, "Two-stage robust optimization for $N - k$ contingency-constrained unit commitment," *IEEE Trans. Power Syst.*, vol. 28, no. 3, pp. 2366–2375, Aug. 2013.

[21] J. Carpentier, "Contribution to the economic dispatch problem," *Bull. Soc. Franc. Electr.*, vol. 3, no. 8, pp. 431–447, Aug. 1962.

[22] E. Mohagheghi, A. Gabash, and P. Li, "Real-time optimal power flow under wind energy penetration - part I: Approach," in *Proc. IEEE Int. Conf. Environ. Electr. Eng.*, Florence, Italy, Jun. 2016, pp. 1–6.

[23] ——, "Real-time optimal power flow under wind energy penetration - part II: Implementation," in *Proc. IEEE Int. Conf. Environ. Electr. Eng.*, Florence, Italy, Jun. 2016, pp. 1–6.

[24] J. C. Lpez, M. Lavorato, J. F. Franco, and M. J. Rider, "Robust optimisation applied to the reconfiguration of distribution systems with reliability constraints," *IET Gener. Transm. Distrib.*, vol. 10, no. 4, pp. 917–927, Mar. 2016.

[25] M. Arun and P. Aravindhababu, "A new reconfiguration scheme for voltage stability enhancement of radial distribution systems," *Energy Convers. Manage.*, vol. 50, no. 9, pp. 2148–2151, Sept. 2009.

[26] J. M. Harris, J. L. Hirst, and M. J. Mossinghoff, *Combinatorics and graph theory*. New York, NY, USA: Springer, 2008.

[27] V. Roberge, M. Tarbouchi, and F. A. Okou, "Distribution system optimization on graphics processing unit," *IEEE Trans. Smart Grid*, vol. 8, no. 4, pp. 1689–1699, Jul. 2017.

[28] H. Ahmadi, J. R. Mart, and H. W. Dommel, "A framework for Volt-VAR optimization in distribution systems," *IEEE Trans. Smart Grid*, vol. 6, no. 3, pp. 1473–1483, May 2015.

[29] W. Zheng, W. Wu, B. Zhang, and Y. Wang, "Robust reactive power optimisation and voltage control method for active distribution networks via dual time-scale coordination," *IET Gener. Transm. Distrib.*, vol. 11, no. 6, pp. 1461–1471, May 2017.

[30] A. T. Saric and A. M. Stankovic, "A robust algorithm for Volt/Var control," in *Proc. IEEE/PES Power Syst. Conf. Expo.*, Seattle, WA, USA, Mar. 2009, pp. 1–8.

[31] A. Mohapatra, P. R. Bijwe, and B. K. Panigrahi, "An efficient hybrid approach for Volt/Var control in distribution systems," *IEEE Trans. Power Delivery*, vol. 29, no. 4, pp. 1780–1788, Aug. 2014.

[32] A. Borghetti, "Using mixed integer programming for the volt/var optimization in distribution feeders," *Electr. Power Syst. Res.*, vol. 98, pp. 39–50, May 2013.

[33] S. Rahimi, K. Zhu, S. Massucco, and F. Silvestro, "Stochastic Volt-Var optimization function for planning of MV distribution networks," in *Proc. IEEE Power Energy Soc. Gen. Meeting*, 2015, pp. 1–5.

[34] X. Fang, F. Li, Y. Wei, R. Azim, and Y. Xu, "Reactive power planning under high penetration of wind energy using benders decomposition," *IET Gener. Transm. Distrib.*, vol. 9, no. 14, pp. 1835–1844, Oct. 2015.

[35] Z. Wang, J. Wang, B. Chen, M. M. Begovic, and Y. He, "MPC-based voltage/var optimization for distribution circuits with distributed generators and exponential load models," *IEEE Trans. Smart Grid*, vol. 5, no. 5, pp. 2412–2420, Sept. 2014.

[36] Y. Malachi and S. Singer, "A genetic algorithm for the corrective control of voltage and reactive power," *IEEE Trans. Power Syst.*, vol. 21, no. 1, pp. 295–300, Feb. 2006.

[37] A. Ulinuha, M. A. S. Masoum, and S. Islam, "Hybrid genetic-fuzzy algorithm for volt/var/total harmonic distortion control of distribution systems with high penetration of non-linear loads," *IET Gener. Transm. Distrib.*, vol. 5, no. 4, pp. 425–439, Apr. 2011.

[38] Y. Fukuyama, "Parallel particle swarm optimization for reactive power and voltage control verifying dependability," in *Proc. IEEE Congr. Evol. Comput.*, Sendai, Japan, May 2015, pp. 304–310.

[39] T. Niknam, B. B. Firouzi, and A. Ostadi, "A new fuzzy adaptive particle swarm optimization for daily volt/var control in distribution networks considering distributed generators," *Applied Energy*, vol. 87, no. 6, pp. 1919–1928, Jun. 2010.

[40] D. Chaudhary, W. Sun, Q. Zhou, and A. Golshani, "Chance-constrained real-time volt/var optimization using simulated annealing," in *Proc. IEEE Power Energy Soc. Gen. Meeting*, Denver, CO, USA, Jul. 2015, pp. 1–5.

[41] M. Manbachi, A. Sadu, H. Farhangi, A. Monti, A. Palizban, F. Ponci, and S. Arzanpour, "Real-time co-simulation platform for smart grid volt-var optimization using IEC 61850," *IEEE Trans. on Ind. Inf.*, vol. 12, no. 4, pp. 1392–1402, Aug. 2016.

