PowerModelsADA: A Framework for Solving Optimal Power Flow using Distributed Algorithms

Mohannad Alkhraijah, Rachel Harris, Carleton Coffrin, and Daniel K. Molzahn

***Abstract*—This letter presents PowerModelsADA, an open-source framework for solving Optimal Power Flow (OPF) problems using Alternating Distributed Algorithms (ADA). PowerModelsADA provides a framework to test, verify, and benchmark both existing and new ADAs. This letter demonstrates use cases for PowerModelsADA and validates its implementation with multiple OPF formulations.**

***Index Terms*—Distributed Optimization, Optimal Power Flow.**

I. Introduction

ALTERNATING Distributed Algorithms (ADA) decompose large optimization problems into smaller subproblems to share calculations among multiple computing agents [1], [2]. ADAs have potential advantages in scalability, reliability, and communication requirements. However, there is no standard framework for benchmarking ADAs, leading to implementation, comparison, and replicability challenges that are compounded by different formulations, data structures, and communication requirements among ADAs.

To address these challenges, we present an open-source Julia package called PowerModelsADA (**Power Models A**lternating **D**istributed **A**lgorithms)[1] that provides a framework for solving Optimal Power Flow (OPF) problems using ADAs. PowerModelsADA uses the power system optimization package PowerModels [3] and the optimization modeling package JuMP [4] to solve the subproblems. PowerModels separates the OPF formulation from the optimization model and JuMP separates the optimization model from the solver. PowerModelsADA adds a layer to these tools that permits selecting among multiple ADAs.

As shown in Fig. 1, PowerModelsADA enables the "plug-and-play" selection among several ADAs, initializations, power flow models, and solvers. With standardized data, communication, and computational structures, PowerModelsADA currently implements four ADAs: Alternating Direction Method of Multipliers (ADMM) [5], Analytical Target Cascading (ATC) [6], Auxiliary Problem Principle (APP) [7], and Augmented Lagrangian Alternating Direction Inexact Newton (ALADIN) [8].

This letter demonstrates the implementation and use cases of PowerModelsADA. Section II provides the OPF problem formulation and overviews the ADAs. Section III explains the architecture of PowerModelsADA. Section IV illustrates PowerModelsADA's benchmarking capabilities. Section V presents conclusions and future directions.

[1]https://github.com/mkhraijah/PowerModelsADA.jl

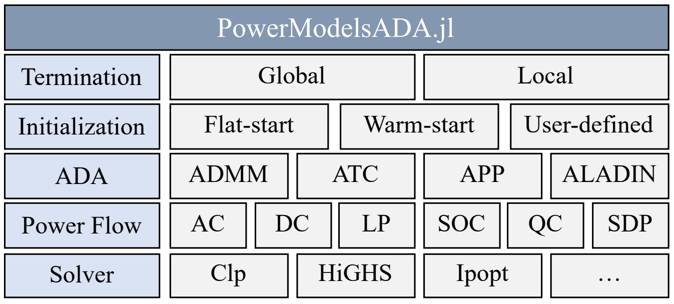


Fig. 1. Illustration of PowerModelsADA options of termination methods, initializations, ADAs, power flow models, and solvers.

II. Optimal Power Flow

A. Problem Formulation

The OPF problem minimizes an objective, typically generation cost, subject to power flow and engineering constraints:

$$\underset{\text{subject to:}}{\text{minimize}}\ f_G := \sum_{g\in\mathcal{G}} c_{g_2}\Re(S_g^G)^2 + c_{g_1}\Re(S_g^G) + c_{g_0} \tag{1a}$$

$$\sum_{g\in\mathcal{G}_i} S_g^G - \sum_{l\in\mathcal{L}_i} S_l^L - \sum_{s\in\mathcal{S}_i} Y_s^{sh}|V_i|^2 = \sum_{(i,j)\in\mathcal{E}} S_{ij}, \quad \forall i\in\mathcal{N}, \tag{1b}$$

