

MÁSTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

Distributed System-Level Control for Multi-Terminal DC Distribution Networks

Study on the Communication Failures Effects

Autor: Miguel Berbegal Almohalla

Director: Asimenia Korompili

Madrid

Junio de 2019

AUTHORIZATION FOR DIGITALIZATION, STORAGE AND DISSEMINATION IN THE NETWORK OF END-OF-DEGREE PROJECTS, MASTER PROJECTS, DISSERTATIONS OR BACHILLERATO REPORTS

1. Declaration of authorship and accreditation thereof.

The author Mr. /Ms. Miguel Bertegal Almohalla

HEREBY DECLARES that he/she owns the intellectual property rights regarding the piece of work: Distributed System-Level Control for Multi-Terminal DC Distribution Networks that this is an original piece of work, and that he/she holds the status of author, in the sense granted by the Intellectual Property Law.

2. Subject matter and purpose of this assignment.

With the aim of disseminating the aforementioned piece of work as widely as possible using the University's Institutional Repository the author hereby **GRANTS** Comillas Pontifical University, on a royalty-free and non-exclusive basis, for the maximum legal term and with universal scope, the digitization, archiving, reproduction, distribution and public communication rights, including the right to make it electronically available, as described in the Intellectual Property Law. Transformation rights are assigned solely for the purposes described in a) of the following section.

3. Transfer and access terms

Without prejudice to the ownership of the work, which remains with its author, the transfer of rights covered by this license enables:

a) Transform it in order to adapt it to any technology suitable for sharing it online, as well as including metadata to register the piece of work and include "watermarks" or any other security or protection system.

b) Reproduce it in any digital medium in order to be included on an electronic database, including the right to reproduce and store the work on servers for the purposes of guaranteeing its security, maintaining it and preserving its format.

c) Communicate it, by default, by means of an institutional open archive, which has open and cost-free online access.

d) Any other way of access (restricted, embargoed, closed) shall be explicitly requested and requires that good cause be demonstrated.

e) Assign these pieces of work a Creative Commons license by default.

f) Assign these pieces of work a HANDLE (*persistent* URL). by default.

4. Copyright.

The author, as the owner of a piece of work, has the right to:

a) Have his/her name clearly identified by the University as the author

b) Communicate and publish the work in the version assigned and in other subsequent versions using any medium.

c) Request that the work be withdrawn from the repository for just cause.

d) Receive reliable communication of any claims third parties may make in relation to the work and, in particular, any claims relating to its intellectual property rights.

5. Duties of the author.

The author agrees to:

a) Guarantee that the commitment undertaken by means of this official document does not infringe any third party rights, regardless of whether they relate to industrial or intellectual property or any other type.

b) Guarantee that the content of the work does not infringe any third party honor, privacy or image rights.
c) Take responsibility for all claims and liability, including compensation for any damages, which may be brought against the University by third parties who believe that their rights and interests have been infringed by the assignment.
d) Take responsibility in the event that the institutions are found guilty of a rights infringement regarding the work subject to assignment.

6. Institutional Repository purposes and functioning.

The work shall be made available to the users so that they may use it in a fair and respectful way with regards to the copyright, according to the allowances given in the relevant legislation, and for study or research purposes, or any other legal use. With this aim in mind, the University undertakes the following duties and reserves the following powers:

a) The University shall inform the archive users of the permitted uses; however, it shall not guarantee or take any responsibility for any other subsequent ways the work may be used by users, which are non-compliant with the legislation in force. Any subsequent use, beyond private copying, shall require the source to be cited and authorship to be recognized, as well as the guarantee not to use it to gain commercial profit or carry out any derivative works.
b) The University shall not review the content of the works, which shall at all times fall under the exclusive responsibility of the author and it shall not be obligated to take part in lawsuits on behalf of the author in the event of any infringement of intellectual property rights deriving from storing and archiving the works. The author hereby waives any claim against the University due to any way the users may use the works that is not in keeping with the legislation in force.
c) The University shall adopt the necessary measures to safeguard the work in the future.
d) The University reserves the right to withdraw the work, after notifying the author, in sufficiently justified cases, or in the event of third party claims.

Madrid, on 2nd of July, 2019

HEREBY ACCEPTS

Signed.....

Reasons for requesting the restricted, closed or embargoed access to the work in the Institution's Repository

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Distributed System-Level Control for Multi Terminal DC Distribution Networks

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2018/2019. es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Miguel Berbegal Almohalla Fecha: 26 / 06 / 19

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Fecha: 1 / 7 / 2019



MÁSTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

Distributed System-Level Control for Multi-Terminal DC Distribution Networks

Study on the Communication Failures Effects

Autor: Miguel Berbegal Almohalla

Director: Asimenia Korompili

Madrid

Junio de 2019

CONTROL DE SISTEMA DISTRIBUIDO PARA REDES DE DISTRIBUCIÓN DE CORRIENTE CONTINUA MULTI-TERMINAL

Autor: Berbegal Almohalla, Miguel.

Director: Korompili, Asimenia.

Entidad Colaboradora: E.ON. Energy Research Center-RWTH.

RESUMEN DEL PROYECTO

INTRODUCCIÓN

Planteamiento del problema

En la red de distribución del futuro veremos una gran cantidad de fuentes de energía distribuidas interconectadas con convertidor, tales como fuentes de energía renovables y sistemas de almacenamiento de energía por batería. Estas tecnologías operan en muchos casos de manera más eficiente en corriente continua. También veremos un aumento en el número de un nuevo tipo de participante llamado prosumidor, que es consumidor y productor de electricidad al mismo tiempo. Todo esto significa que las redes de distribución tendrán que pasar de ser redes pasivas a ser redes activas, donde se coordine la generación distribuida. Esto plantea una serie de retos de control específicos para las redes de distribución de corriente continua del futuro. En primer lugar, las fuentes de energía renovables son muy volátiles y tienen dinámicas muy rápidas. En segundo lugar, la cantidad de nodos en las redes de distribución es muy superior al de las redes de transmisión. Por todo lo anterior, se necesita una estrategia de control que sea capaz de dar soluciones rápidas y fiables, capaces de lidiar con la rápida dinámica de las energías renovables, tener buenas propiedades de escalabilidad para poder manejar la gran cantidad de nodos en las redes de distribución, ser modular para aceptar la expansión e inclusión de nuevos generadores distribuidos con facilidad, y permitir la privacidad de datos, lo cual será un tema de gran importancia en el paradigma del prosumidor. En los sistemas de CC, los puntos de ajuste de tensión deben calcularse resolviendo las ecuaciones de flujo de potencia. Si se desea una coordinación a nivel de sistema de los generadores distribuidos, entonces se debe resolver el flujo de cargas óptimo. Teniendo en cuenta los requisitos de control enumerados anteriormente, la estrategia de control más adecuada es una estrategia de control distribuido. Por esta razón, el principal foco de esta tesis será el flujo de potencia óptimo distribuido (DOPF).

Estado de la técnica

Los artículos sobre técnicas de descomposición (cómo dividir un gran problema de optimización en subproblemas más pequeños) son muy comunes[1], [2]. Los algoritmos DOPF también se estudian ampliamente en multitud publicaciones[3]-[6]. Sin embargo, la mayoría de los estudios aplican el algoritmo DOPF a estimación de estado[7]-[9] y muy pocos lo aplican al control a nivel sistema para redes de CC. Otro tema importante cuando se trata de algoritmos distribuidos es la comunicación. La comunicación desempeña un papel clave en el rendimiento de los algoritmos distribuidos,

ya que la velocidad y fiabilidad global del algoritmo dependerá en gran medida de la velocidad y fiabilidad de la comunicación. Aunque se pueden encontrar ciertos artículos que abordan los retos y requisitos de comunicación en los sistemas de control en red[10]-[12], no se han encontrado artículos sobre cómo los fallos de comunicación afectan a la velocidad y convergencia de los algoritmos DOPF y qué modificaciones se pueden hacer para que los algoritmos sean más robustos frente a ellos.

Objetivos

El objetivo de esta tesis es cubrir el vacío detectado en la literatura estudiando el efecto de los fallos de comunicación en los algoritmos DOPF aplicados al control a nivel sistema en redes de distribución de CC multiterminal, y diseñar una estrategia de detección y mitigación de fallos de comunicación con el fin de minimizar el impacto de las fallos en el rendimiento general del algoritmo.

METODOLOGÍA

Para alcanzar los objetivos descritos anteriormente, se desarrolló una herramienta de simulación capaz de probar un algoritmo DOPF en condiciones de fallos de comunicación. El algoritmo DOPF utilizado es un "Proximal Message Passing Alternating Direction Method of Multipliers based Distributed Optimal Power Flow with sequential convex approximations"[13]. La ventaja de este algoritmo sobre los muchos otros posibles algoritmos que podrían haberse elegido es que éste puede implementarse de forma totalmente distribuida y en cualquier topología de red dada, a diferencia de otros que requieren coordinadores centrales y sólo son válidos para ciertos tipos de topologías. Aunque se podría haber utilizado una implementación secuencial del algoritmo para estudiar los efectos de los fallos de comunicación, se eligió la implementación completamente distribuida por medio de programación paralela, ya que daría una idea adicional de los desafíos que conlleva una implementación más realista con comunicación real. Como un algoritmo secuencial es un paso previo para lograr el paralelo, se llevó a cabo una comparación entre las dos implementaciones sin incluir ningún fallo de comunicación como un estudio adicional, arrojando algunos resultados interesantes con respecto a los tiempos de comunicación vs. los tiempos de cálculo y la escalabilidad global del algoritmo.

El algoritmo y la herramienta de simulación fueron programados usando Python[14], que es un lenguaje ampliamente difundido que puede ejecutarse en muchas máquinas y sistemas operativos diferentes y tiene un gran soporte comunitario y una amplia gama de paquetes útiles. Tres paquetes fueron de particular importancia para este proyecto: NumPy[15] fue clave para la gestión eficiente de las operaciones vectoriales, CVXOPT[16] se utilizó para resolver los subproblemas de optimización locales y mpi4py[17] para la paralelización del algoritmo utilizando el estándar MPI[18].

En cuanto al hardware, se utilizaron dos máquinas. Para la creación de prototipos y casos de prueba más pequeños se utilizó un MacBook Pro Retina de principios de 2015, mientras que para simular los casos más grandes y tener un entorno más adecuado para programas paralelos se utilizó un Cluster de Computación de Alto Rendimiento en el Centro de Investigación Energética E.ON. de la Universidad RWTH de Aquisgrán.

Para realizar las pruebas, se extrajeron seis redes de prueba estándar IEEE de la base de datos MATPOWER[19,20], cuyo tamaño oscila entre 5 y 300 nodos.

En esta tesis la descomposición del OPF se realiza a nivel de nodo eléctrico. Esto significa que el problema se divide en tantos subproblemas como nodos eléctricos tenga la red. Cada nodo tendrá asignada una "entidad", que es un concepto similar al de "agente" en los sistemas multiagente y puede ser visto como una unidad de control distribuido. Las entidades son la combinación de hardware y software, es decir, los ordenadores y programas que se ejecutan en ellos. Cada entidad se encargará de resolver un subproblema local y comunicarse con otras entidades para resolver de forma colaborativa el OPF de la red.

Los tipos de fallos de comunicación que se consideran en esta tesis son dos: fallos de entidades y fallos de enlaces. Los fallos de enlace se producen cuando una ruta de comunicación punto a punto deja de funcionar, es decir, cuando dos entidades que deberían estar comunicándose ya no pueden comunicarse. Este tipo de fallo es muy poco probable en un entorno de comunicación inalámbrica, pero tendría relevancia en un escenario de comunicación PLC. Los fallos de las entidades pueden ser causados ya sea por un fallo del ordenador o por el hecho de que el programa que se ejecuta en él se bloquee. De cualquier manera, la entidad afectada no puede calcular, recibir o establecer nuevos puntos de ajuste para su convertidor.

Con el fin de estudiar el impacto de los fallos de comunicación y la efectividad de la estrategia de detección y mitigación de fallos de comunicación desarrollada en esta tesis, se ejecutaron más de 80 escenarios diferentes utilizando la herramienta de simulación. Los escenarios comprenden diferentes redes, diferentes estados previos del sistema, diferente cantidad de fallos de enlaces y entidades, en diferentes momentos de la ejecución, con diferentes propiedades de los enlaces o entidades que fallaron, etc., en un intento de explorar todos los diferentes factores que podrían influir en el rendimiento del algoritmo y las limitaciones de la estrategia de detección y mitigación.

RESULTADOS

Cada simulación da como resultado una gran cantidad de datos, que se representan en gráficos como los que se muestran en las siguientes figuras:

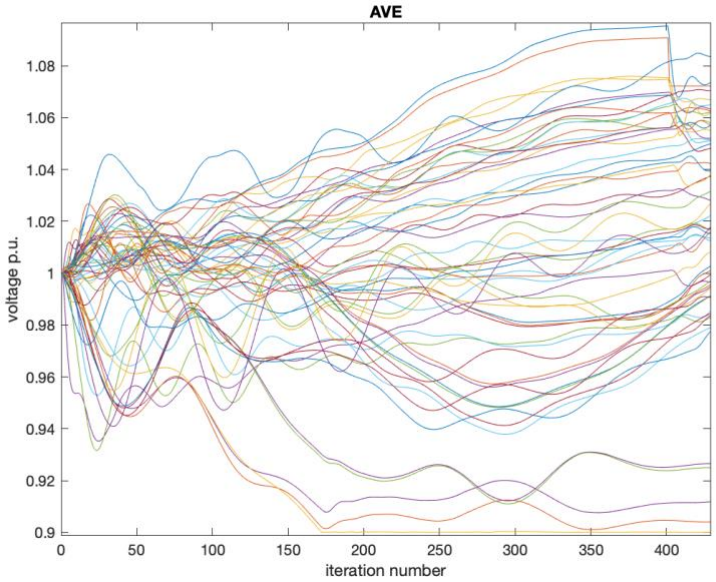


Fig. 1: Evolución de los voltajes nodales con respecto a las iteraciones del algoritmo.

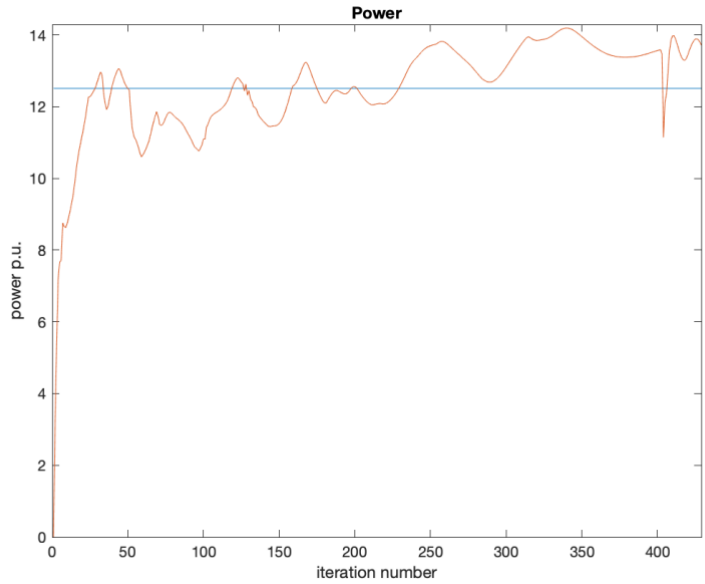


Fig. 2: Evolución de la potencia total generada en la red vs. la potencia total demandada respecto a las iteraciones del algoritmo.

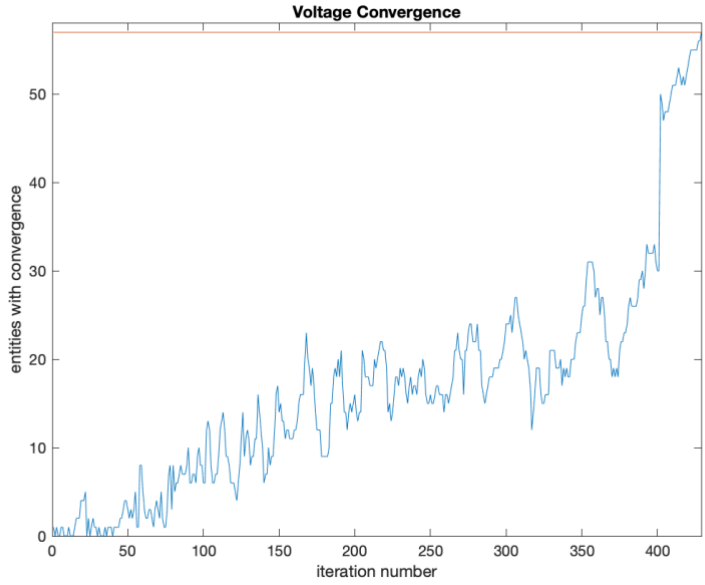


Fig. 3: Número de entidades que alcanzan convergencia en los voltajes y número total de entidades en el sistema respecto a las iteraciones del algoritmo.