[42] A. Zakariazadeh, H. Modaghegh, and S. Jadid, "Real time volt/var control using advance metering infrastructure system in FAHAM project," in *roc. Int. Conf. Exhi. Electr. Distrib.*, Stockholm, Sweden, Jun. 2013, pp. 1–4.

[43] M. Manbachi, A. Sadu, H. Farhangi, A. Monti, A. Palizban, F. Ponci, and S. Arzanpour, "Real-time co-simulated platform for novel volt-var optimization of smart distribution network using AMI data," in *Proc. IEEE Int. Conf. Smart Energy Grid Eng.*, Oshawa, Canada, Aug. 2015, pp. 1–7.

[44] H. Zhang, V. Vittal, G. T. Heydt, and J. Quintero, "A mixed-integer linear programming approach for multi-stage security-constrained transmission expansion planning," *IEEE Trans. Power Syst.*, vol. 27, no. 2, pp. 1125–1133, May 2012.

[45] S. H. M. Hashimoto, R. Romero, and J. R. S. Mantovani, "Efficient linear programming algorithm for the transmission network expansion planning problem," *IEE Proc. Gener., Transm. Distrib.*, vol. 150, no. 5, pp. 536–542, Sept. 2003.

[46] R. Romero, R. A. Gallego, and A. Monticelli, "Transmission system expansion planning by simulated annealing," *IEEE Trans. Power Syst.*, vol. 11, no. 1, pp. 364–369, Feb. 1996.

[47] R. Romero, E. N. Asada, E. Carreno, and C. Rocha, "Constructive heuristic algorithm in branch-and-bound structure applied to transmission network expansion planning," *IET Gener. Transm. Distrib.*, vol. 1, no. 2, pp. 318–323, Mar. 2007.

[48] P. Murugan, "Modified particle swarm optimisation with a novel initialisation for finding optimal solution to the transmission expansion planning problem," *IET Gener., Transm., Distrib.*, vol. 6, no. 11, pp. 1132–1142, Nov. 2012.

[49] P. Gavela, J. L. Rueda, A. Vargas, and I. Erlich, "Performance comparison of heuristic optimization methods for optimal dynamic transmission expansion planning," *Int. Trans. Electr. Energy Syst.*, vol. 24, no. 10, pp. 1450–1472, Aug. 2014.

[50] G.-R. Kamyab, M. Fotuhi-Firuzabad, and M. Rashidinejad, "A PSO based approach for multi-stage transmission expansion planning in electricity markets," *Int. J. Electr. Power Energy Syst.*, vol. 54, pp. 91–100, Jan. 2014.

[51] H. Shayeghi, M. Mahdavi, and A. Bagheri, "An improved DPSO with mutation based on similarity algorithm for optimization of transmission lines loading," *Energy Convers. Manage.*, vol. 51, no. 12, pp. 2715–2723, Dec. 2010.

[52] M. C. Da Rocha and J. T. Saraiva, "A discrete evolutionary PSO based approach to the multiyear transmission expansion planning problem considering demand uncertainties," *Int. J. Electr. Power Energy Syst.*, vol. 45, no. 1, pp. 427–442, Feb. 2013.

[53] S. P. Torres, C. A. Castro, and M. J. Rider, "Transmission expansion planning by using DC and AC models and particle swarm optimization," *Swarm Intell. Electr. Electron. Eng.*, pp. 260–284, 2012.

[54] E. Karimi and A. Ebrahimi, "Inclusion of blackouts risk in probabilistic transmission expansion planning by a multi-objective framework," *IEEE Trans. Power Syst.*, vol. 30, no. 5, pp. 2810–2817, Sep. 2015.

[55] S. Binato, M. V. F. Pereira, and S. Granville, "A new Benders decomposition approach to solve power transmission network design problems," *IEEE Trans. Power Syst.*, vol. 16, no. 2, pp. 235–240, May 2001.

[56] S. Asadamongkol and B. Eua-arporn, "Application of Benders decomposition to transmission expansion planning with N-1 security constraints," *IEEJ Trans. Electr. Electron. Eng.*, vol. 6, no. 2, pp. 127–133, Mar. 2011.

[57] S. Lumbreras and A. Ramos, "Transmission expansion planning using an efficient version of Benders' decomposition. a case study," in *Proc. IEEE PowerTech*, Grenoble, France, Jun. 2013, pp. 1–7.

[58] M. Jenabi, S. M. T. Fatemi Ghomi, S. A. Torabi, and S. H. Hosseinian, "Acceleration strategies of Benders decomposition for the security constraints power system expansion planning," *Ann. Oper. Res.*, vol. 235, no. 1, pp. 337–369, Dec. 2015.

[59] O. Alizadeh-Mousavi and M. Zima-Bočkarjova, "Efficient Benders cuts for transmission expansion planning," *Elect. Power Syst. Res.*, vol. 131, pp. 275–284, Feb. 2016.

[60] H. D. Sherali and B. J. Lunday, "On generating maximal nondominated Benders cuts," *Ann. Oper. Res.*, vol. 210, no. 1, pp. 57–72, Apr. 2013.

[61] X. Wang, Y. Song, and M. Irving, *Modern power systems analysis*. New York, NY, USA: Springer, 2008.