$$S_{ij} = Y_{ij}^* V_i V_i^* - Y_{ij}^* V_i V_j^*, \quad \forall (i,j)\in\mathcal{E}, \tag{1c}$$

$$V_i^{min} \le |V_i| \le V_i^{max}, \quad \forall i\in\mathcal{N}, \tag{1d}$$

$$S_g^{min} \le S_g^G \le S_g^{max}, \quad \forall g\in\mathcal{G}, \tag{1e}$$

$$|S_{ij}| \le S_{ij}^{max}, \quad \forall (i,j)\in\mathcal{E}, \tag{1f}$$

$$\angle V_r = 0, \tag{1g}$$

where $\mathcal{N}$, $\mathcal{G}$, $\mathcal{L}$, $\mathcal{E}$, and $\mathcal{S}$ are the sets of buses, generators, loads, branches, and shunts. The subsets $\mathcal{G}_i \subset \mathcal{G}$, $\mathcal{L}_i \subset \mathcal{L}$, and $\mathcal{S}_i \subset \mathcal{S}$ are the corresponding elements at bus $i \in \mathcal{N}$. The decision variables are the buses' voltage phasors $V_i \in \mathbb{C}$, $\forall i \in \mathcal{N}$, the generators' power outputs $S_g^G \in \mathbb{C}$, $\forall g \in \mathcal{G}$, and the branches' power flows $S_{ij} \in \mathbb{C}$, $\forall (i,j) \in \mathcal{E}$. The load demands are $S_l^L \in \mathbb{C}$, $\forall l \in \mathcal{L}$, branch admittances are $Y_{ij} \in \mathbb{C}$, $\forall (i,j) \in \mathcal{E}$, and shunt admittances are $Y_s^{sh} \in \mathbb{C}$, $\forall s \in \mathcal{S}$. Each generator $g \in \mathcal{G}$ has a quadratic cost function with coefficients c_{g_2}, c_{g_1}, and c_{g_0}. We use $\Re(\,\cdot\,)$, $|\,\cdot\,|$, $\angle(\,\cdot\,)$, and $(\,\cdot\,)^*$ to denote the real part, magnitude, phase angle, and conjugate of complex variables. Complex-valued inequalities are interpreted element-wise on the real and imaginary parts.

Objective (1a) minimizes total generation cost. The equalities (1b)–(1c) model AC power flow. The inequalities (1d)–(1f) bound voltage magnitudes, generator outputs, and apparent power flows. Constraint (1g) sets the reference angle.

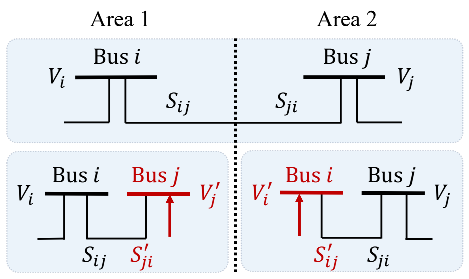


Fig. 2. Two connected areas separated by the dashed line before (top) and after (bottom) decomposition.

The formulation in (1) is commonly called the ACOPF problem since it uses the AC power flow equations. The ACOPF (1) is non-convex and generally NP-hard [9]. Nonetheless, local non-linear solvers can often find good solutions. There are also many OPF approximations and relaxations that are more tractable than (1) [10]. PowerModelsADA builds on PowerModels's flexibility to select the OPF formulation among the polar (ACP) and rectangular (ACR) forms, various approximations (e.g., DC power flow), and relaxations (e.g., second-order cone (SOC) and quadratic convex (QC)) [3].