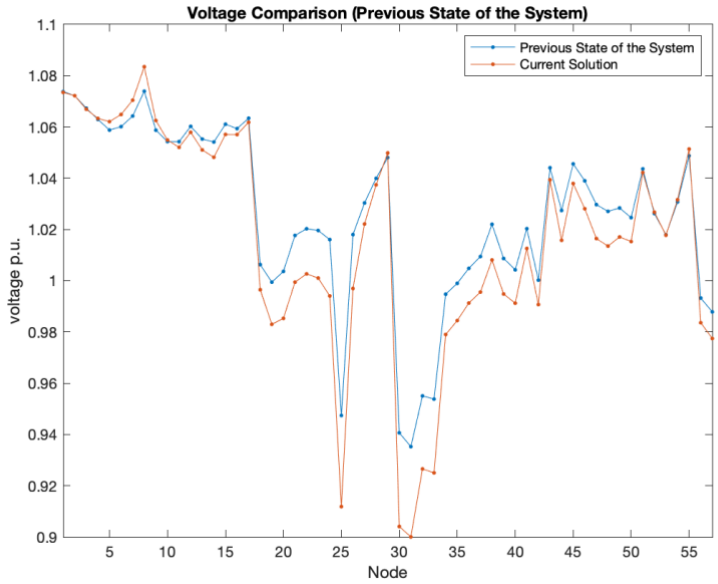


Fig. 4: Perfil final de voltajes del estado anterior del sistema vs. el estado actual del sistema.

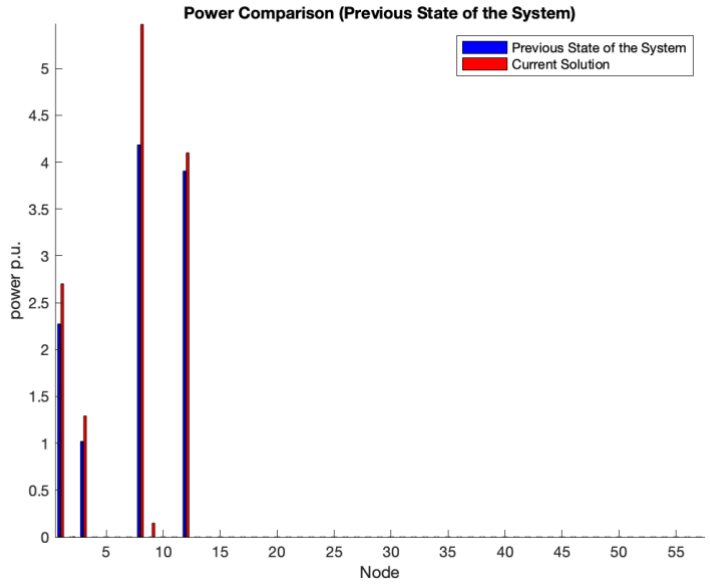


Fig. 5: Comparación de potencia generada por nodo del estado anterior del sistema vs. la solución actual.

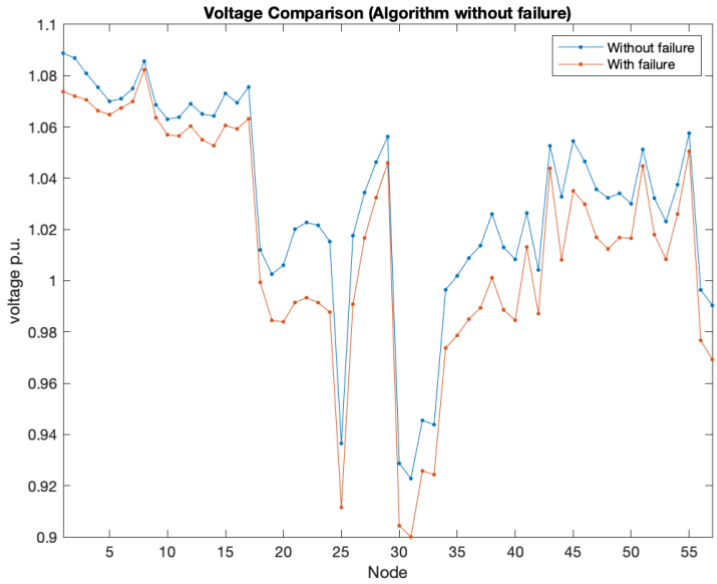


Fig. 6: Perfil final de voltajes del caso con y sin fallo de comunicación.

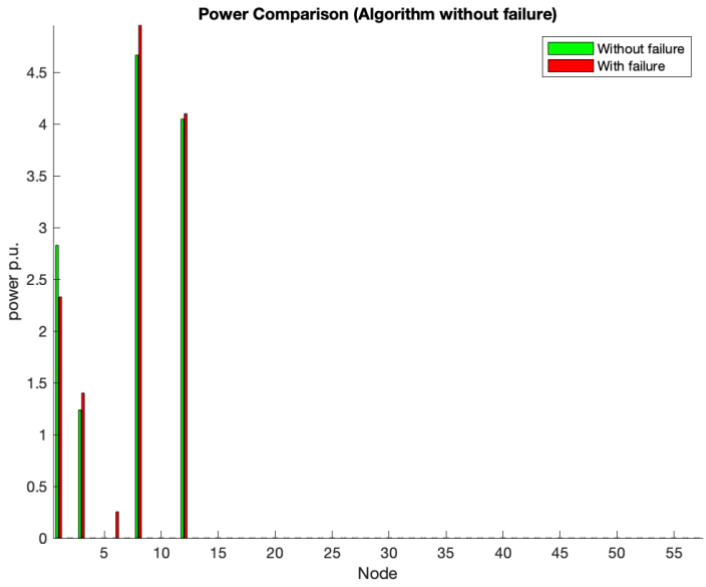


Fig. 7: Comparación de potencia generada por nodo del caso con y sin fallo de comunicación.

La información principal se resume en tablas como la 1:

Single entity failure resilts - I					
30 node network	**Reference**	**Biggest generator**		**Smallest generator**	
Iteration of failure	-	100	1200	100	1200
Entity failed	-	1		5	
Neighbors	-	2,3(2)		2,7(2)	
Generation(p.u.)	-	3,602		1	
Voltage convergence	yes	yes	yes	yes	yes
Power convergence	yes	yes	yes	yes	yes
Total power	2,97	3,11	3,109	3,109	3,117
Total demand		2,83			
Power losses dev. (p.u.)	-	0,14	0,139	0,139	0,147
Max iterations		3000			
Iterations till conv.	1379	197	1223	193	1229
Iterations after failure	-	97	23	93	29

Tabla 1: Ejemplo de tabla de resultados.

Esta tabla muestra, a modo de ejemplo, algunos casos de fallo en los que sólo una entidad falla, en una red de 30 nodos. La columna de referencia muestra los resultados del caso sin ningún fallo de comunicación. Las otras dos columnas muestran dos casos, en los que la entidad que tiene el nodo con el generador más grande y el más pequeño es la que falla. Para cada caso se consideran dos subcasos en los que la iteración del algoritmo en el que se produce el fallo se encuentra al principio o al final de la ejecución del algoritmo en condiciones normales. Se da información sobre qué entidad es la que está fallando, cuántos vecinos eléctricos tiene su nodo y cuál es el tamaño del generador en ese nodo. Luego se presentan los resultados con respecto a la convergencia, la potencia generada, las iteraciones totales necesarias hasta la convergencia y el número de iteraciones necesarias después del fallo.

Por razones prácticas, las tablas correspondientes a los más de 80 casos de simulación no se pueden presentar en este resumen, pero los principales resultados de estas pruebas son:

- El tamaño del generador de la entidad que falla no tiene ninguna influencia particular en la convergencia del algoritmo.
- Cuando la carga del estado anterior del sistema es mayor que la carga del actual, la solución del estado previo del sistema en términos de generación ya es suficiente para cubrir la demanda actual, mientras que cuando la carga del estado anterior del sistema es inferior a la actual, entonces el algoritmo tiene que hacer más esfuerzo para cumplir el criterio de potencia (potencia generada mayor o igual que potencia demandada). Por lo tanto, el algoritmo es más resistente a los fallos y tiene mejor convergencia cuando la carga está en una tendencia decreciente en comparación con cuando la carga está en una tendencia creciente.
- El número de vecinos de la entidad que falla no muestra ninguna influencia particular en nuestros resultados. Sólo en casos de aumento extremo de la carga desde el estado anterior del sistema hasta el actual, el número de vecinos se vuelve

relevante, donde cuantos menos vecinos tenga la entidad fallida, mejor será la convergencia.

- El tiempo del fallo tampoco tiene en general influencia en la convergencia del algoritmo en fallos de una única entidad. Sólo en los casos de fallo de múltiples entidades, si los fallos se producen al principio del algoritmo, es más probable que el algoritmo converja que si se produce al final.
- En los fallos de múltiples entidades, los fallos dispersos son más críticos que los fallos localizados.
- Las redes más grandes pueden tolerar más fallos, pero la cantidad de fallos que cada red puede tolerar es específica de la red y de las condiciones del sistema.
- Las redes con más generación distribuida tienen mejor convergencia y son más resistentes a los fallos.
- Gracias a la estrategia de detección y mitigación de fallos de comunicación, el algoritmo es capaz de producir soluciones rápidas y factibles en una amplia variedad de escenarios de fallo y redes.

En cuanto al estudio adicional realizado durante el desarrollo del algoritmo, se analizó la escalabilidad del algoritmo. Los algoritmos secuencial y paralelo fueron cronometrados bajo las mismas condiciones, ejecutando las 6 redes de prueba, y comparados. Los resultados demuestran que el algoritmo es computacionalmente escalable, pero está limitado en comunicación. Esto significa que el tiempo de comunicación es mucho mayor que el tiempo de cálculo. En nuestra implementación, el algoritmo secuencial, que calcula todos los subproblemas locales uno tras otro pero que carece de comunicación, es en realidad más rápido y escalable que la implementación paralela, donde los subproblemas se resuelven en paralelo pero se necesita una comunicación real. El problema reside en que la comunicación en el clúster donde se ejecutó el algoritmo paralelo es heterogénea en tecnologías de comunicación y en velocidad de las mismas, y, que además, tienden a crear cuellos de botella en casos de mucha paralelización. Sin embargo, con una tecnología de comunicación consistente, homogénea y rápida como la 5G, se espera que el algoritmo paralelo sea escalable también desde el punto de vista de las comunicaciones y produzca tiempos de ejecución totales por debajo de la marca de un minuto.

CONCLUSIONES

- La estrategia de detección y mitigación de fallos de comunicación funciona como se espera y es capaz de dar soluciones rápidas y factibles en una amplia variedad de escenarios de fallos de comunicación.
- Se han identificado factores que influyen en la convergencia y velocidad del algoritmo bajo fallos de comunicación.
- El problema está limitado en comunicación y la tecnología de comunicación es crítica para la velocidad final y la escalabilidad del algoritmo.
- La herramienta de simulación es capaz de probar la convergencia del algoritmo aplicado a diferentes redes bajo cualquier combinación posible de fallos de comunicación.

REFERENCIAS

[1] S. Boyd and L. Vandenberghe, Convex Optimization. New York, NY, USA: Cambridge University Press, 2004, isbn: 0521833787.

[2] S. Boyd, “Notes on decomposition methods”, Notes for EE364B, pp. 1–36, Jan. 2007.

[3] D. K. Molzahn, F. Dörfler, H. Sandberg, S. H. Low, S. Chakrabarti, R. Baldick, and J. Lavaei, “A survey of distributed optimization and control algorithms for electric power systems”, IEEE Transactions on Smart Grid, vol. 8, no. 6, pp. 2941–2962, 2017, issn: 1949-3053. doi: 10.1109/TSG.2017.2720471.

[4] B. H. Kim and R. Baldick, “A comparison of distributed optimal power flow algorithms”, IEEE Transactions on Power Systems, vol. 15, no. 2, pp. 599–604, 2000, issn: 0885-8950. doi: 10.1109/59.867147.

[5] W. Lu, M. Liu, S. Lin, and L. Li, “Fully decentralized optimal power flow of multi-area interconnected power systems based on distributed interior point method”, IEEE Transactions on Power Systems, vol. 33, no. 1, pp. 901–910, 2018, issn: 0885-8950. doi: 10.1109/TPWRS.2017.2694860.

[6] T. Erseghe, “Distributed optimal power flow using admm”, IEEE Transactions on Power Systems, vol. 29, no. 5, pp. 2370–2380, 2014, issn: 0885-8950. doi: 10.1109/TPWRS.2014.2306495.

[7] J. Liang, B. Shen, H. Dong, and J. Lam, “Robust distributed state estimation for sensor networks with multiple stochastic communication delays”, International Journal of Systems Science, vol. 42, no. 9, pp. 1459–1471, 2011. doi: 10.1080/00207721.2010.550402. eprint: https://doi.org/10.1080/00207721.2010.550402. [Online]. Available: https://doi.org/10.1080/00207721.2010.550402.

[8] D. M. Falcao, F. F. Wu, and L. Murphy, “Parallel and distributed state estimation”, IEEE Transactions on Power Systems, vol. 10, no. 2, pp. 724–730, 1995, issn: 0885-8950. doi: 10.1109/59.387909.

[9] S. Kar, J. M. F. Moura, and K. Ramanan, “Distributed parameter estimation in sensor networks: Nonlinear observation models and imperfect communication”, IEEE Transactions on Information Theory, vol. 58, no. 6, pp. 3575–3605, 2012, issn: 0018-9448. doi: 10.1109/TIT.2012.2191450.

[10] J. Hespanha, P. Naghshtabrizi, and Y. Xu, “A survey of recent results in networked control systems”, Proceedings of the IEEE, vol. 95, pp. 138 –162, Feb. 2007. doi: 10.1109/JPROC.2006.887288.

[11] R. A. Gupta and M. Chow, “Networked control system: Overview and research trends”, IEEE Transactions on Industrial Electronics, vol. 57, no. 7, pp. 2527–2535, 2010, issn: 0278-0046. doi: 10.1109/TIE.2009.2035462.

[12] J. Baillieul and P. J. Antsaklis, “Control and communication challenges in networked real-time systems”, Proceedings of the IEEE, vol. 95, no. 1, pp. 9–28, 2007, issn: 0018-9219. doi: 10.1109/JPROC.2006.887290.

[13] S. Magnússon, P. C. Weeraddana, and C. Fischione, “A distributed approach for the optimal power-flow problem based on admm and sequential convex approximations”, IEEE Transactions on Control of Network Systems, vol. 2, no. 3, pp. 238–253, 2015, issn: 2325-5870. doi: 10.1109/TCNS.2015.2399192.

[14] Python, https://www.python.org/, Accessed: 2018-09-23.

[15] Numpy, https://www.numpy.org/, Accessed: 2018-10-10.

[16] Cvxopt: Python software for convex optimization, https : / / cvxopt . org/, Accessed: 2018-10-10.

[17] L. Dalcín, R. Paz, and M. Storti, “Mpi for python”, J. Parallel Distrib. Comput., vol. 65, no. 9, pp. 1108–1115, Sep. 2005, issn: 0743-7315. doi: 10.1016/ j.jpdc.2005.03.010. [Online]. Available: http://dx.doi.org/10.1016/j. jpdc.2005.03.010.

[18] Mpi forum: Documents, https://www.mpi- forum.org/docs/, Accessed: 2018-11-25.

[19] Matpower: Free, open-source electric power system simulation and optimization tools for matlab and octave, http://www.pserc.cornell.edu/matpower/, Accessed: 2018-09-13.

[20] Electric power systems analysis and nature-inspired optimization algorithms, https://www.al-roomi.org/power-flow, Accessed: 2018-09-14.

DISTRIBUTED SYSTEM-LEVEL CONTROL FOR MULTI-TERMINAL DC DISTRIBUTION NETWORKS

INTRODUCTION

Problem statement

In the distribution grid of the future we will see a great amount of converter interfaced distributed energy sources such as renewable energy sources and battery storage systems. These technologies operate in many cases more efficiently in DC. We will also see an increase in this new type of participants called prosumers, which are consumers and producers of electricity at the same time. All this means that the distribution networks will have to transition from passive networks to active ones, where the distributed generators are coordinated. This poses a set of control challenges specific to DC distribution networks. First, renewable energy sources are very volatile and have very fast dynamics. Second, the amount of nodes in distribution networks is far superior to that in transmission networks. Therefore, a control strategy is needed that is able to give fast and reliable solutions in order to keep up with the fast dynamics of renewables, have good scalability properties to be able to handle the huge amount of nodes in distribution networks, be modular to accept expansion and inclusion of new distributed generators with ease, and allow for data privacy, which will be of great importance in the prosumer paradigm. In DC systems, the voltage setpoints have to be calculated by solving the power flow equations. If system-wide coordination of the distributed generators is desired, then the optimal power flow must be solved instead. Taking into account the control requirements enumerated above, the most suitable control strategy is a distributed control strategy. For this reason, the main problem of this thesis will be the distributed optimal power flow (DOPF).

Literature review

Articles on decomposition techniques (how to split a big optimization problem into smaller subproblems) can be easily found in literature[1], [2]. DOPF algorithms are also extensively studied in literature [3]-[6]. However, most of the papers apply the DOPF algorithm to state estimation [7]-[9] and very few apply it to system-level control for DC networks. Another important topic when dealing with distributed algorithms is communication. Communication plays a key role in the performance of distributed algorithms, as the speed and reliability of the overall algorithm will strongly depend on the speed and reliability of the communication. While there are some papers that tackle communication challenges and requirements in networked control systems [10]-[12], no papers were found on the topic of how communication failures affect the speed and convergence of DOPF algorithms and what modifications can be done to make the algorithms more robust against them.

Objectives

The aim of this thesis is to cover the detected gap in literature by studying the effect of communication failures on DOPF algorithms applied to system-level control in multi-terminal DC distribution networks, and designing a communication failure

detection and mitigation strategy in order to minimize the impact of the failures in the overall performance of the algorithm.