[62] C. Thompson, K. McIntyre, S. Nuthalapati, A. Garcia, and E. A. Villanueva, "Real-time contingency analysis methods to mitigate congestion in the ERCOT region," in *Proc. IEEE Power Energy Soc. Gen. Meeting,*, Calgary, AB, Canada, Jul. 2009, pp. 1–7.

[63] Y. Chen, Z. Huang, and M. Rice, "Evaluation of counter-based dynamic load balancing schemes for massive contingency analysis on over 10,000 cores," in *Proc. SC Comp.: High Perform. Comput. Networking Storage Anal.*, Nov. 2012, pp. 341–346.

[64] A. Mittal, J. Hazra, N. Jain, V. Goyal, D. P. Seetharam, and Y. Sabharwal, "Real time contingency analysis for power grids," in *Proc. Euro-Par 2011 Parallel Processing*, Bordeaux, France, Sept. 2011, pp. 303–315.

[65] Z. Huang, Y. Chen, and J. Nieplocha, "Massive contingency analysis with high performance computing," in *Proc. IEEE Power Energy Soc. Gen. Meeting,*, Calgary, AB, Canada, Jul. 2009, pp. 1–8.

[66] F. Garcia, N. D. R. Sarma, V. Kanduri, G. Nissankala, K. Gopinath, J. Polusani, T. Mortensen, and I. Flores, "ERCOT control center experience in using real-time contingency analysis in the new nodal market," in *Proc. IEEE Power Energy Soc. Gen. Meeting,*, San Diego, CA, USA, Jul. 2012, pp. 1–8.

[67] X. Yang, C. Liu, and J. Wang, "Large-scale branch contingency analysis through master/slave parallel computing," *J. Mod. Power Syst. Clean Energy*, vol. 1, no. 2, pp. 159–166, Sept. 2013.

[68] T. Cui, R. Yang, G. Hug, and F. Franchetti, "Accelerated AC contingency calculation on commodity multi-core SIMD CPUs," in *Proc. IEEE Power Energy Soc. Gen. Meeting,*, MD, USA, Jul. 2014, pp. 1–5.

[69] A. Gopal, D. Niebur, and S. Venkatasubramanian, "DC power flow based contingency analysis using graphics processing units," in *Proc. IEEE Power Tech.*, Lausanne, Switzerland, Jul. 2007, pp. 731–736.

[70] X. Li and F. Li, "GPU-based power flow analysis with chebyshev preconditioner and conjugate gradient method," *Electr. Power Syst. Res.*, vol. 116, pp. 87–93, Nov. 2014.

[71] G. Zhou, X. Zhang, Y. Lang, R. Bo, Y. Jia, J. Lin, and Y. Feng, "A novel GPU-accelerated strategy for contingency screening of static security analysis," *Int. J. Electr. Power Energy Syst.*, vol. 83, pp. 33 – 39, 2016.

[72] C. Guo, B. Jiang, H. Yuan, Z. Yang, L. Wang, and S. Ren, "Performance comparisons of parallel power flow solvers on GPU system," in *Proc. IEEE 18th Int. Conf. Embedded Real-Time Comput. Syst. Appl. (RTCSA)*, Seoul, Korea, Aug. 2012, pp. 232–239.

[73] V. Roberge, M. Tarbouchi, and F. Okou, "Parallel power flow on graphics processing units for concurrent evaluation of many networks," *IEEE Trans. Smart Grid*, vol. PP, no. 99, pp. 1–10, Nov. 2015.

[74] D. Chen, H. Jiang, Y. Li, and D. Xu, "A two-layered parallel static security assessment for large-scale grids based on GPU," *IEEE Trans. Smart Grid*, vol. 8, no. 3, pp. 1396–1405, May 2017.

[75] G. Zhou, Y. Feng, R. Bo, L. Chien, X. Zhang, Y. Lang, Y. Jia, and Z. Chen, "GPU-accelerated batch-ACPF solution for N-1 static security analysis," *IEEE Trans. Smart Grid*, vol. 8, no. 3, pp. 1406–1416, May 2017.

[76] J. Singh and I. Aruni, "Accelerating power flow studies on graphics processing unit," in *Proc. Annu. IEEE India Conf.*, Kolkata, India, Dec. 2010, pp. 1–5.

[77] M. Marin, D. Defour, and F. Milano, "Asynchronous power flow on graphic processing units," in *Proc. Euro. Int. Conf. Parallel Distrib. Network Process.*, St. Petersburg, Russia, Mar. 2017, pp. 255–261.

[78] X. Li, F. Li, H. Yuan, H. Cui, and Q. Hu, "GPU-based fast decoupled power flow with preconditioned iterative solver and inexact newton method," *IEEE Trans. Power Syst.*, vol. 32, no. 4, pp. 2695–2703, Jul. 2017.

[79] T. A. Davis, *Direct methods for sparse linear systems*. Philadelphia, PA, USA: SIAM, 2006.

[80] O. Alsac, B. Stott, and W. F. Tinney, "Sparsity-oriented compensation methods for modified network solutions," *IEEE Trans. Power Appar. Syst.*, vol. PAS-102, no. 5, pp. 1050–1060, May 1983.

[81] W. Yuan and Q. Zhai, "Power-based transmission constrained unit commitment formulation with energy-based reserve," *IET Gener. Transm. Distrib.*, vol. 11, no. 2, pp. 409–418, Jan. 2017.