B. Alternating Distributed Algorithms

ADAs solve large problems by iteratively solving smaller subproblems associated with each area $a \in \mathcal{A}$. There are several decomposition strategies with differing implications for the number of variables and the convergence rate [11]. PowerModelsADA decomposes the OPF problem by introducing fictitious buses and generators, with zero-cost unbounded active and reactive power outputs, at branch terminals connecting two areas as shown in Fig. 2. We then impose consistency constraints between the fictitious and the original variables:

$$V_i = V_i', \qquad \forall i \in \mathcal{N}_B, \tag{2a}$$

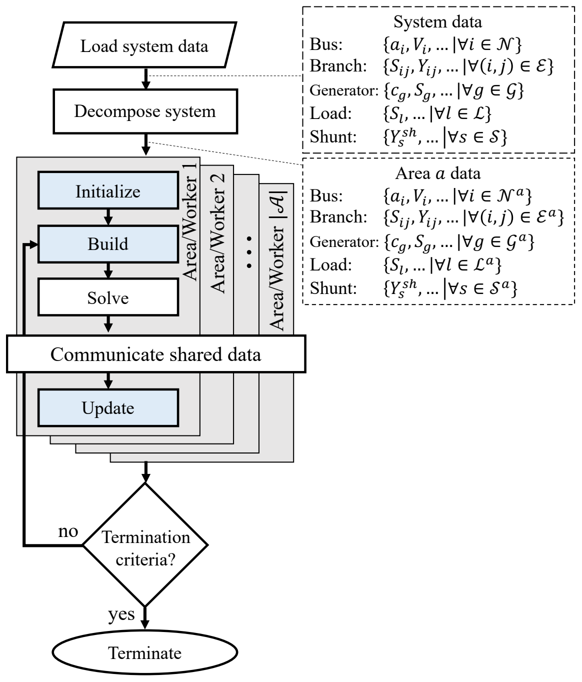
$$S_{ij} = S_{ij}', \qquad \forall (i,j) \in \mathcal{L}_B, \tag{2b}$$

where the sets $\mathcal{N}_B$ and $\mathcal{L}_B$ are the boundary buses and branches. To separate the subproblems, we relax the consistency constraints (2) using an augmented Lagrangian method and evaluate the fictitious variables with values shared by neighbors. The ADAs then 1) solve the subproblems, 2) share the boundary variable values with neighboring areas, 3) update the relaxed consistency constraints with the shared boundary variables, and 4) repeat this process until achieving consensus.

III. POWERMODELSADA ARCHITECTURE

PowerModelsADA solves distributed OPF problems via the function `solve_dopf`. This function takes the system data, algorithm-specific functions and parameters, the power flow model, and the optimization solver as inputs, and returns the optimal solution of each area. The initial release of PowerModelsADA implements four ADAs: ADMM, ATC, APP, and ALADIN.

Fig. 3 shows the PowerModelsADA algorithmic flowchart. The first two blocks denote the loading and decomposition of system data into multiple areas. The next blocks comprise the agents' local computation, communication, and termination.



Load system data
Decompose system
Initialize
Build
Solve
Communicate shared data
Update
Area/Worker 1
Area/Worker 2
Area/Worker $|\mathcal{A}|$
Termination criteria?
no
yes
Terminate
System data
Bus: $\{a_i, V_i, \ldots | \forall i \in \mathcal{N}\}$
Branch: $\{S_{ij}, Y_{ij}, \ldots | \forall (i,j) \in \mathcal{E}\}$
Generator: $\{c_g, S_g, \ldots | \forall g \in \mathcal{G}\}$
Load: $\{S_l, \ldots | \forall l \in \mathcal{L}\}$
Shunt: $\{Y_s^{sh}, \ldots | \forall s \in \mathcal{S}\}$
Area a data
Bus: $\{a_i, V_i, \ldots | \forall i \in \mathcal{N}^a\}$
Branch: $\{S_{ij}, Y_{ij}, \ldots | \forall (i,j) \in \mathcal{E}^a\}$
Generator: $\{c_g, S_g, \ldots | \forall g \in \mathcal{G}^a\}$
Load: $\{S_l, \ldots | \forall l \in \mathcal{L}^a\}$
Shunt: $\{Y_s^{sh}, \ldots | \forall s \in \mathcal{S}^a\}$

Fig. 3. PowerModelsADA implementation flowchart. The boxes on the right show the data passed to the next block. The stacked Area/Worker blocks run in parallel. The light-blue blocks indicate algorithm-specific blocks.