METHODOLOGY

In order to achieve the objectives described above, a simulation tool able to test a DOPF algorithm under communication failure conditions was developed. The DOPF algorithm used is a Proximal Message Passing Alternating Direction Method of Multipliers based Distributed Optimal Power Flow with sequential convex approximations[13]. The advantage of this algorithm over the many other possible algorithms that could have been chosen is that this one can be implemented in a fully distributed manner and in any given network topology, unlike others that require central coordinators and are only valid for certain types of topologies. Although a sequential implementation of the algorithm could have been used to study the communication failure effects, the fully distributed implementation by means of parallel programming was chosen, as it would give additional insight in what challenges come with a more realistic implementation with actual communication. As a sequential algorithm is a step to achieve the parallel one, a comparison between the two implementations without including any communication failures was carried out as an additional side-study, throwing some interesting results with regards to communication vs. computation times and overall scalability.

The algorithm and the simulation tool were programmed using Python[14], which is a widely spread language that can run in many different machines and operative systems and has great community support and a wide range of useful packages. Three packages were of particular importance for this project: NumPy[15] was key to managing efficient array operations, CVXOPT[16] was used as a solver for the local optimization sub-problems and mpi4py[17] for the parallelization of the algorithm using the MPI standard[18].

On the hardware side, two machines were used. For the prototyping and smaller test cases a MacBook Pro Retina early 2015 was used, while to simulate the bigger cases and to have an environment more suitable for parallel programs a High-Performance Computing Cluster at the E.ON. Energy Research Center of the RWTH Aachen University was used.

To run the tests, six IEEE standard test networks were extracted from the MATPOWER[19,20] data base, ranging in size from 5 to 300 nodes.

In this thesis the decomposition of the OPF is done at the electrical node level. This means, that the problem is divided in as many subproblems as electrical nodes has the grid. Each node will have an “entity” assigned to it, which is a similar concept to “agent” in Multi-Agent Systems and can be seen as a distributed control unit. Entities are the combination of hardware and software, i.e., the computers and program running on them. Each entity will be in charge of solving the local sub-problem and communicate with other entities in order to solve collaboratively the OPF of the grid.

The types of communication failures considered in this thesis are two: entity failures and link failures. Link failures occur when a point-to-point communication path stops working, i.e., when two entities that should be communicating cannot communicate anymore. This type of failure is very unlikely in a wireless communication environment but would have relevance in a PLC communication scenario. Entity failures can be caused either by a computer failing or the program running on it crashing. In any way, the entity affected cannot calculate, receive or set any new setpoints for their converter.

In order to study the impact of communication failures and the effectiveness of the communication failure detection and mitigation strategy developed in this thesis, more than 80 different scenarios were run using the simulation tool. The scenarios entitled different networks, different previous states of the system, different amount of link and entity failures, at different moments in the execution, with different properties of the links or entities that failed etc., in an attempt to explore all the different factors that could have an influence on the performance of the algorithm and the limitations of the detection and mitigation strategy.

RESULTS

Each simulation of the simulation tool gives as a result a big amount of data, which is then graphically represented in plots such as the ones seen in the following figures:

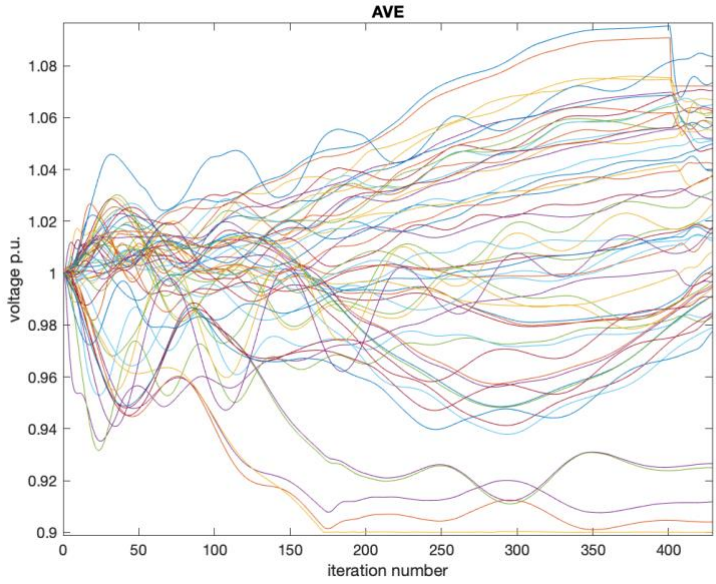


Fig. 1: Evolution of the voltages of all nodes throughout the iterations of the algorithm.

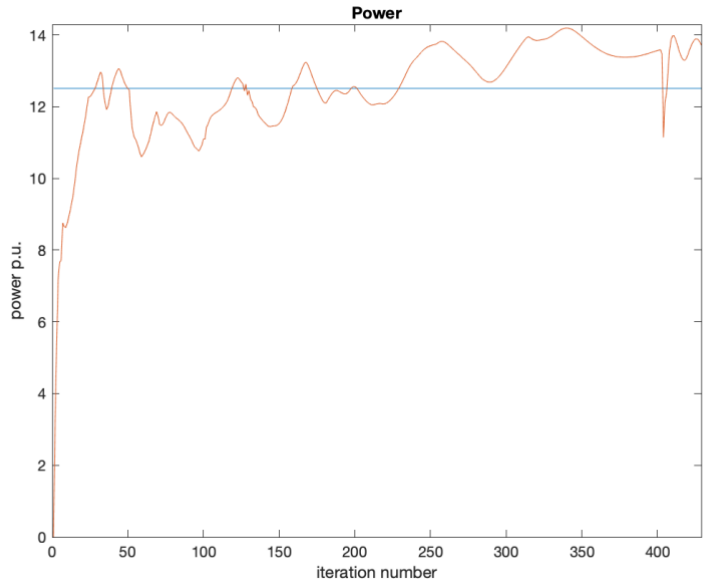


Fig. 2: Evolution of the total power generated in the grid vs. the total power demanded throughout the iterations of the algorithm.

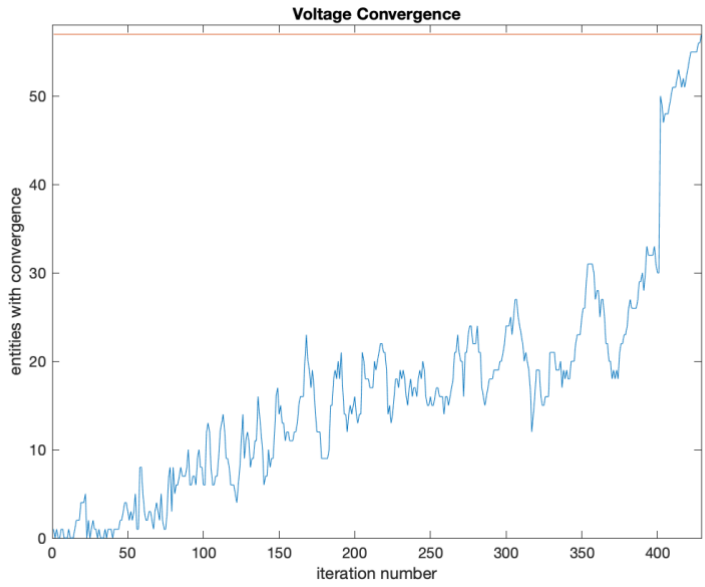


Fig. 3: Number of entities that reach voltage convergence with respect to the total number of entities in the system throughout the iterations of the algorithm.

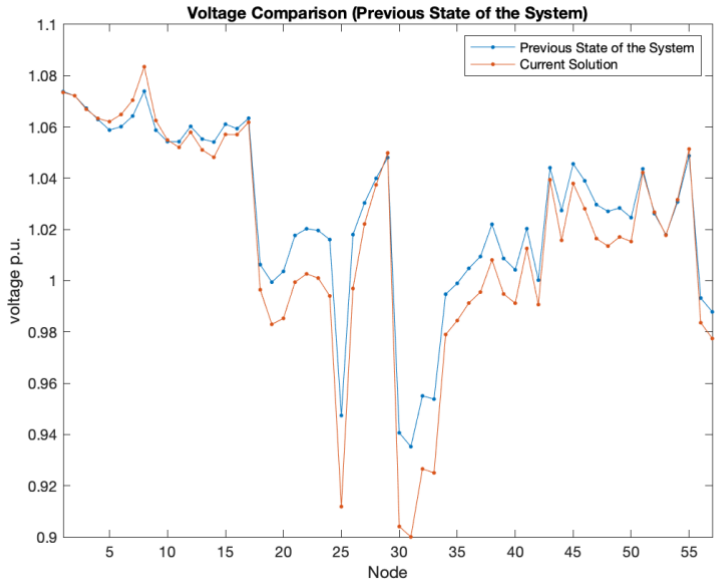


Fig. 4: Final voltage profile of the system of the previous state of the system vs. the current state of the system.

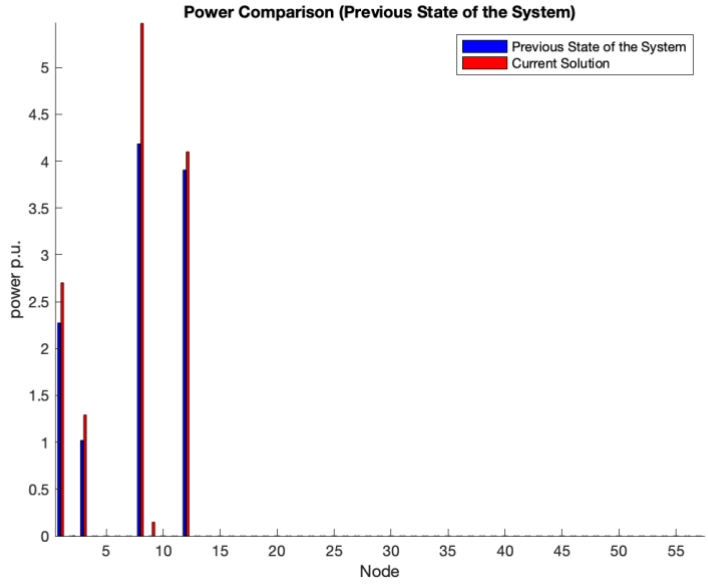


Fig. 5: Power generation per node comparison between the previous state of the system and the current one.

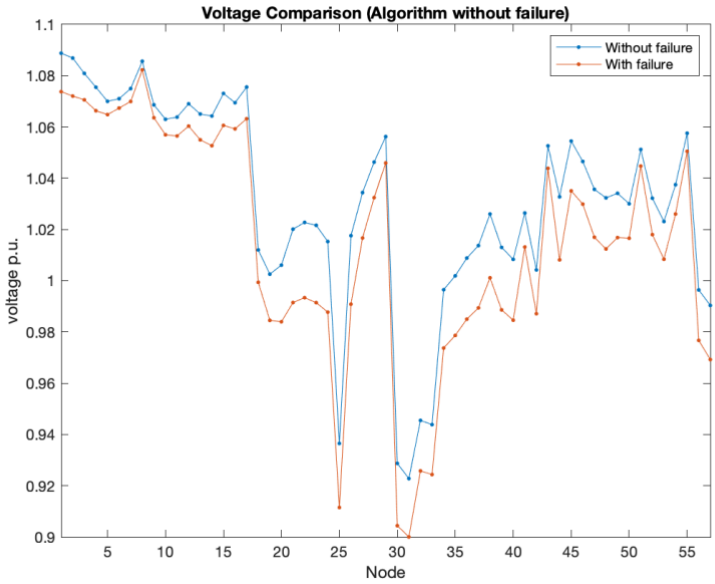


Fig. 6: Final voltage profile of the system of the same case with and without communication failure.

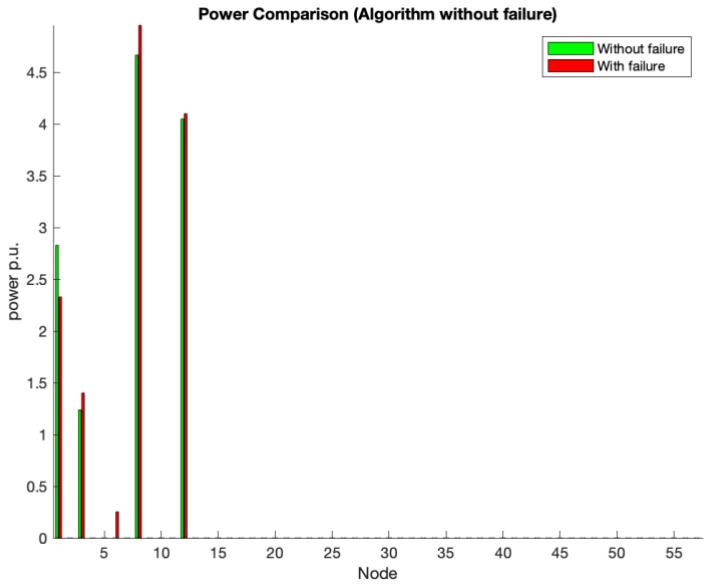


Fig. 7: Power generation per node comparison between the same case with and without failure.

The main information is summed up in tables such as table 1:

Single entity failure resilts - I					
30 node network	**Reference**	**Biggest generator**		**Smallest generator**	
Iteration of failure	-	100	1200	100	1200
Entity failed	-	1		5	
Neighbors	-	2,3(2)		2,7(2)	
Generation(p.u.)	-	3,602		1	
Voltage convergence	yes	yes	yes	yes	yes
Power convergence	yes	yes	yes	yes	yes
Total power	2,97	3,11	3,109	3,109	3,117
Total demand	2,83				
Power losses dev. (p.u.)	-	0,14	0,139	0,139	0,147
Max iterations	3000				
Iterations till conv.	1379	197	1223	193	1229
Iterations after failure	-	97	23	93	29

Table 1: Example of results.

This table shows, as an example, some single entity failure cases, where only one entity fails, in a 30 node network. The reference column shows the results of the case without any communication failures. The other two columns show two cases, in which the entity that has the node with the biggest and smallest generator is the one that fails. For each case two subcases are considered where the iteration of the algorithm at which the failure occurs is either at the beginning or at the end of the execution of the algorithm in normal conditions. Information is given about what entity is the one failing, how many electrical neighbors does it's node have and what is the size of the generator in that node. Then the results are presented with regards to convergence, power generated, total iterations needed until convergence and number of iterations needed after the failure.

Due to practical reasons, the tables corresponding to the more than 80 simulation cases cannot be presented in this summary, but the main results of these tests are:

- The size of the generator of the entity that fails does not have any particular influence on the convergence of the algorithm.
- When the load of the previous state of the system is greater than the load of the current one, the solution of the previous state of the system in terms of generation is already enough to cover the current demand, while when the load of the previous state of the system is lower than the current one, then the algorithm has to make more efforts to meet the power criterion. Therefore, the algorithm is more fault resistant and has better convergence when the load is in a decreasing trend compared to when the load is in an increasing trend.
- The number of neighbors of the entity that fails do not show any particular influence in our results. Only in cases of extreme load increase from the previous state of the system to the current one the number of neighbors becomes relevant, where the less neighbors the failed entity has the better the convergence.
- The time of the failure also has in general no influence in the convergence of the algorithm in single entity failures. Only in the multiple entity failure

cases, if the failures occur at the beginning of the algorithm the algorithm is more likely to converge than if it occurs at the end.

- In multiple entity failures, spread out failures are more critical than localized failures.
- Bigger networks can handle more failures, but the amount of failures that each network can tolerate is specific to the network and conditions of the system.
- Networks with more distributed generators have better convergence and are more failure resistant.
- Thanks to the communication failure detection and mitigation strategy, the algorithm is able to produce fast and feasible solutions in a wide variety of communication failure scenarios and networks.

Regarding the side-study carried out while developing the algorithm, the scalability of the algorithm was analyzed. The sequential and parallel algorithms were timed under the same conditions, running the 6 different test networks, and compared. The results demonstrate that the algorithm is computationally scalable, but communication bound. This means that the communication time is much bigger than the computation time. In our implementation, the sequential algorithm, which computes all the local sub-problems one after the other but lacks communication, is actually faster and more scalable than the parallel implementation, where the sub-problems are solved in parallel but real communication is needed. The problem resides in that the communication in the HPC cluster where the parallel algorithm was executed is heterogeneous in communication technologies and speed and prompted to bottle-necks. However, with a consistent, homogeneous and fast communication technology such as 5G, it is expected that the parallel algorithm becomes scalable also from the communications point of view and yield total execution times below the one-minute mark.

CONCLUSIONS

- The communication failures detection and mitigation strategy works as expected and is able to give fast and feasible solutions in a wide variety of communication failure scenarios.
- Influencing factors in the convergence and speed of the algorithm under communication failures have been identified.
- The problem is communication-bound and the communication technology is critical for the final speed and scalability of the algorithm.
- The simulation tool is able to test the convergence of the algorithm applied to different networks under any possible combination of communication failures.

REFERENCES

[1] S. Boyd and L. Vandenberghe, Convex Optimization. New York, NY, USA: Cambridge University Press, 2004, isbn: 0521833787.

[2] S. Boyd, “Notes on decomposition methods”, Notes for EE364B, pp. 1–36, Jan. 2007.

[3] D. K. Molzahn, F. Dörfler, H. Sandberg, S. H. Low, S. Chakrabarti, R. Baldick, and J. Lavaei, "A survey of distributed optimization and control algorithms for electric power systems", IEEE Transactions on Smart Grid, vol. 8, no. 6, pp. 2941–2962, 2017, issn: 1949-3053. doi: 10.1109/TSG.2017.2720471.

[4] B. H. Kim and R. Baldick, "A comparison of distributed optimal power flow algorithms", IEEE Transactions on Power Systems, vol. 15, no. 2, pp. 599–604, 2000, issn: 0885-8950. doi: 10.1109/59.867147.

[5] W. Lu, M. Liu, S. Lin, and L. Li, "Fully decentralized optimal power flow of multi-area interconnected power systems based on distributed interior point method", IEEE Transactions on Power Systems, vol. 33, no. 1, pp. 901–910, 2018, issn: 0885-8950. doi: 10.1109/TPWRS.2017.2694860.