[82] D. Bertsimas, E. Litvinov, X. A. Sun, J. Zhao, and T. Zheng, "Adaptive robust optimization for the security constrained unit commitment problem," *IEEE Trans. Power Syst.*, vol. 28, no. 1, pp. 52–63, Feb. 2013.

[83] P. Xiong and P. Jirutitijaroen, "A stochastic optimization formulation of unit commitment with reliability constraints," *IEEE Trans. Smart Grid*, vol. 4, no. 4, pp. 2200–2208, Dec. 2013.

[84] A. Street, F. Oliveira, and J. M. Arroyo, "Contingency-constrained unit commitment with $n - k$ security criterion: a robust optimization approach," *IEEE Trans. Power Syst.*, vol. 26, no. 3, pp. 1581–1590, Aug. 2011.

[85] A. Street, A. Moreira, and J. M. Arroyo, "Energy and reserve scheduling under a joint generation and transmission security criterion: An adjustable robust optimization approach," *IEEE Trans. Power Syst.*, vol. 29, no. 1, pp. 3–14, Jan. 2014.

[86] N. G. Cobos, J. M. Arroyo, and A. Street, "Least-cost reserve offer deliverability in day-ahead generation scheduling under wind uncertainty and generation and network outages," *IEEE Trans. Smart Grid*, vol. PP, no. 99, pp. 1–14, 2016.

[87] J. P. Ruiz, J. Wang, C. Liu, and G. Sun, "Outer-approximation method for security constrained unit commitment," *IET Gener. Transm. Distrib.*, vol. 7, no. 11, pp. 1210–1218, Nov. 2013.

[88] H. Ye, J. Wang, and Z. Li, "MIP reformulation for max-min problems in two-stage robust SCUC," *IEEE Trans. Power Syst.*, vol. 32, no. 2, pp. 1237–1247, Mar. 2017.

[89] R. A. Jabr, "Tight polyhedral approximation for mixed-integer linear programming unit commitment formulations," *IET Gener. Transm. Distrib.*, vol. 6, no. 11, pp. 1104–1111, Nov. 2012.

[90] S. Xia, X. Luo, K. W. Chan, M. Zhou, and G. Li, "Probabilistic transient stability constrained optimal power flow for power systems with multiple correlated uncertain wind generations," *IEEE Trans. Sustain. Energy*, vol. 7, no. 3, pp. 1133–1144, Jul. 2016.

[91] D. Ke, C. Y. Chung, and Y. Sun, "A novel probabilistic optimal power flow model with uncertain wind power generation described by customized gaussian mixture model," *IEEE Trans. Sustain. Energy*, vol. 7, no. 1, pp. 200–212, Jan. 2016.

[92] Z. S. Zhang, Y. Z. Sun, D. W. Gao, J. Lin, and L. Cheng, "A versatile probability distribution model for wind power forecast errors and its application in economic dispatch," *IEEE Trans. Power Syst.*, vol. 28, no. 3, pp. 3114–3125, Aug. 2013.

[93] S. S. Reddy and P. Bijwe, "Day-ahead and real time optimal power flow considering renewable energy resources," *Int. J. Electr. Power Energy Syst.*, vol. 82, pp. 400–408, Nov. 2016.

[94] R. J. Bessa, V. Miranda, A. Botterud, J. Wang, and E. M. Constantinescu, "Time adaptive conditional kernel density estimation for wind power forecasting," *IEEE Trans. Sustain. Energy*, vol. 3, no. 4, pp. 660–669, Oct. 2012.

[95] A. Webberley and D. W. Gao, "Study of artificial neural network based short term load forecasting," in *Proc. IEEE Power Energy Soc. Gen. Meeting*, Vancouver, BC, Canada, Jul. 2013, pp. 1–4.

[96] J. R. Andrade and R. J. Bessa, "Improving renewable energy forecasting with a grid of numerical weather predictions," *IEEE Trans. Sustain. Energy*, vol. PP, no. 99, pp. 1–10, Apr. 2017.

[97] L. Gan and S. H. Low, "An online gradient algorithm for optimal power flow on radial networks," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 3, pp. 625–638, Mar. 2016.

[98] S. S. Reddy and J. A. Momoh, "Realistic and transparent optimum scheduling strategy for hybrid power system," *IEEE Trans. Smart Grid*, vol. 6, no. 6, pp. 3114–3125, Nov. 2015.

[99] Y. Tang, K. Dvijotham, and S. Low, "Real-time optimal power flow," *IEEE Trans. Smart Grid*, vol. PP, no. 99, pp. 1–11, May 2017.

[100] E. Mohagheghi, A. Gabash, and P. Li, "A framework for real-time optimal power flow under wind energy penetration," *Energies*, vol. 10, no. 4, Apr. 2017.

[101] N. Gupta, A. Swarnkar, and K. R. Niazi, "A modified branch-exchange heuristic algorithm for large-scale distribution networks reconfiguration," in *Proc. IEEE Power Energy Soc. Gen. Meeting*, San Diego, CA, USA, Jul. 2012, pp. 1–7.

[102] Q. Peng and S. H. Low, "Optimal branch exchange for feeder reconfiguration in distribution networks," in *Proc. IEEE Conf. Decis. Control*, Florence, Italy, Dec. 2013, pp. 2960–2965.

[103] E. Miguez, J. Cidras, E. Diaz-Dorado, and J. L. Garcia-Dornelas, "An improved branch-exchange algorithm for large-scale distribution network planning," *IEEE Trans. Power Syst.*, vol. 17, no. 4, pp. 931–936, Nov. 2002.