The light-blue blocks in Fig. 3 are algorithm-specific blocks, while the white blocks are common to all ADAs.

Although the algorithmic flow in Fig. 3 is used by many ADAs, PowerModelsADA also implements another algorithmic flow with a central coordinator. Further, the framework can be extended to consider other algorithmic flows (e.g., multiple hierarchical levels) by defining new solve functions. Next, we explain the PowerModelsADA data loading, local computation, communication, and termination criteria.

A. Data Loading and Decomposition

PowerModelsADA inherits PowerModels' ability to load data in MATPOWER [12] and PTI formats, and includes a decomposition function to separate the system data into multiple areas as described in Section II-B. PowerModelsADA uses dictionaries of key-value pairs for data structures, both internally and to interface with PowerModels. The inputs and outputs of the next blocks use the same data structure.

B. Local Computation and Communication

The agents receive the area data structure, perform the local computations in parallel using the Distributed library in Julia, and synchronize by communicating their results until achieving the termination criteria. Local computations consist of four functions: initialize, build, solve, and update for the local subproblem.

Initialize is an algorithm-specific function that defines the ADA's shared variables, iteration counter, and status (mismatches and termination criteria) as well as the shared variables' initial values via a flat-start, a warm-start, or a user-defined method. In each iteration, the agents *build* their local subproblems by defining the variables, constraints, and objective. Most ADAs have the same variables and constraints, while the objective is algorithm-specific. The agents then *solve* their subproblem using PowerModels and store the results in the area data structure. Each agent then exchanges shared data with the neighboring areas in the communication block and stores the received data. Then, the agents *update* their area data structure for the next iteration.

C. Termination

The algorithm terminates after reaching a maximum number of iterations or achieving shared variable mismatches below a specified tolerance, measured via either an ℓ_2- or ℓ_∞-norm. Upon termination, the agents store the results in the area data structure. PowerModelsADA can check the termination criteria using either central or distributed methods. The central method uses a global variable to store the termination status. The distributed method requires additional iterations to allow agents to communicate the termination status with other agents and terminate the local computations simultaneously.

IV. Test Case and Benchmarking

To demonstrate the capabilities of PowerModelsADA, we solved OPF problems using the ADMM algorithm with five power flow formulations. We used the 588-bus system with 8 areas from PGLib-OPF [13] (for other ADAs results and additional test cases, see PowerModelsADA repository[2]). We ran the test case and varied the number of parallel computations from 1 to 10 processors. The results here used PowerModelsADA v0.1 in Julia v1.8 with the Ipopt solver.

An ADAs performance depends on the choice of hyper parameters that can be challenging to tune. We tuned the parameters of the ADMM algorithm by starting with a large value (10^6) and then reducing the parameter gradually (dividing by 10) until we found a good setting. We reported the results that achieved an objective function value within 1% of the solution from PowerModels. We calculated the computation time by taking the average time of five runs.

The computation time of solving the OPF problem with five power flow formulations while varying the number of processors is shown in Fig. 4. As we increase the number of processors, the computation time decreases until reaching 8 processors. In some cases, the computation time increases with more processors due to the uneven assignment of the subproblems to the processors. The computation time reduces by a factor of 4 when using 8 processors compared to a single processor. Increasing the number of processors beyond 8 does not reduce the computation time because the test case has 8 areas/workers such that the additional processors are not used.