[6] T. Erseghe, "Distributed optimal power flow using admm", IEEE Transactions on Power Systems, vol. 29, no. 5, pp. 2370–2380, 2014, issn: 0885-8950. doi: 10.1109/TPWRS.2014.2306495.

[7] J. Liang, B. Shen, H. Dong, and J. Lam, "Robust distributed state estimation for sensor networks with multiple stochastic communication delays", International Journal of Systems Science, vol. 42, no. 9, pp. 1459–1471, 2011. doi: 10.1080/00207721.2010.550402. eprint: https://doi.org/10.1080/00207721.2010.550402. [Online]. Available: https://doi.org/10.1080/00207721.2010.550402.

[8] D. M. Falcao, F. F. Wu, and L. Murphy, "Parallel and distributed state estimation", IEEE Transactions on Power Systems, vol. 10, no. 2, pp. 724–730, 1995, issn: 0885-8950. doi: 10.1109/59.387909.

[9] S. Kar, J. M. F. Moura, and K. Ramanan, "Distributed parameter estimation in sensor networks: Nonlinear observation models and imperfect communication", IEEE Transactions on Information Theory, vol. 58, no. 6, pp. 3575–3605, 2012, issn: 0018-9448. doi: 10.1109/TIT.2012.2191450.

[10] J. Hespanha, P. Naghshtabrizi, and Y. Xu, "A survey of recent results in networked control systems", Proceedings of the IEEE, vol. 95, pp. 138 –162, Feb. 2007. doi: 10.1109/JPROC.2006.887288.

[11] R. A. Gupta and M. Chow, "Networked control system: Overview and research trends", IEEE Transactions on Industrial Electronics, vol. 57, no. 7, pp. 2527–2535, 2010, issn: 0278-0046. doi: 10.1109/TIE.2009.2035462.

[12] J. Baillieul and P. J. Antsaklis, "Control and communication challenges in networked real-time systems", Proceedings of the IEEE, vol. 95, no. 1, pp. 9–28, 2007, issn: 0018-9219. doi: 10.1109/JPROC.2006.887290.

[13] S. Magnússon, P. C. Weeraddana, and C. Fischione, "A distributed approach for the optimal power-flow problem based on admm and sequential convex approximations", IEEE Transactions on Control of Network Systems, vol. 2, no. 3, pp. 238–253, 2015, issn: 2325-5870. doi: 10.1109/TCNS.2015.2399192.

[14] Python, https://www.python.org/, Accessed: 2018-09-23.

[15] Numpy, https://www.numpy.org/, Accessed: 2018-10-10.

[16] Cvxopt: Python software for convex optimization, https : / / cvxopt . org/, Accessed: 2018-10-10.

[17] L. Dalcín, R. Paz, and M. Storti, "Mpi for python", J. Parallel Distrib. Comput., vol. 65, no. 9, pp. 1108–1115, Sep. 2005, issn: 0743-7315. doi: 10.1016/ j.jpdc.2005.03.010. [Online]. Available: http://dx.doi.org/10.1016/j. jpdc.2005.03.010.

[18] Mpi forum: Documents, https://www.mpi- forum.org/docs/, Accessed: 2018-11-25.

[19] Matpower: Free, open-source electric power system simulation and optimization tools for matlab and octave, http://www.pserc.cornell.edu/matpower/, Accessed: 2018-09-13.

[20] Electric power systems analysis and nature-inspired optimization algorithms, https://www.al-roomi.org/power-flow, Accessed: 2018-09-14.



MÁSTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

Distributed System-Level Control for Multi-Terminal DC Distribution Networks

Study on the Communication Failures Effects

Autor: Miguel Berbegal Almohalla

Director: Asimenia Korompili

Madrid

Junio de 2019

Contents

1 Introduction **1**
- 1.1 Motivation and Problem Statement . . . 1
- 1.2 Contribution . . . 4
- 1.3 Outline of the Thesis . . . 5

2 Distributed Optimization Techniques **7**
- 2.1 Basic Definitions . . . 7
- 2.2 Convex Programming . . . 8
- 2.3 Lagrange Duality Theory . . . 8
- 2.4 Dual Ascent Method and Dual Decomposition . . . 9
- 2.5 Augmented Lagrangians and the Method of Multipliers . . . 14
- 2.6 Alternating Direction Method of Multipliers . . . 15

3 Distributed ADMM Algorithm for DC Systems **21**
- 3.1 Entity Concept . . . 21
- 3.2 Centralized Formulation of the Problem . . . 21
- 3.3 Introduction of Clone and Net Variables . . . 24
- 3.4 Alternating Direction Method of Multipliers (ADMM) . . . 25
- 3.5 Proximal Message Passing-Based Distributed ADMM . . . 27
- 3.6 Sequential Convex Approximations . . . 32
- 3.7 Stopping Criteria . . . 34
  - 3.7.1 The Necessity of a "Ruler Entity" . . . 34
- 3.8 Full Algorithm . . . 40

4 Communication Failures **43**
- 4.1 Types of Communication Failures . . . 43
- 4.2 Communication Failures Detection and Mitigation Strategy . . . 44

5 Implementation: Software and Hardware **51**
- 5.1 Description of the Software . . . 51
  - 5.1.1 Python . . . 51
    - NumPy Package . . . 51
    - Convex Optimization Solver for Python (CVXOPT) . . . 52
    - MPI Package for Python (mpi4py) . . . 53
  - 5.1.2 Simulation Tool . . . 54
    - Code, File Structure and User Interface . . . 54

5.2 Description of the Hardware . . . 58
5.2.1 SWARM HPC Cluster at the E.ON ERC . . . 58
5.2.2 Personal Computer . . . 60

6 Results and Discussion **61**
6.1 Description of the Test Networks . . . 61
6.1.1 5 Node Network . . . 62
6.1.2 14 Node Network . . . 63
6.1.3 30 Node Network . . . 64
6.1.4 57 Node Network . . . 65
6.1.5 118 Node Network . . . 66
6.1.6 300 Node Network . . . 66
6.2 Sequential vs. Parallel Execution . . . 68
6.3 Metrics . . . 69
6.4 Computational Scalability Results . . . 70
6.5 Communication Scalability Results . . . 73
6.6 Communication Failure Results . . . 81
6.6.1 Influencing Factors . . . 81
6.6.2 Test Cases . . . 82
6.6.3 Simulation Plots . . . 84
6.6.4 Results and Discussion . . . 89
Single Entity Failure . . . 90
Effect of the Loading Condition of the System . . . 93
Multiple Entity Failures . . . 96
Link Failures . . . 98

7 Conclusion and Future Work **99**

A Test Networks Parameters **105**
A.1 5 Node Network . . . 105
A.2 14 Node Network . . . 106
A.3 30 Node Network . . . 108
A.4 57 Node Network . . . 110
A.5 118 Node Network . . . 113
A.6 300 Node Network . . . 119

List of Figures **131**

List of Tables **133**

1 Introduction

1.1 Motivation and Problem Statement

The grid is evolving, and with this evolution, control techniques have to follow. This evolution is particularly noticeable at the distribution level, where solar and wind energy together with energy storage systems (ESS) are proliferating. These technologies operate more efficiently in DC systems, putting an end to the hegemony of AC systems. Additionally, an increasing number of a new type of participants called "prosumers" are emerging. These participants act as consumers and producers at the same time and are scattered around the grid. The old passive distribution network needs to be changed to an active distribution network where all these new distributed energy resources (DER) are managed. In DC systems, the voltage setpoints of the interface converters of the DERs must be derived from the power flow (PF) equations. For this reason, in order to achieve a system-wide coordination of the DERs the optimal power flow (OPF) problem has to be solved. The OPF is a well-known problem in power systems typically solved for scheduling, but this system-level control for multi-terminal DC (MTDC) distribution grids context poses a set of new control challenges. Due to the fast dynamics and uncertainty of renewable energy sources, as well as the vast number of nodes that are found in distribution networks, a fast and reliable real-time OPF algorithm is needed.

Control strategies in power systems can be roughly divided into three categories: centralized, decentralized and distributed [1].

The classical centralized control strategy is the one currently in use in the transmission level. The centralized control strategy relies on the SCADA system to collect all the measurements and data from the whole system, to then process all this information to generate the control signals. This strategy is too slow or even impracticable when applied to distribution networks which, as mentioned earlier, are characterized by fast dynamics and a huge number of nodes when compared to the transmission network. Furthermore, if the central controller fails the system cannot operate anymore, providing a single point of failure and, if there are any changes in the grid, such as new generators, the central controller has to be updated. A faster, more robust and flexible strategy is required.

Decentralized approaches have some advantages that may make them seem suitable for distribution network control. They are based in local controllers (LC) that only require local information to function. There is no communication needed, which makes this strategy very cheap (as no communication infrastructure is needed) and robust (as if one LC fails the rest still operate keeping the system also operative).

The downside of this strategy is that, while it works good with global variables (such as the droop control for power-frequency regulation in AC systems), it is not suitable for the control of local variables (such as voltage control).

The distributed control strategy is in a way similar to the decentralized one, but instead of using local controllers it is based on distributed control entities (DCEs or entities), that can communicate with one another. Entity-based systems gather all the characteristics desirable for the control of MTDC distribution networks:

- Fast computing and communication: The main problems are divided into several subproblems that can be solved in parallel by all entities.
- Robustness: Even if one entity fails the rest of them can make up for it.
- Scalability: DCE based systems are applicable to many different sizes of networks.
- Modularity and Plug-and-play capability: This strategy can adapt itself very easily to changes in the grid, such as the introductions of new generators, which would come along with the introduction of a new entity that will instantly start cooperating with the existing ones.
- Data privacy: Unlike in the centralized strategy where all the information has to be sent to the central controller, the entities rely only on local information plus some additional exchange of information between them, allowing for data privacy.

Implementing a distributed control strategy in MTDC distribution networks implies having to modify the OPF problem. Many different decomposition algorithms achieve this goal (see [2] and [3]). The most frequently used algorithms are those based on the Augmented Lagrangian Decomposition: auxiliary problem principal (APP), alternating direction method of multipliers (ADMM), proximal message passing (PMP), predictor-corrector proximal multiplier method (PCPM) and dual decomposition (DD). Of course, one of the first things that influence the speed in solving the OPF problem is the algorithm itself. In [4], a comparison between different algorithms for the same system is carried out. Out of all the existing algorithms, the ADMM seems to be the most widely spread for distributed OPF (DOPF) applications, mainly due to its flexibility: it can be used in a hierarchical structure, fully distributed structure, or with or without a coordinator. Examples of its application can be seen in [5] [6] [7]. In this thesis, we decided to take the algorithm described in [8] as reference. The main advantages of this algorithm over other ADMM based algorithms are that it takes the form of a fully distributed OPF algorithm (not needing a centralized coordinator, thus allowing for data privacy) and that it is compatible with any given network topology (normally fully distributed ADMM algorithms are limited to radial topologies).

Regarding the different factors that affect the performance and convergence of distributed algorithms (not specific to power systems or DOPF), several studies tackle

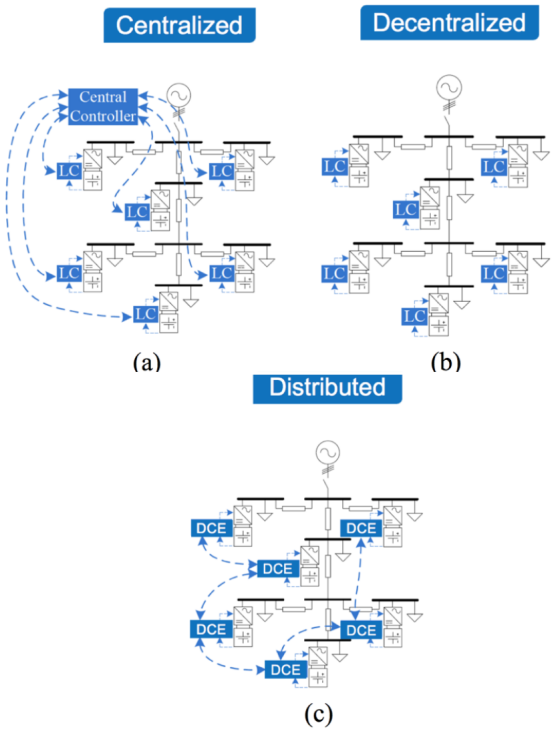


Figure 1.1: Control strategies: (a) centralized (b) decentralized (c) distributed.

the effect of the partitioning of the system, such as [9] and [10]. Others investigate the control structure (with a central coordinator, hierarchical, fully distributed)and the impact on performance ([11], [12]). Also, a big part of literature is dedicated to communication factors such as communication topology, communication synchronicity or communication efficiency. In [13] an algorithm that is able to find the optimal communication topology in order to balance the number of iterations and the communication cost per iteration is developed and tested; [14] investigates the optimal way to choose with which DCE another one should communicate in order to have a fast propagation of the information with the minimum amount of communication. Sometimes, communication topology can lead to problems such as redundancy or information accumulation, hindering the convergence of the algorithm. This can be solved with an algorithm that analyzes the communication topology and breaks any rings in order to have a tree structure as purposed in [15]. The communication requirements may reach very high levels, especially in very big systems, leading to excessive traffic of information. [16] tries to solve this problem by utilizing message clustering, which involves creating "packages" of information that allows for fewer but bigger messages. Asynchronicity is also an important research topic when it comes to distributed algorithms. Not all subproblems may reach a solution at the same time, some may turn out infeasible or there may be a communication delay. Therefore, there is an interest in making the algorithms robust to these issues. Ex-

amples of articles dealing with this problems can be found in references [6], [17], [18] and [19].

There is also some work done in the direction of Networked Control Systems ([20], [21]) and the communication challenges they pose [22]. There are even papers more specific to electrical systems describing the different communication architectures and requirements, and the communication-related problems to them such as communication delays, synchronization, communication reliability and communication security ([23],[24]).

However, no study was found that assesses the concrete influence of communication failures in the performance and convergence of DOPF algorithms specifically, and what modification can be done to them to minimize the impact of these communication failures. DOPF algorithms in literature are typically applied to scheduling or state estimation ([25],[26],[27]), where a communication failure does not pose a critical threat to the stability of the system, and there is enough time to solve issues and run the algorithm several times before the results are needed. Alternatively, failures can be treated as bad or missing data and tackled in a similar way to bad or missing measurements in the traditional centralized state estimation. In contrast to that, when using DOPF algorithms for system-level control, there is a need for speed, robustness and reliability, as voltage and power set-points are calculated in real time every few minutes and the stability of the system depends critically from these set-points. The ability to give fast and feasible solutions to preserve the stability of the system even under communication failures circumstances is of the utmost importance.

The main goal of this thesis is to analyze the communication failures effect on DOPF algorithms for system-level control in MTDC Distribution Systems and to design a failure mitigation strategy for them.

1.2 Contribution

The contributions of this thesis are:

- Investigation of the effect of communication failures on DOPF algorithms with regards to performance and convergence.
- Design of a communication failure detection and mitigation strategy for DOPF algorithms applied to system-level control in multi-terminal DC distribution networks.
- Fully distributed implementation of a PMP ADMM DOPF algorithm thanks to parallel programming inside a simulation tool able to test the convergence of the algorithm under communication failures. The simulation tool is user-friendly, open-source, reusable and expandable for use in future research.
- Hardware implementation of the algorithm, checking scalability and showing the communication challenges present in a real distributed environment.

1.3 Outline of the Thesis

The overall structure of the study takes the form of seven chapters, including this introductory chapter, where the motivation, problem statement, contributions and outline of the thesis are given.

The second chapter of this thesis reviews the mathematical tools used to separate complex optimization problems in smaller, coordinated sub-problems. It starts with general mathematical optimization definitions, then introduces duality and how it can be used to decompose big problems in smaller, coordinated subproblems, to then finish with the ADMM.

Chapter three describes in detail the particular DOPF algorithm used in this study. It guides through the step-by-step transformation of the traditional OPF into a PMP ADMM with sequential convex approximation DOPF.

The fourth chapter introduces the different communication failures contemplated in this thesis and presents the countermeasures programmed into the algorithm to minimize the effect of these failures on convergence and speed so that the algorithm is able to produce quick and valid results even under fault conditions.

Chapter five describes the hardware and software implementation, i.e., the machines where the algorithm was run and tested, and the code language, software packages and communication standards used to implement the algorithm and develop the simulation tool.

The sixth chapter exhibits the results of the simulation. It first presents the test networks and describes the two different modes of execution of the algorithm (sequential and parallel). Then the different metrics used through the rest of the chapter are enumerated and explained. Lastly, the main results of the thesis are displayed and discussed.

The last chapter summarizes the results, findings and conclusions of the thesis, and proposes areas for future research.

2 Distributed Optimization Techniques

In this chapter, the previous fundamental knowledge required for the understanding of this thesis is presented.

2.1 Basic Definitions

Optimization is defined as the selection of the best element from a set of available alternatives. In mathematics, we defined what is "best" by means of an *objective function*(OF), which value we want to minimize or maximize by finding the values of the objective function's variables that yield the lowest or highest *objective function's value*(OFV). The OF variables have to comply with the *constraints*, which mathematically defined the allowable values for each variable and the relationships between them. The set of variables value that fulfills the constraint equations is called the *feasible region*. All values out of the feasible region are *unfeasible solutions* and violate at least one of the constraints. A problem may also be *unfeasible* if no values of the variables manage to meet all the constraints at the same time (fulfilling one constraint means violating another one) or *unbounded* if the feasible region is not closed and the OFV can be improved indefinitely. In some cases, there may be more than one optimal solution that yields the same OFV (*degenerated solution*).