[104] D. Shirmohammadi and H. W. Hong, "Reconfiguration of electric distribution networks for resistive line losses reduction," *IEEE Trans. Power Delivery*, vol. 4, no. 2, pp. 1492–1498, Apr. 1989.

[105] J.-Y. Fan, L. Zhang, and J. D. McDonald, "Distribution network reconfiguration: single loop optimization," *IEEE Trans. Power Syst.*, vol. 11, no. 3, pp. 1643–1647, Aug. 1996.

[106] F. Ding and K. A. Loparo, "A simple heuristic method for smart distribution system reconfiguration," in *Proc. IEEE Energytech*, Cleveland, OH, USA, May 2012, pp. 1–6.

[107] Y. Ju, W. Wu, B. Zhang, and H. Sun, "Loop-analysis-based continuation power flow algorithm for distribution networks," *IET Gener. Transm. Distrib.*, vol. 8, no. 7, pp. 1284–1292, Jul. 2014.

[108] B. Enacheanu, B. Raison, R. Caire, O. Devaux, W. Bienia, and N. HadjSaid, "Radial network reconfiguration using genetic algorithm based on the Matroid theory," *IEEE Trans. Power Syst.*, vol. 23, no. 1, pp. 186–195, Feb. 2008.

[109] R. D. Zimmerman and H.-D. Chiang, "Fast decoupled power flow for unbalanced radial distribution systems," *IEEE Trans. Power Syst.*, vol. 10, no. 4, pp. 2045–2052, Nov. 1995.

[110] W.-M. Lin and J.-H. Teng, "Three-phase distribution network fast-decoupled power flow solutions," *Int. J. Electr. Power Energy Syst.*, vol. 22, no. 5, pp. 375–380, Jun. 2000.

[111] P. Aravindhababu and R. Ashokkumar, "A fast decoupled power flow for distribution systems," *Electr. Power Compon. Syst.*, vol. 36, no. 9, pp. 932–940, Aug. 2008.

[112] M. R. Irving and M. J. H. Sterling, "Efficient Newton-Raphson algorithm for load-flow calculation in transmission and distribution networks," *IEE Proc. Gener. Transm. Distrib.*, vol. 134, no. 5, pp. 325–330, Sept. 1987.

[113] P. A. N. Garcia, J. L. R. Pereira, S. Carneiro, V. M. da Costa, and N. Martins, "Three-phase power flow calculations using the current injection method," *IEEE Trans. Power Syst.*, vol. 15, no. 2, pp. 508–514, May 2000.

[114] H. Yang, F. Wen, and L. Wang, "Newton-Raphson on power flow algorithm and Broyden method in the distribution system," in *Proc. IEEE Int. Power Energy Conf.*, Johor Bahru, Malaysia, Dec. 2008, pp. 1613–1618.

[115] D. Shirmohammadi, H. W. Hong, A. Semlyen, and G. X. Luo, "A compensation-based power flow method for weakly meshed distribution and transmission networks," *IEEE Trans. Power Syst.*, vol. 3, no. 2, pp. 753–762, May 1988.

[116] G. W. Chang, S. Y. Chu, and H. L. Wang, "An improved backward/forward sweep load flow algorithm for radial distribution systems," *IEEE Trans. Power Syst.*, vol. 22, no. 2, pp. 882–884, May 2007.

[117] T. Alinjak, I. Pavi, and M. Stojkov, "Improvement of backward/forward sweep power flow method by using modified breadth-first search strategy," *IET Gener. Transm. Distrib.*, vol. 11, no. 1, pp. 102–109, Jan. 2017.

[118] A. B. Eltantawy and M. M. A. Salama, "A novel zooming algorithm for distribution load flow analysis for smart grid," *IEEE Trans. Smart Grid*, vol. 5, no. 4, pp. 1704–1711, Jul. 2014.

[119] J.-H. Teng, "A direct approach for distribution system load flow solutions," *IEEE Trans. Power Delivery*, vol. 18, no. 3, pp. 882–887, Jul. 2003.

[120] A. Alsaadi and B. Gholami, "An effective approach for distribution system power flow solution," *Int. J. Electr. Electron. Eng.*, vol. 3, no. 1, pp. 1–5, 2009.

[121] R. Chandra, *Parallel programming in OpenMP*. San Francisco, CA, USA: Morgan kaufmann, 2001.

[122] S. Dehghan, N. Amjady, and A. J. Conejo, "Reliability-constrained robust power system expansion planning," *IEEE Trans. Power Syst.*, vol. 31, no. 3, pp. 2383–2392, May 2016.

[123] D. Oertel and R. Ravi, "Complexity of transmission network expansion planning," *Energy Syst.*, vol. 5, no. 1, pp. 179–207, Mar. 2014.

[124] S. Haffner, A. Monticelli, A. Garcia, J. Mantovani, and R. Romero, "Branch and bound algorithm for transmission system expansion planning using a transportation model," *IEE Proc. - Gener. Transm. Distrib.*, vol. 147, no. 3, pp. 149–156, May 2000.

[125] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Netw.*, vol. 4, Perth, WA, Australia, Nov. 1995, pp. 1942–1948.

[126] B. Fox, "The sobol quasirandom sequence," [Online]. Available:, http://people.sc.fsu.edu/~jburkardt/m_src/sobol/sobol.html.