V. Conclusion and Future Work

PowerModelsADA provides a plug-and-play modular framework to test and benchmark ADAs for solving OPF problems. The framework makes it straightforward to implement newly developed ADAs by defining three blocks of code (initialization, building, and updating functions). Users can then solve the OPF problem using ADAs with multiple

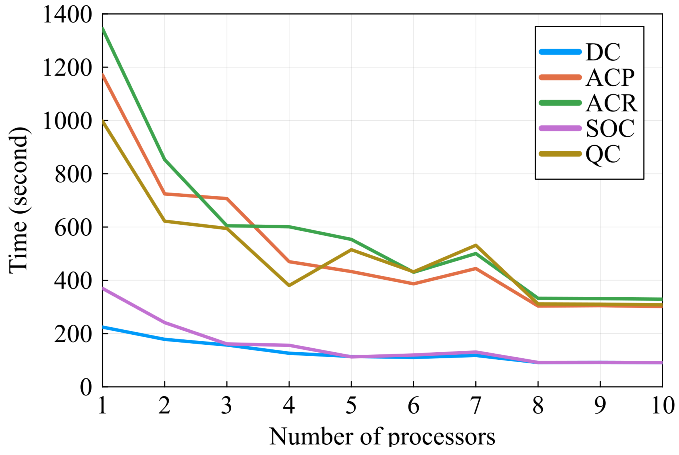


Fig. 4. Computation time of solving OPF using five power flow formulations with various number of processors.

power flow formulations and optimization solvers. Facilitating the implementation of ADAs provides many advantages to the research community for development and benchmarking.

We are pursuing several directions for extending PowerModelsADA. We intend to consider other power system optimization problems besides OPF and complete the functionality that is in PowerModels. Other possible extensions include incorporating additional decomposition and initialization methods as well as adding more ADAs.

References

[1] D. K. Molzahn *et al.*, "A survey of distributed optimization and control algorithms for electric power systems," *IEEE Trans. Smart Grid*, vol. 8, no. 6, pp. 2941–2962, 2017.

[2] M. Alkhraijah, C. Menendez, and D. K. Molzahn, "Assessing the impacts of nonideal communications on distributed optimal power flow algorithms," *Electric Power Syst. Res.*, vol. 212, p. 108297, 2022, presented at *22nd Power Syst. Comput. Conf. (PSCC 2022)*.

[3] C. Coffrin, R. Bent, K. Sundar, Y. Ng, and M. Lubin, "PowerModels.jl: An open-source framework for exploring power flow formulations," in *20th Power Syst. Comput. Conf. (PSCC)*, 2018.

[4] I. Dunning, J. Huchette, and M. Lubin, "JuMP: A modeling language for mathematical optimization," *SIAM Rev.*, vol. 59, no. 2, 2017.

[5] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, July 2010.

[6] A. Kargarian, Y. Fu, and Z. Li, "Distributed security-constrained unit commitment for large-scale power systems," *IEEE Trans. Power Syst.*, vol. 30, no. 4, pp. 1925–1936, 2015.

[7] B. Kim and R. Baldick, "Coarse-grained distributed optimal power flow," *IEEE Trans. Power Syst.*, vol. 12, no. 2, pp. 932–939, 5 1997.

[8] A. Engelmann, Y. Jiang, T. Muhlpfordt, B. Houska, and T. Faulwasser, "Toward distributed OPF using ALADIN," *IEEE Trans. Power Syst.*, vol. 34, no. 1, pp. 584–594, 1 2019.

[9] D. Bienstock and A. Verma, "Strong NP-hardness of AC power flows feasibility," *Oper. Res. Lett.*, vol. 47, no. 6, pp. 494–501, 2019.

[10] D. K. Molzahn and I. A. Hiskens, "A survey of relaxations and approximations of the power flow equations," *Found. Trends Electric Energy Syst.*, vol. 4, no. 1-2, pp. 1–221, 2019.

[11] R. Harris, M. Alkhraijah, D. Huggins, and D. K. Molzahn, "On the impacts of different consistency constraint formulations for distributed optimal power flow," in *IEEE Texas Power Energy Conf. (TPEC)*, 2022.

[12] R. D. Zimmerman, C. E. Murillo-Sánchez, 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, 2011.

[13] S. Babaeinejadsarookolaee *et al.*, "The power grid library for benchmarking AC optimal power flow algorithms," *arXiv:1908.02788*, 2019.

[2]https://mkhraijah.github.io/PowerModelsADA.jl/stable/comparison/