An optimization problem written in standard form is given by:

$$\min_{x} \quad f(x) \tag{2.1a}$$

$$\text{s.t.} \quad g_i(x) \leq 0 \qquad \forall i, \tag{2.1b}$$

$$h_j(x) = 0 \qquad \forall j, \tag{2.1c}$$

where

- $f(x)$ is the objective function (OF).
- x is the optimization variable.
- $g_i(x)$ are the inequality constraints.
- $h_j(x)$ are the equality constraints.

The minimum OFV (the optimum) is usually denoted as p^*, and the value of all the optimization variables at the optimum point as x^*, such as $f(x^*) = p^*$.

2.2 Convex Programming

Optimization problems can be categorized according to several mathematical criteria. According to these criteria and the actual size of the problem (number of variables, number of constraints) complexity varies significantly. Each group of optimization problems requires different methods to solve them. The most common group of optimization problems are *convex optimization problems*, which OF is convex (if minimizing, concave if maximizing) and the constraint set is also convex. Convexity is the mathematical property of any twice differentiable function of several variables which Hessian matrix of second partial derivatives is positive semidefinite. The Hessian matrix H of the function $f(x_1, x_2, ..., x_N)$is defined as:

$$H_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j} \qquad \forall i = j = 1, 2, ..., N \tag{2.2}$$

A matrix is said to be positive semidefinite when all its eigenvalues are non-negative. One way to find out if all eigenvalues of a matrix M $(n \times n)$ are non-negative is by proving:

$$x^T M x \geq 0 \text{ for any non-zero vector } x \in \mathbb{R}^n \tag{2.3}$$

Another way to check if a matrix is positive semidefinite is by calculating all the leading principal minors (determinants of order 1 to n obtained by adding consecutive rows and columns starting from the first element). If all leading principal minors of a matrix are non-negative, the matrix is positive semidefinite.

One of the most important properties of convex problems is that any local optimal point found is also a global optimum. Convex problems also have the advantage of being able to be solved reliably and efficiently even if the solution cannot be found analytically thanks to resolution methods such as the interior-point algorithm that solve convex optimization problems in polynomial time. To the group of convex optimization problems, also known as *convex programming*, belong several problem subclasses such as *linear programming, quadratic programming, second-order cone programming, semidefinite programming, cone programming and graph form programming.*[28]

2.3 Lagrange Duality Theory

According to the Lagrange duality theory, all optimization problems can be viewed from two different perspectives: the *primal problem* and the *dual problem.* If we call the original minimization problem the primal, the dual problem is a maximization problem that maximizes the value of the *Lagrange dual function.* To obtain the Lagrange dual function, first, the *Lagrangian* function associated with the primal problem must be obtained. The Lagrangian is formed by adding to the objective function each of the constraints multiplied by a term called *Lagrange multiplier* or

dual variable. This way, the Lagrangian form of the optimization problem represented in 2.1 is given by:

$$\mathcal{L}(x, \lambda, \nu) = f(x) + \sum_{i=0}^{i=max(i)} \lambda_i \cdot g_i(x) + \sum_{j=0}^{j=max(j)} \nu_j \cdot h_j(x) \tag{2.4}$$

where

- $\mathcal{L}(x, \lambda, \nu)$ is the Lagrangian associated to 2.1.
- λ_i, ν_j are the Lagrange multipliers or dual variables of the inequality constraint i-th and the equality constraint j-th respectively.

The Lagrange dual function is then defined as the minimum value of the Lagrangian function over x:

$$g(\lambda, \nu) = \inf_x(\mathcal{L}(x, \lambda, \nu)) = \inf_x(f(x) + \sum_{i=0}^{i=max(i)} \lambda_i \cdot g_i(x) + \sum_{j=0}^{j=max(j)} \nu_j \cdot h_j(x)) \tag{2.5}$$

The Lagrange dual function has two important properties. First, it is always concave, regardless of if the primal problem is convex or not, which means there is always a global maximum denoted by d^*. Second, it always holds that $g(\lambda, \nu) \leq p^*$ and, therefore, it always holds that $d^* \leq p^*$, i.e., the dual problem sets a lower bound d^* to the optimum of the primal problem. To find d^* the dual problem is defined:

$$\max_{\lambda,\nu} \quad g(\lambda, \nu) \tag{2.6a}$$

$$\text{s.t.} \quad \lambda \geq 0 \tag{2.6b}$$

The difference between the solution of the primal problem p^* and the solution of the dual problem d^* is called *duality gap* and it is always a non-negative value ($p^* - d^* \geq 0$). In case of *zero duality gap*, the solution of the dual problem is equal to the solution of the primal, which means that *strong duality* holds ($p^* = d^*$) and the dual problem can be solved instead of the primal in order to find p^*. The reason why this is relevant is that in many cases, the dual problem is easier to solve. This property would not be useful if we were not able to know before solving both the primal and dual problems if the duality gap is zero. However, there are ways to proof zero duality gap in convex optimization problems a priori, such as the Slater's condition. The Slater condition states that strong duality holds if an *strictly feasible point* exist (or, in other words, an *interior point* exists), i.e., $\exists x \in (D)$ such that $g_i(x) < 0 \quad \forall i$ and $h_j(x) = 0 \quad \forall j$. [28] Further advantages and uses of Lagrange duality theory are presented in the next section.

2.4 Dual Ascent Method and Dual Decomposition

In many cases, optimization problems have certain characteristics that allow them to be separated or decomposed in simpler, smaller subproblems, that can be solved

in parallel or sequentially, and are coordinated by means of a master problem. Decomposition was originally motivated by very big problems that standard techniques were not able to solve. However, there are other reasons to use decomposition methods, one being that they allow for distributed solutions. Distributed algorithms may be faster than their centralized counterparts depending on the problem type and scale and, even if they are slower, a distributed solution may be preferred for data privacy or cost motives.

There are several decomposition methods, but in this thesis, our attention goes to the *dual ascent method*, that, when applied to decompose optimization problem, is known as *dual decomposition method*(DD)[29]. The main idea behind DD is to introduce a few new variables and work with the dual problem, manipulating the dual variables to solve the global problem.

A basic example of a simple unconstrained optimization problem separated in two sub-problems extracted from [30] is presented:

$$\min_{x} \qquad f(x) = f_1(x_1, y) + f_2(x_2, y) \tag{2.7}$$

where the OF $f(x)$ is divided in two functions f_1 and f_2, and the variables x are divided in (x_1, x_2, y), being x_1 and x_2 the *privat variables*, also called *local variables*, of f_1 and f_2 and y the *complicating varibles*, also known as *public variables*, *interface variables* or *boundary variables*. If y were to be constant, the function would be block separable in x_1 and x_2 and the two subproblems could be solve independently from each other; however, varible y "complicates" things by linking the two subproblems. One can rewrite the problem above like:

$$\min_{x} \qquad f(x) = f_1(x_1, y_1) + f_2(x_2, y_2) \tag{2.8a}$$

$$\text{s.t.} \qquad y_1 = y_2 \tag{2.8b}$$

where *local copies* (y_1, y_2) or *clones* of the complicating varibles y are introduced in addition to a *consistency constraint* that guarantees that the clone variables to be equal. Now the OF is completely seprable, leaving the coupling to the new constraint. To form the dual problem, first the Langrangian must be obtained:

$$\mathcal{L}(x_1, y_1, x_2, y_2, \nu) = f_1(x_1, y_1) + f_2(x_2, y_2) + \nu^T y_1 - \nu^T y_2 \tag{2.9}$$

which is separable. The Lagrange dual function is

$$g(\nu) = g_1(\nu) + g_2(\nu) \tag{2.10}$$

where

$$g_1(\nu) = \inf_{x_1, y_1} (f_1(x_1, y_1) + \nu^T y_1) \qquad g_2(\nu) = \inf_{x_2, y_2} (f_2(x_2, y_2) - \nu^T y_2) \tag{2.11}$$

The separated dual functions $g_1(\nu)$ and $g_2(\nu)$ can be evaluated completely independently, potentially in parallel. Both dual functions can be expressed in terms of the conjugate of f_1 and f_2[some ref to convex conjugate]:

$$g_1(\nu) = -f_1^*(0, -\nu) \qquad g_2(\nu) = -f_2^*(0, \nu) \tag{2.12}$$

The resulting dual problem can be expressed as:

$$\max_{\nu} \qquad g_1(\nu) + g_2(\nu) = -f_1^*(0, -\nu) - f_2^*(0, \nu) \tag{2.13}$$

This problem is the *master problem* in dual decomposition. This problem can be easily solved using the subgradient method. The subgradient of $-g_1$ can be evaluated by finding $\overline{x}_1$ and $\overline{y}_1$ that minimize $f_1(x_1, y_1) + \nu^T y_1$ over x_1 and y_1. The subgradient of $-g_1$ at ν is given by $-\overline{y}_1$. In the same fashion, the subgradient of $-g_2$ at ν is given by $\overline{y}_2$. Therefore, the subgradient of $-g$ is given by $\overline{y}_2 - \overline{y}_1$ which is actually the consistency constraint residual.

It can be seen that using the subgradient method yields a very simple dual decomposition algorithm:

iterate:
Solve subproblems (possibly in parallel):

- Find x_1 and y_1 that minimize $f_1(x_1, y_1) + \nu^T y_1$.
- Find x_2 and y_2 that minimize $f_2(x_2, y_2) - \nu^T y_2$.

Update dual variables:

- $\nu := \nu - \alpha_k(y_2 - y_1)$.

α_k is the step size which can be chosen in several ways(see [30]).

In the previous example, DD was applied to a generic unconstrained optimization problem that was separated into two sub-problems. DD can be applied to much more complex problems. To see a step-by-step process on how to apply DD to increasingly complex problems, please refer to [30].

For this thesis, we would like to decompose an optimization problem with equality and inequality constraints in several subproblems, each of them having private variables and coupled to some of the other subproblems by means of complicating (public) variables. This decomposition structure can be better represented by a hypergraph. An example is provided in figure 2.1.

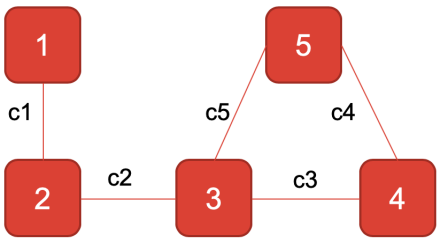


Figure 2.1: Complex decomposition structure.

Each node represents one subproblem. Each subproblem has each own OF f_i, local variables x_i and local constraints. The hyperedges or *nets* represent the cou-

pling between subproblems implicating the complicating variables, which, in DD, are cloned in adjacent subproblems. Each subproblem will have n clone variables, being n the number of "neighbors" each subproblem has. In figure 2.1 there are 5 nodes and 5 nets. As there are five nets, there must be five consistency constraints, one per net, and, as each net only links two subproblems, ten clone variables. For instance, subproblem 1 has only one adjacent node (subproblem 2), and therefore only one clone variable y_1 that must be equal to clone variable y_2 belonging to subproblem 2. Both y_1 and y_2 are clone variables for net $c1$. Subproblem 3 has three neighbours $(2, 4, 5)$, and, consequently, three clone variables (y_4, y_5, y_6) that must be consistent with the respective clone variables of subproblems 2, 4 and 5. The mathematical formulation of the hypergraph is:

$$\begin{aligned}
\min_{x} \quad & f(x) = f_1(x_1, y_1) + f_2(x_2, y_2, y_3) + f_3(x_3, y_4, y_5, y_6) + f_4(x_4, y_7, y_8) \\
& + f_5(x_5, y_9, y_{10}) && \text{(2.14a)} \\
\text{s.t.} \quad & (x_1, y_1) \in \mathcal{C}_1, \qquad (x_2, y_2, y_3) \in \mathcal{C}_2, \qquad (x_3, y_4, y_5, y6) \in \mathcal{C}_3, \\
& (x_4, y_7, y_8) \in \mathcal{C}_4, \qquad (x_5, y_9, y_{10}) \in \mathcal{C}_5, && \text{(2.14b)} \\
& y_1 = y_2 \qquad y_3 = y_4 \qquad y_5 = y_7 \qquad y_6 = y_9 \qquad y_8 = y_{10} && \text{(2.14c)}
\end{aligned}$$

As previously stated, in DD, each subproblem will have a local copy of the complicating variable of the net adjacent to it, as well as a dual variable vector. The subproblems use the dual variables to optimize their local variables, including the local copies of the complicating variables, i.e., the clone variables. The clone variables for each net are then compared and used to update the dual variables in a way that will eventually bring the clone variables to be consistent and, thus, optimality.

If we consider y the vector of clone variables so that $y = (y_1, y_2..., y_{10})$, we will denote $(y)_i$ as the group of elements of vector y that correspond to subproblem i, to distinguish it from each element of vector y, y_i. This way, $(y)_3 = (y_4, y_5, y_6)$.

We can also define z as a vector of size N (being N the number of nets in the hypergraph) that contains the common values of the clone variables on the nets. Vector z is then called the *net variable vector* Then, the consistency constraints can be expressed as $y = Ez$, where E is a matrix that has as many rows as elements have vector y and N columns:

$$E_{i,j} = \begin{cases} 1 & \text{if } y_i \text{ is in edge } N \\ 0 & \text{otherwise.} \end{cases} \tag{2.15}$$

Matrix E is a binary matrix that allocates clone variables to their respective nets. In our example, E is a (10×5) matrix (10 clone variables, 5 nets):

$$E = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.16}$$

Note: if $(y)_3 = (y_4, y_5, y_6)$, $(E)_3$ is the submatrix formed by rows 4,5 and 6.

All this considered, and being N the number of subproblems, we can rewrite problem 2.14 in a more compact way:

$$\min_{x} \quad f(x) = \sum_{i=1}^{N} f_i(x_i, (y)_i) \tag{2.17a}$$

$$\text{s.t.} \quad (x_i, (y)_i) \in \mathcal{C}_i, \qquad \forall i \tag{2.17b}$$

$$(y)_i = (E)_i z, \qquad \forall i \tag{2.17c}$$

Applying DD, first the Lagrangian:

$$\begin{aligned} \mathcal{L}(x, y, z, \nu) &= \sum_{i=1}^{N} f_i(x_i, (y)_i) + \nu^T (y - Ez) \\ &= \sum_{i=1}^{N} (f_i(x_i, (y)_i) + (\nu)_i^T (y)_i) - \nu^T E z, \end{aligned} \tag{2.18a}$$

where ν is the vector of Lagrange multiplier associated with $y = Ez$ and, therefore, has the same dimension as vector y, and $(\nu)_i$ is the subvector of multipliers associated to subproblem i.

If we minimize over z we find the condition $E^T \nu = 0$ for $g(\nu) > -\infty$. This condition means that the sum of the Lagrange multipliers of each net must be zero. $g_i((\nu)_i)$ is defined as the optimal value of the subproblem:

$$\min_{x_i, (y)_i} \quad f_i(x_i, (y)_i) + (\nu)_i^T (y)_i \tag{2.19a}$$

$$\text{s.t.} \quad (x_i, (y)_i) \in \mathcal{C}_i, \tag{2.19b}$$

minimizng over $(\nu)_i$. A subgradient of $-g_i$ at $(\nu)_i$ is just $-(y)_i$, being $-(y)_i$ an optimal value of subproblem 2.19.

The dual of the original problem 2.14 is:

$$\max_{\nu} \quad g(\nu) = \sum_{i=1}^{N} g_i((\nu)_i) \tag{2.20a}$$

$$\text{s.t.} \quad E^T\nu = 0 \tag{2.20b}$$

The dual problem is the master problem and it can be solved using a projected subgradient method. Luckly, the solution is again very simple[30]: the subgradient of $-g$ is given by $y - E\hat{z}$, being $\hat{z} = (E^TE)^{-1}E^Ty$, which is nothing but the avarage value of the clone variables over each net. This means that at each iteration the net variables are calculated as the avarage value of the clone variables for each net and then compared with the individual clone variables. The difference of the clone and net variables is, like in example 2.7, nothing but the residuals of the consistency constraints.

We can sum up the algorithm in:

given initial dual vector ν that satisfies $E^T\nu = 0$ (e.g. , $\nu = 0$)
iterate:
Solve subproblems (possibly in parallel):

- Find x_i and $(y)_i$ that minimize $f_i(x_i, (y)_i) + (\nu)_i^T(y)_i$.

Calculate net variables as average of clone variables:

- $\hat{z} = (E^TE)^{-1}E^Ty$.

Update dual variables:

- $\nu := \nu - \alpha_k(y - E\hat{z})$.

with α_k an appropriate size.

2.5 Augmented Lagrangians and the Method of Multipliers

In the previous section, we have seen how powerful the dual ascent method is when it comes to decomposing big optimization problems into smaller ones and solving them in a distributed manner. The dual ascent method needs to meet some strict criteria to yield convergence, such us strict convexity or finiteness of f. The *augmented Lagrangians* came as an attempt to increase the robustness of the dual ascent method. Having the original problem:

$$\min_{x} \quad f(x) \tag{2.21a}$$

$$\text{s.t.} \quad Ax - b = 0 \tag{2.21b}$$

It's augmented Langrangian takes the form:

$$\mathcal{L}_\rho(x, \nu) = f(x) + \nu^T(Ax - b) + (\rho/2)\|Ax - b\|_2^2 \tag{2.22}$$

Which can be seen as the unaugmented Lagrangian of:

$$\min_x \quad f(x) + (\rho/2)\|Ax - b\|_2^2 \tag{2.23a}$$
$$\text{s.t.} \quad Ax - b = 0 \tag{2.23b}$$

That it an equivalent problem to 2.21, as for any feasible x the added term must be zero. The advantage of adding the extra term is that the dual function $g_\rho = inf_x\mathcal{L}_\rho(x, \nu)$ can be shown to be differentiable under rather mild conditions on the original problem[29]. The gradient of the dual function is found in the same fashion as we have seen in the previous section: first minimizing the Lagrangian over x and then evaluating the residual of the equality constraint to update the dual variable. The only difference is that the augmented Langrangian is minimized instead of the unagumented one and that where we had the step size α_k before we have now the *penalty parameter* ρ.

iterate:
x-update:

- Find $x^{(k+1)}$ that minimizes $f(x) + (\nu^{(k)})^T(Ax - b) + (\rho/2)\|Ax - b\|_2^2$.

dual-update:

- $\nu^{(k+1)} = \nu^{(k)} + \rho(Ax^{(k+1)} - b)$.