[127] Y.-X. Jin, H.-Z. Cheng, J.-y. Yan, and L. Zhang, "New discrete method for particle swarm optimization and its application in transmission network expansion planning," *Electr. Power Syst. Res.*, vol. 77, no. 3, pp. 227–233, Mar. 2007.

[128] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM J Comput.*, vol. 1, no. 2, pp. 146–160, Jun. 1972.

[129] R. Fang and D. J. Hill, "A new strategy for transmission expansion in competitive electricity markets," *IEEE Trans. Power Syst.*, vol. 18, no. 1, pp. 374–380, Feb. 2003.

[130] L. Robinson, "Meta-heuristics for electric power transmission network expansion planning problem with security restrictions," Master's thesis, Universidade Estadual Paulista, 2015.

[131] S. Binato, "Optimal expansion of transmission networks by Benders decomposition and cutting planes," Ph.D. dissertation, Federal University of Rio de Janeiro, 2000.

[132] Illinois Institute of Technology, "Ieee 118-bus system data," [Online]. Available:, http://motor.ece.iit.edu/data/IEEE118bus_data_figure.xls.

[133] E. L. D. Silva, H. A. Gil, and J. M. Areiza, "Transmission network expansion planning under an improved genetic algorithm," *IEEE Trans. Power Syst.*, vol. 15, no. 3, pp. 1168–1174, Aug. 2000.

[134] E. L. D. Silva, J. M. A. Ortiz, G. C. D. Oliveira, and S. Binato, "Transmission network expansion planning under a Tabu search approach," *IEEE Trans. Power Syst.*, vol. 16, no. 1, pp. 62–68, Feb. 2001.

[135] C. C. Carøe and R. Schultz, "Dual decomposition in stochastic integer programming," *Oper. Res. Lett.*, vol. 24, no. 1, pp. 37–45, Feb. 1999.

[136] M. Shahidehopour and Y. Fu, "Benders decomposition: applying benders decomposition to power systems," *IEEE Power Energy Mag.*, vol. 3, no. 2, pp. 20–21, Mar. 2005.

[137] E. S. Thorsteinsson, "Branch-and-check: A hybrid framework integrating mixed integer programming and constraint logic programming," in *CP 2001*, ser. LNCS, vol. 2239, Berlin, Germany, 2001, pp. 16–30.

[138] J. F. Cordeau, G. Stojković, F. Soumis, and J. Desrosiers, "Benders decomposition for simultaneous aircraft routing and crew scheduling," *Transp. Sci.*, vol. 35, no. 4, pp. 375–388, Nov. 2001.

[139] F. You and I. E. Grossmann, "Multicut Benders decomposition algorithm for process supply chain planning under uncertainty," *Ann. Oper. Res.*, vol. 210, no. 1, pp. 191–211, Nov. 2013.

[140] F. D. Munoz, B. F. Hobbs, and J.-P. Watson, "New bounding and decomposition approaches for MILP investment problems: Multi-area transmission and generation planning under policy constraints," *Eur. J. Oper. Res.*, vol. 248, no. 3, pp. 888–898, Feb. 2016.

[141] L. Bahiense, G. C. Oliveira, M. Pereira, and S. Granville, "A mixed integer disjunctive model for transmission network expansion," *IEEE Trans. Power Syst.*, vol. 16, no. 3, pp. 560–565, Aug. 2001.

[142] M. Rahmani, R. Romero, and M. J. Rider, "Strategies to reduce the number of variables and the combinatorial search space of the multistage transmission expansion planning problem," *IEEE Trans. Power Syst.*, vol. 28, no. 3, pp. 2164–2173, Aug. 2013.

[143] R. D. Zimmerman, C. E. Murillo-Sanchez, and R. J. Thomas, "MATPOWER: Steady-state operations, planning, and analysis tools for power systems research and education," *IEEE Trans. Power Syst.*, vol. 26, no. 1, pp. 12–19, Feb. 2011.

[144] M. Carrion and J. M. Arroyo, "A computationally efficient mixed-integer linear formulation for the thermal unit commitment problem," *IEEE Trans. Power Syst.*, vol. 21, no. 3, pp. 1371–1378, Aug. 2006.

[145] Y. Fu, Z. Li, and L. Wu, "Modeling and solution of the large-scale security-constrained unit commitment," *IEEE Trans. Power Syst.*, vol. 28, no. 4, pp. 3524–3533, Nov. 2013.

[146] H. Pinto, F. Magnago, S. Brignone, O. Alsac, and B. Stott, "Security constrained unit commitment: network modeling and solution issues," in *Proc. IEEE PSCE*, Atlanta, GA, USA, Oct. 2006, pp. 1759–1766.

[147] Reliability Test System Task Force, "The IEEE reliability test system," *IEEE Trans. Power Syst.*, vol. 14, no. 3, pp. 1010–1020, Aug. 1999.

[148] J. Lavaei and S. H. Low, "Zero duality gap in optimal power flow problem," *IEEE Trans. Power Syst.*, vol. 27, no. 1, pp. 92–107, Feb. 2012.

[149] Y. Wang, Y. Liu, and D. S. Kirschen, "Scenario reduction with submodular optimization," *IEEE Trans. Power Syst.*, vol. 32, no. 3, pp. 2479–2480, May 2017.

[150] E. Mohagheghi, A. Gabash, and P. Li, "A study of uncertain wind power in active-reactive optimal power flow," in *Proc. Power Energy Stud. Summit*, Dortmund, Germany, Jan. 2016, pp. 1–6.