This is known as the *method of multipliers*. This method converges under far more general circumstances than the dual ascent (for example in cases where f takes the value $+\infty$ or is not strictly convex). However, this improved convergence comes at the cost of, even if the original function is separable, the augmented Lagrangian function is not, meaning that the x-update cannot be carried out separately for each x_i like in the dual ascent method. This disqualifies the method for distributed optimization purposes.

2.6 Alternating Direction Method of Multipliers

On the one hand, we have the dual ascent or dual decomposition methods that prove to be very convenient when it comes to solving optimization problems in a distributed manner but have very strict convergence rules. On the other, the method of multipliers that, thanks to the augmented Lagrange, improves the convergence of the dual ascent method at the cost of making the problem inseparable. The *Alternating Direction Method of Multipliers*(ADMM) tries to blend the advantages of

both methods, achieving a fast distributed algorithm with good convergence properties. In comparison to DD, the ADMM algorithm necessitates minor mathematical assumptions, which means that the ADMM algorithm only requires mathematical objective functions that are close, proper, and convex and that the unaugmented Lagrangian function of the associated convex optimization problem has a saddle point. [29]

The ADMM algorithm solves problems of the form:

$$\min_{x} \quad f(x) + g(z) \tag{2.24a}$$

$$\text{s.t.} \quad Ax + Bz - c = 0 \tag{2.24b}$$

The augmented Lagrangian is formed similarly as in the method of multipliers:

$$\mathcal{L}_\rho(x, z, \nu) = f(x) + g(z) + \nu^T(Ax + Bz - c) + (\rho/2)\|Ax + Bz - c\|_2^2 \tag{2.25}$$

The algorithm consists of:

iterate:
x-update:

- Find $x^{(k+1)}$ that minimizes $\mathcal{L}_\rho(x, z^{(k)}, \nu^{(k)})$
  $= f(x) + g(z^{(k)}) + (\nu^{(k)})^T(Ax + Bz^{(k)} - c) + (\rho/2)\|Ax + Bz^{(k)} - c\|_2^2$.

z-update:

- Find $z^{(k+1)}$ that minimizes $\mathcal{L}_\rho(x^{(k+1)}, z, \nu^{(k)})$
  $= f(x^{(k+1)}) + g(z) + (\nu^{(k)})^T(Ax^{(k+1)} + Bz - c) + (\rho/2)\|Ax^{(k+1)} + Bz - c\|_2^2$.

dual-update:

- $\nu^{(k+1)} = \nu^{(k)} + \rho(A^{(k+1)} + Bz^{(k+1)} - c)$.

The algorithm is very similar to the dual ascent method presented in 2.2.4 and the method of multipliers presented in 2.2.5. It consist of an *x-minimization* step followed by a *z-minimization* step and then the *dual variable update*.

The method of multipliers for the same problem would have the form:

$$(x^{(k+1)}, z^{(k+1)}) := \operatorname*{argmin}_{x,z} \mathcal{L}_\rho(x, z, \nu^{(k)}) \tag{2.26a}$$

$$\nu^{(k+1)} := \nu^{(k)} + \rho(A^{(k+1)} + Bz^{(k+1)} - c) \tag{2.26b}$$

As we can observe, in the method of multipliers the augmented Lagrangian is optimized jointly for the two primal variables, whereas in the ADMM the primal variables x and z are optimized in a sequential or alternating fashion, thus the name *alternating direction*. The ADMM can be viewed as a version of the method of multipliers where a single *Gauss-Seidel* pass over x and z is used instead of the

usual joint minimization [29]. This sequential minimization of x and z is what allows for decomposition when f and g are separable.

There are many variations of the ADMM algorithm. For a comprehensive review on them and mathematical proof of the convergence of the ADMM algorithms, please refer to [29].

In this thesis, we are particularly interested in ADMM applied to solve a consensus optimization problem. In consensus problems, we have several sub-problems with local variables, vector x. We also have a vector of global variables, z. Each local variable corresponds to one of the global variables. Several sub-problems will have local variables referring to the same global variable. Consensus is achieved when all local variables in the different sub-problems that refer to the same global variable agree in value.

Following the convention used previously, the group of local variables belonging to sub-problem i will be represented as $(x)_i$ to differentiate it from the i-th element of vector x, x_i.

Using the same definition of matrix E as in 2.16, as a binary matrix that maps which local variables x_i corresponds to which global variable z_j, the general form of a consensus optimization problem is:

$$\min_{x} \quad \sum_{i=1}^{N} f_i((x)_i) \tag{2.27a}$$

$$\text{s.t.} \quad (x)_i - (E)_i z = 0 \qquad \forall i \tag{2.27b}$$

The augmented Lagrangian associate to this problem is:

$$\mathcal{L}_\rho(x, z, \nu) = \sum_{i=1}^{N} f_i((x)_i) + (\nu)_i^T((x)_i - (E)_i z) + (\rho/2)\|(x)_i - (E)_i z\|_2^2 \tag{2.28}$$

Using the ADMM algorithm results in:

iterate:
x-update, possibly in parallel:

- Find $(x)_i^{(k+1)}$ that minimizes $\mathcal{L}_{\rho_i}((x)_i, z^{(k)}, (\nu)_i^{(k)})$
  $= f_i((x)_i) + (\nu)_i^{(k)T}((x)_i - (E)_i z^{(k)}) + (\rho/2)\|(x)_i - (E)_i z^{(k)}\|_2^2$.
  equivalent to minimizing:
  $f_i((x)_i) + (\nu)_i^{(k)T}(x)_i + (\rho/2)\|(x)_i - (E)_i z^{(k)}\|_2^2$.

z-update:

- Find $z^{(k+1)}$ that minimizes $\mathcal{L}_\rho(x^{(k+1)}, z, \nu^{(k)})$
  $= \sum_{i=1}^{N} f_i((x)_i^{(k+1)}) + (\nu)_i^{(k)T}((x)_i^{(k+1)} - (E)_i z) + (\rho/2)\|(x)_i^{(k+1)} - (E)_i z\|_2^2$.
  equivalent to minimizing:
  $= \sum_{i=1}^{N} -(\nu)_i^{(k)T}(E)_i z + (\rho/2)\|(x)_i^{(k+1)} - (E)_i z\|_2^2$.

dual-update:

- $(\nu)_i^{(k+1)} = (\nu)_i^{(k)} + \rho((x)_i^{(k+1)} - (E)_i z^{(k+1)})$.

In [29] is demonstrated how, similarly to what happened in DD, each of the global variables, i.e., elements of vector $z^{(k+1)}$, can be calculated as the average of the elements of the vector $x^{(k+1)}$ that refer to that global variable. Mathematically this can be written with the help of matrix E as:

$$z^{(k+1)} = (E^T E)^{-1} E^T x^{(k+1)} \tag{2.29}$$

leaving a much simpler algorithm:

iterate:
x-update, possibly in parallel:

- Find $(x)_i^{(k+1)}$ that minimizes:
  $f_i((x)_i) + (\nu_i^{(k)})^T(x)_i + (\rho/2)\|(x)_i - (E)_i z^{(k)}\|_2^2$.

z-update:

- Calculate $z^{(k+1)}$ as the average value of the local variables:
  $z^{(k+1)} = (E^T E)^{-1} E^T x^{(k+1)}$.

dual-update, possibly in parallel:

- $(\nu)_i^{(k+1)} = (\nu)_i^{(k)} + \rho((x)_i^{(k+1)} - (E)_i z^{(k+1)})$.

The resemblance between the ADMM consensus algorithm and the DD algorithm is even more evident if we rename the local variables as clone variables and the global variables as net variables.

We can see that the local or clone variables update, as well as the update of the dual variable, can be separately done in parallel. The global or net variable update may seem inseparable, posing the need of a centralized coordinator that gathers all the clone variables $x^{(k+1)}$, calculates the net variables $z^{(k+1)}$ and broadcasts them so that the subproblems can continue with the algorithm.

However, in the problem we are interested in, a certain strategy can be taken to avoid this issue and manage a fully distributed algorithm. As stated in chapter 1, this thesis focuses on the *optimal power flow*(OPF) applied to DC grids, which is an electrical network flow problem. The problem is decomposed in a way such that each node of the network has one sub-problem attached, and the global variables involved in each subproblem appear as local or clone variables only in itself and neighboring nodes. In other words, two sub-problems have clone variables of the same global variable only if in the actual network, the nodes are attached to have an electrical connection. Furthermore, in this problem, there are as many global variables as nodes (subproblems), and each node is "responsible" for one of them. The problem of the z-update can then be addressed with peer-to-peer communication between neighboring nodes. At each iteration, after the x-update, a communication round is carried out so that each subproblem sends to its neighbors the value of the clone variable of the global variable the neighbor is in charge of. This way, each subproblem receives all the clone variable values for their global variable and calculates the average value $z_i^{(k+1)}$. A second communication round is done where each subproblem communicates to its neighbors the resulting average value. This means that, by the end of this process, all subproblems have the relevant average values or elements of $z^{(k+1)}$ that they need to carry out the dual-update and proceed with the algorithm. This particular modification to the ADMM algorithm is called *Proximal Message Passing* or PMP ADMM.

For more insight in distributed optimization techniques applied to electrical power systems see [2] and [31]. For more examples of PMP ADMM algorithms applied to Electrical Power Systems optimization see [32] and [33].

With this, all the mathematical tools needed for this thesis were reviewed. More details on how the PMP ADMM is applied to our particular problem will be presented in chapter 3.

3 Distributed ADMM Algorithm for DC Systems

In this chapter, the Proximal Message Passing Alternating Direction Method of Multipliers with Sequential Convex Approximations (PMP ADMM SCA) [8] applied to an OPF for DC grids is described in detail. First, the entity concept is introduced, as it is of vital importance for the rest of the thesis. Then, the algorithm itself is described. All the math described in chapter 2 is applied step-by-step to the OPF problem until we get to the final form of the algorithm.

3.1 Entity Concept

In contrast to the centralized control strategy, where the whole optimization problem is solved by a central computer, in this distributed strategy each node of the electrical network will have its own computer running a part of the distributed algorithm. In this context, we introduce the *entity* concept. An entity is a combination of hardware and software able to:

- Send and receive data from other entities.
- Solve small optimization problems.
- Store data in its own memory.
- Set voltage and power of the electrical node it is in charge of.

Each entity is in charge of one node of the electrical grid, which means there are as many entities as nodes in the grid. With regards to the communication structure, the algorithm needs that, at least, communication between neighboring entities is possible. Two entities are considered to be neighbors of each other if an electrical connection between the nodes they are in charge of exists. A way to picture this relationship between the electrical grid and the entity network is that the entity network mimics the electrical grid, as seen in figure 3.1.

3.2 Centralized Formulation of the Problem

The optimal power flow (OPF) problem is one of the most important optimization problems that have to be solved for the proper management of the grid. The OPF

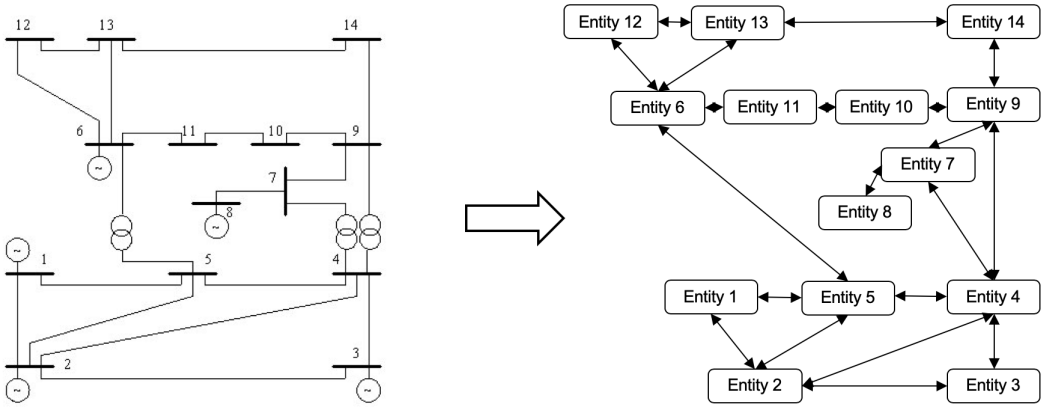


Figure 3.1: Electrical grid vs. entity network.

aims to supply the demanded energy at the minimum operational costs possible, considering the technical limitations of lines and generators and subjected to the power flow equations. The OPF is generally non-convex and NP-hard, and, therefore, challenging to solve. There are different approximations and relaxations of the OPF, being one of the most popular the so-called DC OPF, which is a linear approximation of the OPF applied to AC networks and must not be confused with an OPF applied to DC networks, as it is the case for this thesis.

The OPF formulation, minimizing quadratic generation costs and applied to a DC grid, is as follows:

$$\min_{p_g} \quad f(p_g) = \sum_{q=1}^{N} (\alpha_q + \beta_q p_{g_q} + \gamma_q p_{g_q}^2) \tag{3.1a}$$

$$\text{s.t.} \quad p_{g_q} = p_{load_q} + p_q \qquad \forall q, \tag{3.1b}$$

$$i_q = \sum_{j=1}^{N} G_{q,j} \cdot v_j \qquad \forall q, \tag{3.1c}$$

$$i_{qj} = G_{q,j} \cdot (v_j - v_q) \qquad \forall q, j \quad q \neq j, \tag{3.1d}$$

$$p_q = v_q \cdot i_q \qquad \forall q, \tag{3.1e}$$

$$p_{qj} = v_q \cdot i_{qj} \qquad \forall q, j \quad q \neq j, \tag{3.1f}$$

$$p_{gmin_q} \leq p_{g_q} \leq p_{gmax_q} \qquad \forall q, \tag{3.1g}$$

$$p_{fmin_q} \leq p_{qj} \leq p_{fmax_q} \qquad \forall q, j \quad q \neq j, \tag{3.1h}$$

$$p_{imin_q} \leq p_q \leq p_{imax_q} \qquad \forall q, \tag{3.1i}$$

$$i_{fmin_q} \leq i_{qj} \leq i_{fmax_q} \qquad \forall q, j \quad q \neq j, \tag{3.1j}$$

$$i_{imin_q} \leq i_q \leq i_{imax_q} \qquad \forall q, \tag{3.1k}$$

$$v_{min_q} \leq v_q \leq v_{max_q} \qquad \forall q, \tag{3.1l}$$

$$q = j = 1, 2, ..., N \tag{3.1m}$$

where

- $f(P_g)$ is the OF: minimization of generation costs.
- $\alpha_q, \beta_q, \gamma_q$ are the cost coefficients for the generator of node q.
- v_q is the voltage of node q.
- $G_{q,j}$ is the admitance value between node q and j. If no line exist between node q and j then $G_{q,j} = 0$.
- p_{g_q} is the power generated in node q.
- p_{load_q} is the power demanded in node q.
- p_q is the power flowing out of node q.
- i_q is the current flowing out node q.
- p_{qj} is the power flow between node q and j.
- i_{qj} is the current flow between node q and j.
- constraints 3.1g to 3.1l represent the upper and lower limits of all optimization variables (generation limits, current and power injection limits, current and power flow limits and voltage limits).

It is important to note that all the variables of problem 3.1 can be written as a function of the nodal voltages, which will help to decompose the problem. Another modification that can be done to the problem is to avoid using the voltages of all nodes in the term $G_{qj} \cdot v_j$. As is, we use the whole vector of nodal voltages multiplied by the whole row corresponding to node q of the admittance matrix G. However, element $Gqj = 0$ if the line between q and j doesn't exist. This means that the nodal voltages of nodes that are not direct neighbors of node q do not in the local equations for that node. We can define the subindex n_q as the subset of j that includes the nodes that are neighbors to node q, and M_q the total number of neighbors that node q has. This will transform the term $G_{qj} \cdot v_j$ into $G_{qn_q} \cdot v_{n_q}$.

Applying this modifications to problem 3.1 results in:

$$\min_{v} \quad f(v) = \sum_{q=1}^{N}(\alpha_q + \beta_q(p_{load_q} + v_q \cdot \sum_{j\forall n_q} G_{q,j} \cdot v_j) + \gamma_q(p_{load_q} + v_q \cdot \sum_{j\forall n_q} G_{q,j} \cdot v_j)^2) \tag{3.2a}$$

$$\text{s.t.} \quad p_{g_q} = p_{load_q} + v_q \cdot \sum_{j\forall n_q} G_{q,j} \cdot v_j \qquad \forall q, \tag{3.2b}$$

$$i_q = \sum_{j\forall n_q} G_{q,j} \cdot v_j \qquad \forall q, \tag{3.2c}$$

$$i_{qn_q} = G_{q,n_q} \cdot (v_{n_q} - v_q) \qquad \forall q, \tag{3.2d}$$

$$p_q = v_q \cdot \sum_{j\forall n_q} G_{q,j} \cdot v_j \qquad \forall q, \tag{3.2e}$$

$$p_{qn_q} = v_q \cdot G_{q,n_q} \cdot (v_{n_q} - v_q) \qquad \forall q, \tag{3.2f}$$

$$p_{gmin_q} \leq p_{g_q} \leq p_{gmax_q} \qquad \forall q, \tag{3.2g}$$

$$p_{fmin_q} \leq p_{qn_q} \leq p_{fmax_q} \qquad \forall q, \tag{3.2h}$$

$$p_{imin_q} \leq p_q \leq p_{imax_q} \qquad \forall q, \tag{3.2i}$$

$$i_{fmin_q} \leq i_{qn_q} \leq i_{fmax_q} \qquad \forall q, \tag{3.2j}$$

$$i_{imin_q} \leq i_q \leq i_{imax_q} \qquad \forall q, \tag{3.2k}$$

$$v_{min_q} \leq v_q \leq v_{max_q} \qquad \forall q, \tag{3.2l}$$

In order to enlightnen the notation, $\sum_{j=n_q^1}^{n_q^{M_q}}$, which means the sum over j for j equal to all the elements of the neighbour vector n_q, is writen like $\sum_{j\forall n_q}$.

3.3 Introduction of Clone and Net Variables

The problem has a structure that invites to decompose it per node q, but each nodal subproblem is linked to its neighbors by v_{n_q}. A consensus approach is taken, where we use the nodal voltages as global variables(from now on v_{net}, or *net voltages*), and each subproblem has a copy of the net voltages for their own node and its neighbours (from now on v_k or *clone voltages*).

Taking figure 3.1 as an example, the represented network has 14 nodes, which means 14 net voltages. Node 1 has two neighbours, nodes 2 and 5. This means that the OF of the subproblem that entity 1 will solve will depend from three clone voltages $v_k^1 = (v_k^{1,1}, v_k^{1,2}, v_k^{1,5})$, corresponding to the net voltages of nodes 1,2 and 5 $(v_{net}^1, v_{net}^2, v_{net}^5)$.

Using the clone and net voltages and adding the consistency constraints, the problem formulation changes to:

$$\begin{aligned}
\min_{v_k} \quad & f(v_k) = \sum_{q=1}^{N} f_q(v_k^q) \\
& = \sum_{q=1}^{N} (\alpha_q + \beta_q (p_{load_q} + v_k^{qq} \cdot \sum_{j \forall n_q} G_{q,j} \cdot v_k^{qj}) \\
& + \gamma_q (p_{load_q} + v_k^{qq} \cdot \sum_{j \forall n_q} G_{q,j} \cdot v_k^{qj})^2) && \text{(3.3a)} \\
\text{s.t.} \quad & p_{g_q} = p_{load_q} + v_k^{qq} \cdot \sum_{j \forall n_q} G_{q,j} \cdot v_k^{qj} && \forall q, \quad \text{(3.3b)} \\
& i_q = \sum_{j \forall n_q} G_{q,j} \cdot v_k^{qj} && \forall q, \quad \text{(3.3c)} \\
& i_{qn_q} = G_{q,n_q} \cdot (v_k^{qn_q} - v_k^{qq}) \quad \forall n_q, && \forall q, \quad \text{(3.3d)} \\
& p_q = v_k^{qq} \cdot \sum_{j \forall n_q} G_{q,j} \cdot v_k^{qj} && \forall q, \quad \text{(3.3e)} \\
& p_{qn_q} = v_k^{qq} \cdot G_{q,n_q} \cdot (v_k^{qn_q} - v_k^{qq}) \quad \forall n_q, && \forall q, \quad \text{(3.3f)} \\
& p_{gmin_q} \leq p_{g_q} \leq p_{gmax_q} && \forall q, \quad \text{(3.3g)} \\
& p_{fmin_q} \leq p_{qn_q} \leq p_{fmax_q} && \forall q, \quad \text{(3.3h)} \\
& p_{imin_q} \leq p_q \leq p_{imax_q} && \forall q, \quad \text{(3.3i)} \\
& i_{fmin_q} \leq i_{qn_q} \leq i_{fmax_q} && \forall q, \quad \text{(3.3j)} \\
& i_{imin_q} \leq i_q \leq i_{imax_q} && \forall q, \quad \text{(3.3k)} \\
& v_{min_q} \leq v_q \leq v_{max_q} && \forall q, \quad \text{(3.3l)} \\
& v_k^{qq} = v_{net}^q && \forall q, \quad \text{(3.3m)} \\
& v_k^{qn_q} = v_{net}^{n_q} \quad \forall n_q && \forall q, \quad \text{(3.3n)}
\end{aligned}$$

which is separable in q with exception of the consistency constraints 3.3m and 3.3n .

3.4 Alternating Direction Method of Multipliers (ADMM)

In order to apply the ADMM, first we must build the augmented Lagrangian (the dual variable name will be changed from ν to λ for a better differentiation from the clone variables v_k and the net variables v_{net}):

$$\mathcal{L}_\rho(v_k, v_{net}, \lambda) = \sum_{q=1}^{N} f_q(v_k^q) + (\lambda)_q^T (v_k^q - (E)_q v_{net}) + (\rho/2) \| v_k^q - (E)_q v_{net} \|_2^2 \quad (3.4)$$

where

- v_k is the set of all clone variables and v_k^q is the group of clone variables that belong to subproblem or entity q.

- E is a binary matrix that maps all clone variables to their net variables so that $v_k = Ev_{net}$ and $(E)_q$ is the portion of the matrix that handles the clone and net variables related to subproblem q.

- v_{net} is the set of all net variables.

- λ is the set of all dual variables and $(\lambda)_q$ is the group of dual variables that belong to subproblem or entity q.

Taking superindex (t) as the iteration number, and applying the algorithm seen in 2.6, the resulting iterative method is:

ADMM DOPF

Step 1 - Initiallization:

- Initialization of clone variables, net variables and dual varaiables, typically $v_k^{q(0)} = 1$, $v_{net}^{(0)} = 1$, $(\lambda)_q^{(0)} = 0$.

Step 2 - Clone voltages update:

- Each entity q solves a local OPF to find $v_k^{q(t+1)}$ that minimizes:
$\mathcal{L}_{\rho_q}(v_k^q, (E)_q v_{net}^{(t)}, (\lambda)_q^{(t)}) = f_q(v_k^q) + (\lambda)_q^{(t)T}(v_k^q - (E)_q v_{net}^{(t)})$
$+ (\rho/2)\|v_k^q - (E)_q v_{net}^{(t)}\|_2^2$.

  s.t. constraints 3.3b to 3.3l

Step 3 - Net voltages update:

- Find $v_{net}^{(t+1)}$ that minimizes:
$\mathcal{L}_{\rho_q}(v_k^{(t+1)}, v_{net}, (\lambda)^{(t)}) = \sum_{q=1}^{N}(\lambda)_q^{(t)T}(v_k^{q(t+1)} - (E)_q v_{net})$
$+ (\rho/2)\|v_k^{q(t+1)} - (E)_q v_{net}\|_2^2$.

  which, as demonstrated in [29], is equivalent to calculating the avarage of the clone voltages as:

  $v_{net}^{(t+1)} = (E^T E)^{-1} E^T v_k^{(t+1)}$

Step 4 - Dual variable update:

- Update dual variables locally:
  $(\lambda)_q^{(t+1)} = (\lambda)_q^{(t)} + \rho((v)_k^{q(t+1)} - (E)_q v_{net}^{(t+1)})$.

It can be observed that Step 3 is preventing the algorithm from being fully distributed algorithm, as it is the only one that can not be done locally.

3.5 Proximal Message Passing-Based Distributed ADMM

The next step is to apply the proximal message passing (PMP) technique, which includes two neighbor-to-neighbor communication tasks: one after the clone voltages update and another one after the average voltage calculation. The algorithm then looks like:

PMP ADMM DOPF

Step 1 - Initiallization:

- Initialization of clone variables, net variables and dual varaiables, typically $v_k^{q(0)} = 1$, $v_{net}^{(0)} = 1$, $(\lambda)_q^{(0)} = 0$.

Step 2 - Clone voltages update:

- Each entity q solves a local OPF to find $v_k^{q(t+1)}$ that minimizes:
  $\mathcal{L}_{\rho_q}(v_k^q, (E)_q v_{net}^{(t)}, (\lambda)_q^{(t)}) = f_q(v_k^q) + (\lambda)_q^{(t)T}(v_k^q - (E)_q v_{net}^{(t)})$
  $+ (\rho/2)\|v_k^q - (E)_q v_{net}^{(t)}\|_2^2$.

  s.t. constraints 3.3b to 3.3l

Step 3 - Net voltages update:
Clone voltages exchange (peer-to-peer communication):

- Each entity q sends to its neighbours n_q the clone voltage they have calculated for those neighbouring entities, $v_k^{qn_q(t+1)}$.

Net voltages update, locally:

- Each entity q calculates the average value of the clone voltages they have received from the neighbours $v_k^{n_q q(t+1)}$, including their own clone voltage $v_k^{qq(t+1)}$:
  $v_{net}^{q(t+1)} = (v_k^{qq(t+1)} + \sum_{j \forall n_q} v_k^{jq(t+1)})/(1 + M_q)$

Net voltages exchange (peer-to-peer communication):

- Each entity q sends to its neighbours n_q the net voltage $v_{net}^{q(t+1)}$ they have calculated.

Step 4 - Dual variable update:

- Update dual variables locally:
  $(\lambda)_q^{(t+1)} = (\lambda)_q^{(t)} + \rho((v)_k^{q(t+1)} - (E)_q v_{net}^{(t+1)})$.

Note that the communication tasks have been highlighted in maroon.

An example with a 5 node network will be used to illustrate the communication steps and average voltage calculations:

We start by the end of the first step of the algorithm, where each subproblem has already calculated the clone voltages:

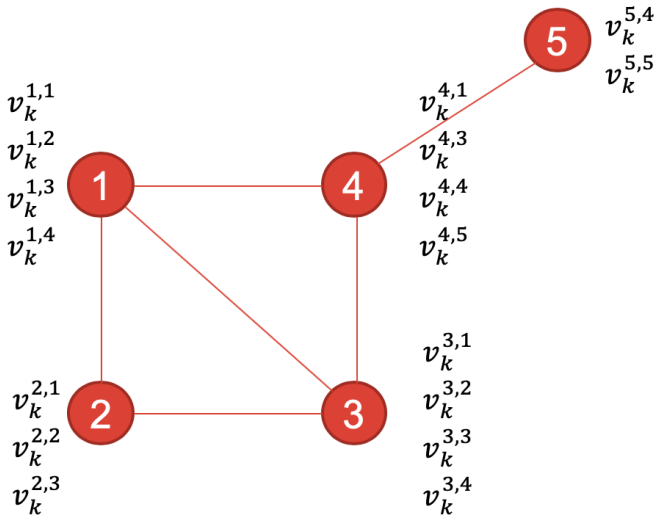


Figure 3.2: Clone voltages: calculation.

Then, the first communication round: each entity sends to its neighbours the clone voltage that refer to them:

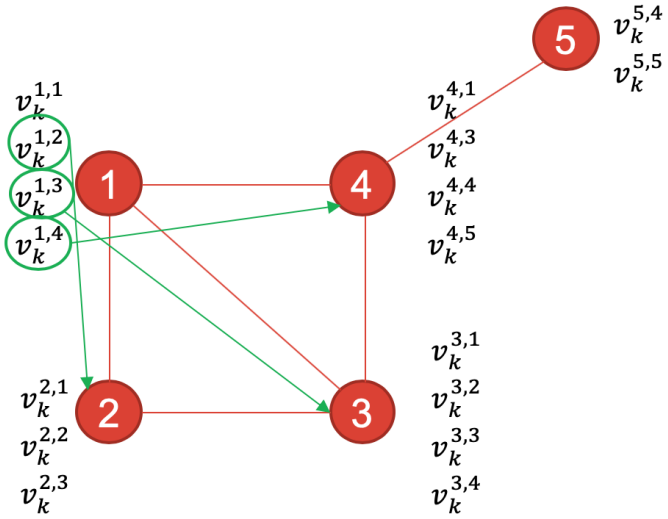


Figure 3.3: Clone voltages: sending.

Which means that each entity receives from the neighbours the clone voltages for its node:

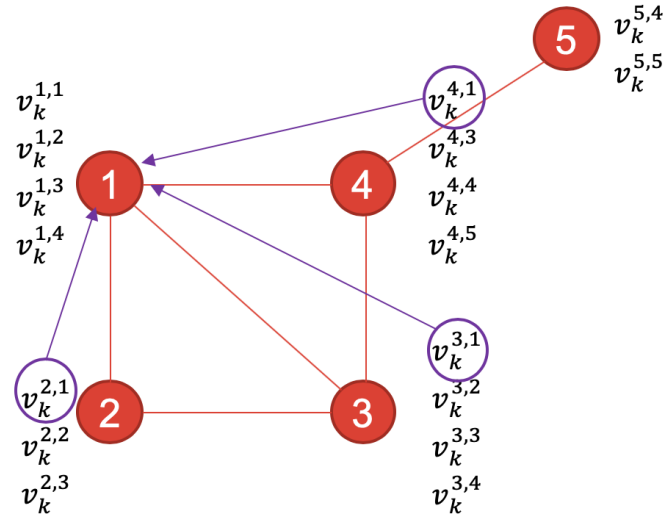


Figure 3.4: Clone voltages: receiving.

Now all entities have the necessary clone voltages to calculate the average value of their node:

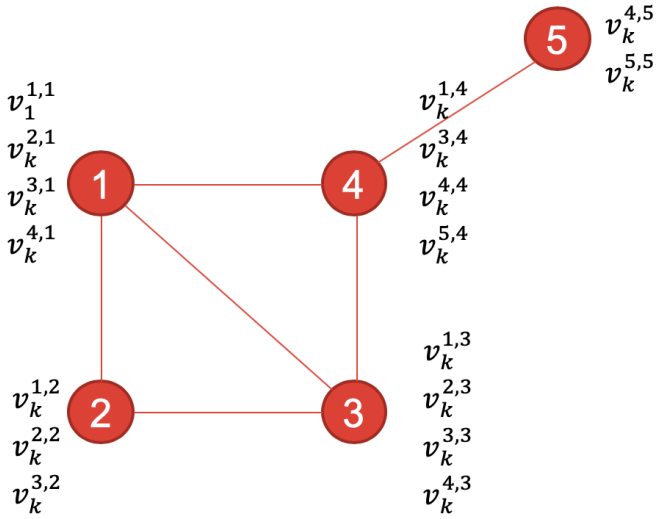


Figure 3.5: Final situation after first communication round.

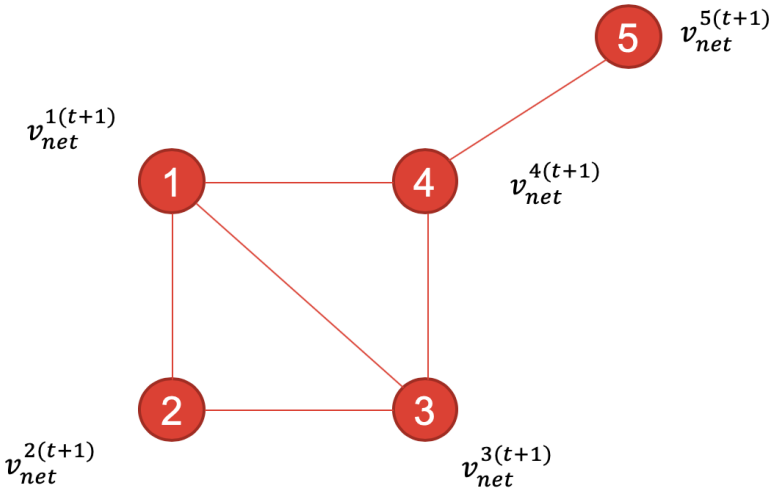


Figure 3.6: Net voltages: calculation.

The second communication starts, where the average values are sent to the neighbours:

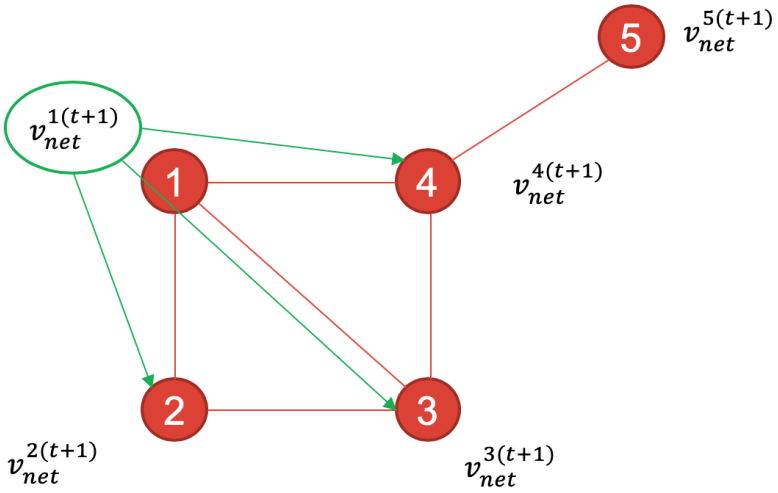


Figure 3.7: Net voltages: sending.