[151] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU computing," *Proc. IEEE*, vol. 96, no. 5, pp. 879–899, May 2008.

[152] R. Vuduc, A. Chandramowlishwaran, J. Choi, M. Guney, and A. Shringarpure, "On the limits of GPU acceleration," in *Proc. USENIX conf. hot topics parallelism*, Berkeley, CA, USA, 2010, pp. 1–6.

[153] G. Geng, Q. Jiang, and Y. Sun, "Parallel transient stability-constrained optimal power flow using GPU as coprocessor," *IEEE Trans. Smart Grid*, vol. 8, no. 3, pp. 1436–1445, May 2017.

[154] N. Bell and M. Garland, "Efficient sparse matrix-vector multiplication on CUDA," Nvidia Corporation, Tech. Rep., 2008.

[155] D. Merrill and M. Garland, "Merge-based sparse matrix-vector multiplication (SpMV) using the CSR storage format," in *Proc. ACM Symp. Princ. Pract. Parallel Program.*, Barcelona, Spain, Mar. 2016, pp. 1–2.

[156] NVIDIA, "CUSOLVER library 8.0," Santa Clara, CA, USA, 2017.

[157] J. R. Gilbert, C. Moler, and R. Schreiber, "Sparse matrices in MATLAB: Design and implementation," *SIAM J. Matrix Anal. Appl.*, vol. 13, no. 1, pp. 333–356, Jan. 1992.

[158] NREL, "Measurement and instrumentation data center (MIDC)," [Online], available: http://www.nrel.gov/midc/.

[159] MathWorks, "Run built-in functions on a GPU," [Online], available: https://www.mathworks.com/help/distcomp/run-built-in-functions-on-a-gpu.html.

[160] NVIDIA, "GPU-accelerated libraries," [Online], available: https://developer.nvidia.com/gpu-accelerated-libraries.

[161] MISO, "MISO reliability assurance," [Online], available: https://www.misoenergy.org/WhatWeDo/Pages/Reliability.aspx.

[162] J. Baranowski and D. J. French, "Operational use of contingency analysis at PJM," in *Proc. IEEE Power Energy Soc. Gen. Meeting,*, San Diego, CA, USA, Jul. 2012, pp. 1–4.

[163] NVIDIA, "CUSPARSE library 8.0," Santa Clara, CA, USA, 2017.

[164] Wikipedia, "Liebig's law of the minimum," [Online], available: https://en.wikipedia.org/wiki/Liebig%27s_law_of_the_minimum.

[165] W. F. Tinney and C. E. Hart, "Power flow solution by Newton's method," *IEEE Trans. Power Appar. Syst.*, vol. PAS-86, no. 11, pp. 1449–1460, Nov. 1967.

[166] B. Stott and O. Alsac, "Fast decoupled load flow," *IEEE Trans. Power Appar. Syst.*, vol. PAS-93, no. 3, pp. 859–869, May 1974.

[167] V. Roberge, "Distribution feeder reconfiguration (DFR) test cases," [Online]. Available:, http://roberge.segfaults.net/joomla/index.php/dfr.

[168] J. H. Teng, "Modelling distributed generations in three-phase distribution load flow," *IET Genera. Transm. Distrib.*, vol. 2, no. 3, pp. 330–340, May 2008.

[169] J. M. Cano, M. R. R. Mojumdar, J. G. Norniella, and G. A. Orcajo, "Phase shifting transformer model for direct approach power flow studies," *Int. J. Electr. Power Energy Syst.*, vol. 91, pp. 71–79, Oct. 2017.

[170] M. A. Woodbury, "Inverting modified matrices," *Memorandum Rep.*, vol. 42, no. 106, p. 336, 1950.

A

Derivation of the Compensation Method Implementation Steps

The process can be described as generating $\Delta\boldsymbol{\theta}_*$ from,

$$\Delta\boldsymbol{P}_*/\boldsymbol{V}_* = \boldsymbol{B}'_*\Delta\boldsymbol{\theta}_*, \tag{A.1}$$

where $\boldsymbol{B}'_* = \boldsymbol{B}' + \boldsymbol{M}'_*\delta\boldsymbol{b}'_*\boldsymbol{M}'^T_*$ and $\boldsymbol{B}' = \boldsymbol{L}'\boldsymbol{U}'$.

Based on $\boldsymbol{B}'_* = \boldsymbol{B}' + \boldsymbol{M}'_*\delta\boldsymbol{b}'_*\boldsymbol{M}'^T_*$ and the inverse matrix modification lemma (given in Appendix B), the inverse of $\boldsymbol{B}'_*$ can be obtained as,

$$\boldsymbol{B}'^{-1}_* = \boldsymbol{B}'^{-1} - \boldsymbol{B}'^{-1}\boldsymbol{M}'_*\boldsymbol{c}_*\boldsymbol{M}'^T_*\boldsymbol{B}'^{-1}, \tag{A.2}$$

$$\boldsymbol{c}'_* = \left[\boldsymbol{I} + \delta\boldsymbol{b}'_*\boldsymbol{M}'^T_*(\boldsymbol{U}'^{-1}(\boldsymbol{L}'^{-1}\boldsymbol{M}'_*))\right]^{-1}\delta\boldsymbol{b}'_*. \tag{A.3}$$

Therefore, based on (A.1) and (A.2), we have,

$$\begin{aligned}
\Delta\boldsymbol{\theta}_* &= \boldsymbol{B}'^{-1}_*\left(\Delta\boldsymbol{P}_*/\boldsymbol{V}_*\right) && \text{(A.4)}\\
&= \left(\boldsymbol{B}'^{-1} - \boldsymbol{B}'^{-1}\boldsymbol{M}'_*\boldsymbol{c}_*\boldsymbol{M}'^T_*\boldsymbol{B}'^{-1}\right)\left(\Delta\boldsymbol{P}_*/\boldsymbol{V}_*\right) && \text{(A.5)}\\
&= \left(\boldsymbol{U}'^{-1}\boldsymbol{L}'^{-1}\left(\Delta\boldsymbol{P}_*/\boldsymbol{V}_*\right) - \boldsymbol{U}'^{-1}\boldsymbol{L}'^{-1}\boldsymbol{M}'_*\boldsymbol{c}_*\boldsymbol{M}'^T_*\boldsymbol{U}'^{-1}\boldsymbol{L}'^{-1}\left(\Delta\boldsymbol{P}_*/\boldsymbol{V}_*\right)\right) && \text{(A.6)}\\
&= \boldsymbol{U}'^{-1}\left(\Delta\boldsymbol{\theta}_1 - \boldsymbol{L}'^{-1}\boldsymbol{M}'_*\boldsymbol{c}_*\boldsymbol{M}'^T_*\boldsymbol{U}'^{-1}\Delta\boldsymbol{\theta}_1\right) && \text{(A.7)}\\
&= \boldsymbol{U}'^{-1}\Delta\boldsymbol{\theta}_2. && \text{(A.8)}
\end{aligned}$$

The transformation from (A.6) to (A.7) – (A.8) is based on the definition of (3.21) – (3.22). Finally, the identity between (A.7) – (A.8) and (3.22) – (3.23) concludes the derivation process. ■

B

Proof of Inverse Matrix Modification Lemma

This lemma is utilized to derive (A.2). We start from $\boldsymbol{B}'^{-1}$,

$$\boldsymbol{B}'^{-1} = \left(\boldsymbol{B}' + \boldsymbol{M}'_* \delta \boldsymbol{b}'_* \boldsymbol{M}'^T_*\right)^{-1} \left(\boldsymbol{B}' + \boldsymbol{M}'_* \delta \boldsymbol{b}'_* \boldsymbol{M}'^T_*\right) \boldsymbol{B}'^{-1} \tag{B.1}$$

$$= \left(\boldsymbol{B}' + \boldsymbol{M}'_* \delta \boldsymbol{b}'_* \boldsymbol{M}'^T_*\right)^{-1} \left(\boldsymbol{I} + \boldsymbol{M}'_* \delta \boldsymbol{b}'_* \boldsymbol{M}'^T_* \boldsymbol{B}'^{-1}\right) \tag{B.2}$$

$$= \left(\boldsymbol{B}' + \boldsymbol{M}'_* \delta \boldsymbol{b}'_* \boldsymbol{M}'^T_*\right)^{-1} + \left(\boldsymbol{B}' + \boldsymbol{M}'_* \delta \boldsymbol{b}'_* \boldsymbol{M}'^T_*\right)^{-1} \boldsymbol{M}'_* \delta \boldsymbol{b}'_* \boldsymbol{M}'^T_* \boldsymbol{B}'^{-1} \tag{B.3}$$

$$= \left(\boldsymbol{B}' + \boldsymbol{M}'_* \delta \boldsymbol{b}'_* \boldsymbol{M}'^T_*\right)^{-1} + \boldsymbol{B}'^{-1} \boldsymbol{M}'_* \left(\delta \boldsymbol{b}'^{-1}_* + \boldsymbol{M}'^T_* \boldsymbol{B}'^{-1} \boldsymbol{M}'_*\right)^{-1} \boldsymbol{M}'^T_* \boldsymbol{B}'^{-1}. \tag{B.4}$$

Since $\boldsymbol{B}'_* = \boldsymbol{B}' + \boldsymbol{M}'_* \delta \boldsymbol{b}'_* \boldsymbol{M}'^T_*$, we have,

$$\boldsymbol{B}'^{-1}_* = \boldsymbol{B}'^{-1} - \boldsymbol{B}'^{-1} \boldsymbol{M}'_* \boldsymbol{c}'_* \boldsymbol{M}'^T_* \boldsymbol{B}'^{-1}, \tag{B.5}$$

where

$$\boldsymbol{c}'_* = \left(\delta \boldsymbol{b}'^{-1}_* + \boldsymbol{M}'^T_* \boldsymbol{B}'^{-1} \boldsymbol{M}'_*\right)^{-1} \tag{B.6}$$

$$= \left(\boldsymbol{I} + \delta \boldsymbol{b}'_* \boldsymbol{M}'^T_* \boldsymbol{B}'^{-1} \boldsymbol{M}'_*\right)^{-1} \delta \boldsymbol{b}'_* \tag{B.7}$$

$$= \left[\boldsymbol{I} + \delta \boldsymbol{b}'_* \boldsymbol{M}'^T_* (\boldsymbol{U}'^{-1} (\boldsymbol{L}'^{-1} \boldsymbol{M}'_*))\right]^{-1} \delta \boldsymbol{b}'_*. \tag{B.8}$$

Thus, equations (B.5) and (B.8) are identical with (A.2) and (A.3). ■