Which again means that all entities receive the net voltages of the neighbours:

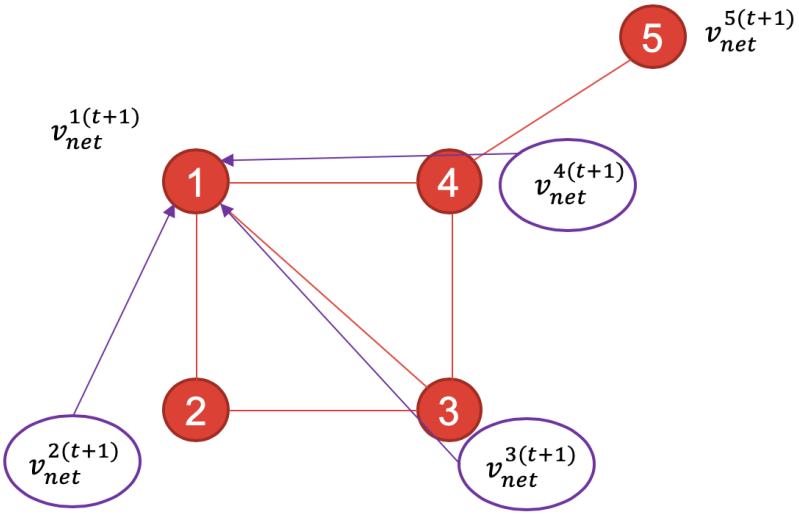


Figure 3.8: Net voltages: receiving.

By the end of this, all entities have the necessary information to proceed to the dual variable update state:

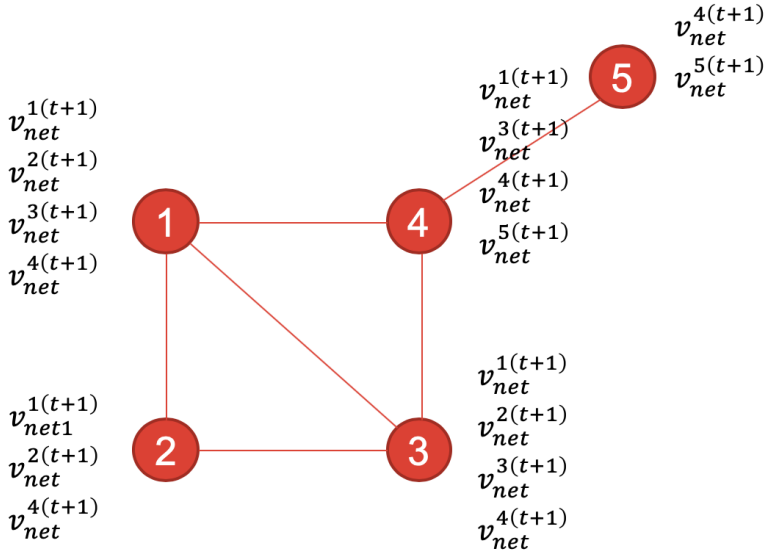


Figure 3.9: Final result after second communication round.

This is how the PMP technique manages to make the net variable update fully distributed only using neighbor-to-neighbor communication.

3.6 Sequential Convex Approximations

Contranints 3.3e and 3.3f are non-linear, but can be linearized using squential convex approximations (first-order Taylor approximations), leaving a quadratic programm (QP) that can be iteratively refined.

The linearized constraints have the form:

$$p_q = \sum_{j \forall n_q} (A_{qj}^{(x)} \cdot v_k^{qj}) + A_{qq}^{(x)} \cdot v_k^{qq} + A_{qq}^{\prime(x)} \tag{3.5a}$$

$$p_{qn_q} = B_{qn_q}^{(x)} \cdot v_k^{qq} + C_{qn_q}^{(x)} \cdot v_k^{qn_q} + D_{qn_q}^{(x)} \tag{3.5b}$$

where A, B, C, D are the Taylor coefficients and superindex (x) indicates the iteration of the sequential convex approximation.

At beginning of each iteration of the main algorithm $(t+1)$ the clone voltages of the previous iteration $v_k^{q(t)}$ will be used as intial Taylor points $T_q^{(0)}$. Then, the Taylor coefficients will be calculated as:

$$A_{qn_q}^{(x)} = G_{q,n_q} \cdot T_{qq}^{(x)} \tag{3.6a}$$
$$A_{qq}^{(x)} = \sum_{j \forall n_q} (G_{q,j} \cdot T_{qj}^{(x)}) + G_{q,q} \cdot T_{qq}^{(x)} \tag{3.6b}$$
$$A_{qq}^{\prime(x)} = -1 \cdot T_{qq}^{(x)} \sum_{j \forall n_q} (G_{q,j} \cdot T_{qj}^{(x)}) \tag{3.6c}$$
$$B_{qn_q}^{(x)} = G_{q,n_q} \cdot T_{qn_q}^{(x)} - 2 \cdot G_{q,n_q} \cdot T_{qq}^{(x)} \tag{3.6d}$$
$$C_{qn_q}^{(x)} = G_{q,n_q} \cdot T_{qq}^{(x)} \tag{3.6e}$$
$$D_{qn_q}^{(x)} = G_{q,n_q} \cdot T_{qq}^{(x)} \cdot (T_{qq}^{(x)} - T_{qn_q}^{(x)}) \tag{3.6f}$$
$$\tag{3.6g}$$

The now linearized constraints reduce the local optimization problem to a quadratic problem. Quadratic problems can be efficiently solved using well-known interior-point algorithms.[34]

The solution to the quadratic problem will give a set of new clone voltages v_k^q. This voltages are then used as new Taylor points $T_q^{(1)}$, the Taylor coefficients are recalculated and the quadratic problem solved again. The process is repeated till the clone voltages v_k^q and the Taylor points $T_q^{(x)}$ used to calculate them have a predefined convergence $((T_q^{(x)} - v_k^q) \leq \epsilon)$. Then we can say that the solution v_k^q is equal to $v_k^{q(t+1)}$ and proceed with the main algorithm.

To sum up the sequential convex approximations method:

Sequential Convex Approximations

given initial Taylor points $T_q^{(0)} = v_k^{q(t)}$**:**
iterate:
Taylor coefficients:

- Calculate $A_q^{(x)}, A_q^{\prime(x)}, B_q^{(x)}, C_q^{(x)}$ and $D_q^{(x)}$ as described in 3.6.

Solve the resulting local quadratic problem giving v_k^q:

$$\min_x \quad \frac{1}{2}x^T Px + qTx \tag{3.7a}$$
$$\text{s.t.} \quad Gx \leq h \tag{3.7b}$$
$$Ax = b \tag{3.7c}$$

Update Taylor points:

- $T_q^{(x+1)} = v_k^q$

Check stopping criterion:

- $(T_q^{(x+1)} - T_q^{(x)}) \leq \epsilon$

When finished:

- $v_k^{q(t+1)} = T_q^{(x+1)}$

3.7 Stopping Criteria

The last point to discuss is step 5 of the algorithm: the stopping criterion for the main algorithm. The objective is to achive consensus on the nodal voltages, i.e., all clone voltages that refer to the same net voltage are sufficiently similar:

$$|v_k^{(t)} - v_{net}^{(t)}| \leq \epsilon \tag{3.8}$$

When the first condition is met an additional condition is checked, where the total power generated must be greater than the total power demanded, but lower than the power demande times 1.1, i.e., the losses must be positive but less than 10% of the total demand:

$$p_{load}^{total} \leq p_{gen}^{total} \leq \frac{110}{100} \cdot p_{load}^{total} \tag{3.9}$$

If both conditions are met, the algorithm stops.

3.7.1 The Necessity of a "Ruler Entity"

It has been demonstrated how most of the steps of the algorithm can be done in a fully distributed way in a parallel fashion. However, the stopping criterion requires coordination between all entities of the network in order to check if the algorithm needs to continue or not. As discussed in section 3.7, the stopping criterion consists of two different conditions. The first one requires that all clone voltages are close enough to the net voltages. Each entity can independently check if their clone voltages meet the convergence criterion, but they have no way to know if all the

other entities meet this criterion or not. This poses the need to have a *ruler entity*. The ruler entity is in charge to ask each of the other entities of the network if voltage convergence exists (this will be communicated by means of a flag). Only if all answers are positive, then the ruler entity requests the power generated and demanded from each entity. The ruler entity checks then the second condition, if the total power losses are positive and less than 10%. If both conditions are met, the ruler entity sends a message to all other entities to tell them they can stop.

The stopping criteria check procedure can be summed up in:

Step 5 - Checking stopping criterion:

5.1. Local voltage convergence check:

- Each entity q checks if $(v_k^{(t)} - v_{net}^{(t)}) \leq \epsilon$ and sets a falg to 0 (false) or 1 (true) accordingly.

5.2. Voltage convergence flag gathering by ruler entity:

- The ruler entity gathers all voltage convergence flags.

5.3. Global voltage convergence check by the ruler entity:

- The ruler entity checks if all voltage convergence flags are equal to 1. If this is true, it sets a global voltage convergence to 1, if not it sets it to 0.

5.4. Global voltage convergence flag broadcasting by ruler entity:

- The ruler entity broadcasts the global voltage convergence flags to all other entities. If the flag is 0, all entities go back to Step 2. If the flag is 1, then the power criterion is checked.

5.5. Power gathering by ruler entity:

- The ruler entity gathers all the local power generation and demand.

5.6. Power criterion check by the ruler entity:

- The ruler entity checks if $p_{load}^{total} \leq p_{gen}^{total} \leq \frac{110}{100} \cdot p_{load}^{total}$. If this holds, it sets the stopping flag to 1, if not to 0.

5.7. Stopping flag broadcasting by ruler entity:

- The ruler entity broadcasts the stopping flag to all other entities. If the flag is 0, all entities go back to Step 2. If the flag is 1, then the algorithm finishes.

A flowchart of this step is presented in figure 3.10:

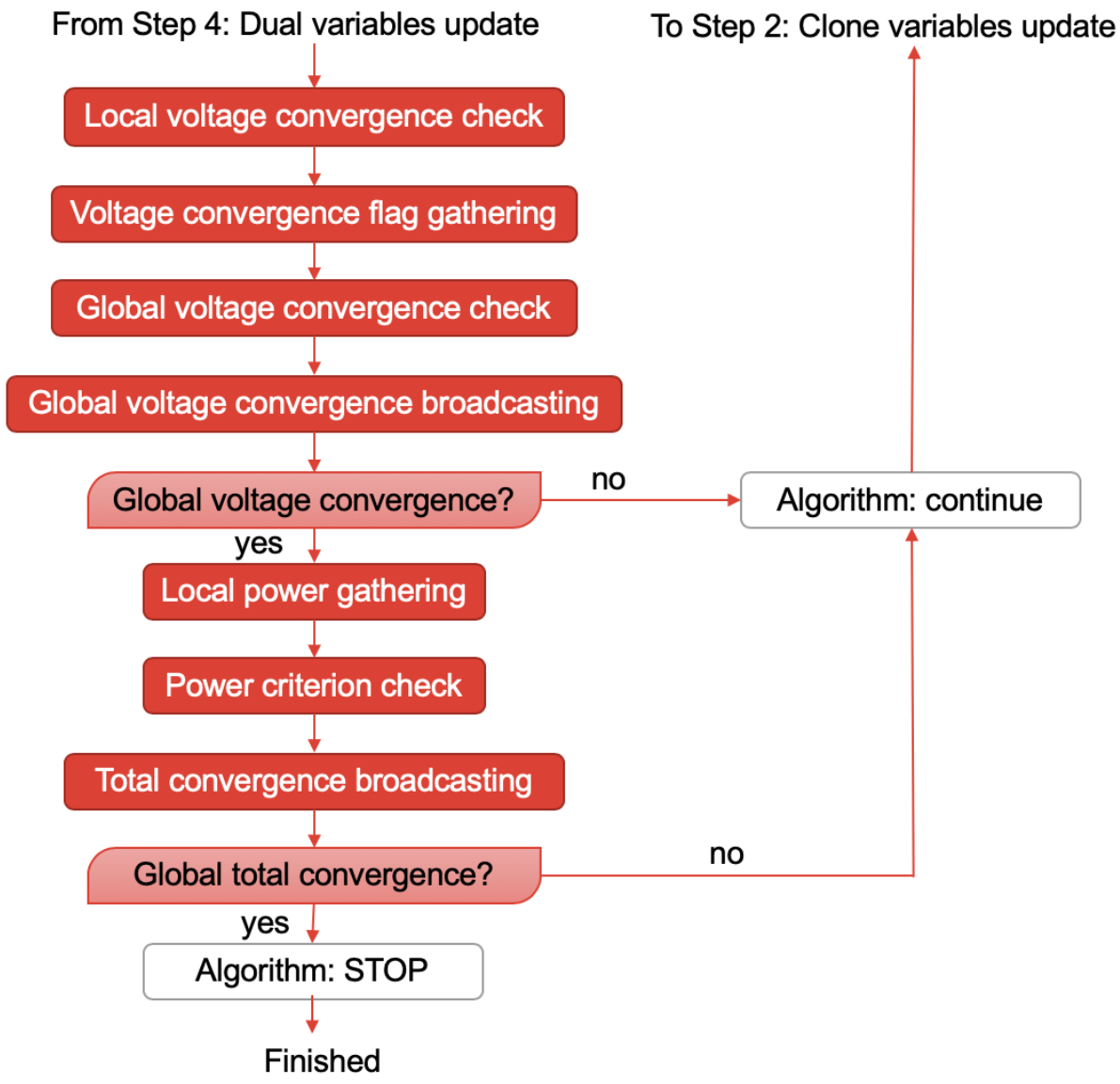


Figure 3.10: Step 5: Checking stopping criteria.

Following the example used in section 3.5, a graphic description is provided in the following figures:

First, each entity checks the local voltage convergence and sets the flag to 1 if convergence is achieved, 0 otherwise:

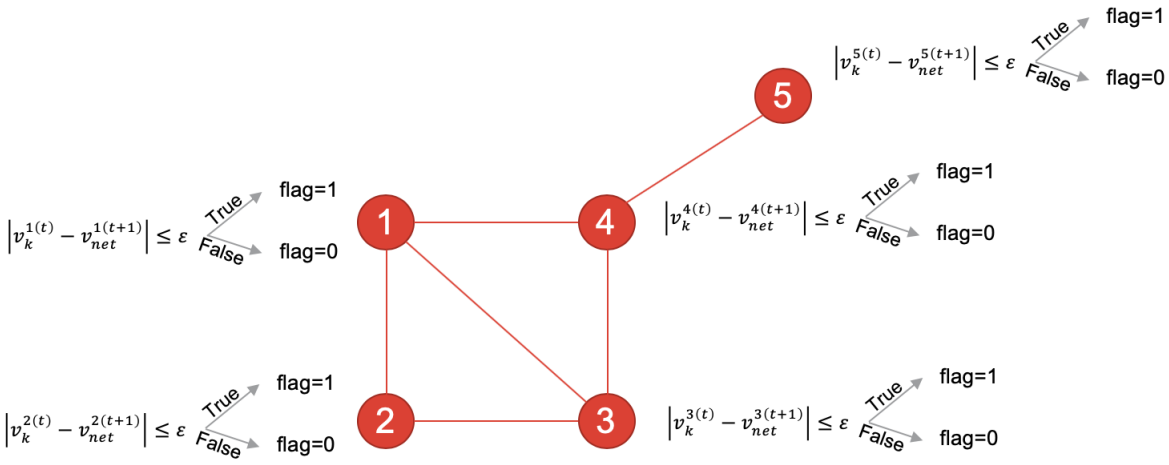


Figure 3.11: Step 5.1: Local voltage convergence check.

Then the ruler (in this case entity 5), gathers all flags, checks if all of them are equal to 1 and broadcast the flag for global voltage convergence back:

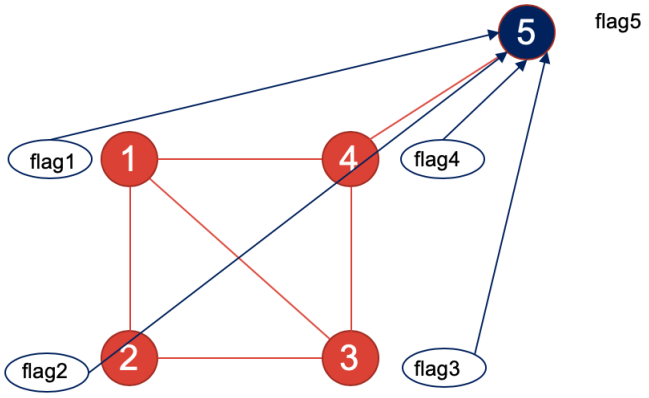


Figure 3.12: Step 5.2: Flag gathering.

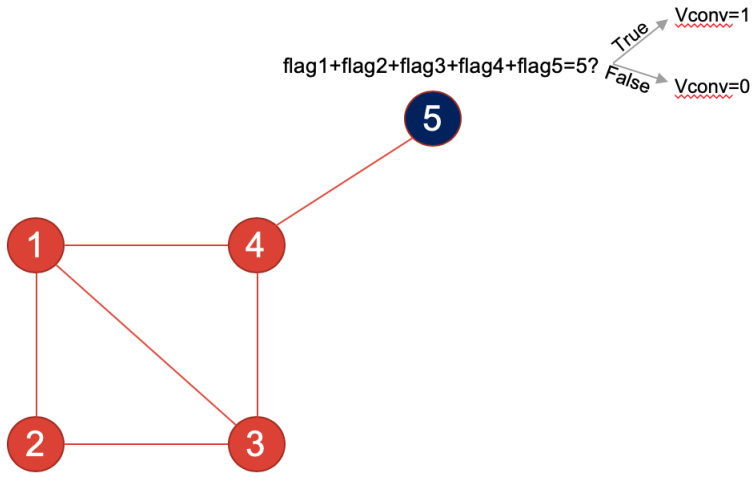


Figure 3.13: Step 5.3: Flag checking.

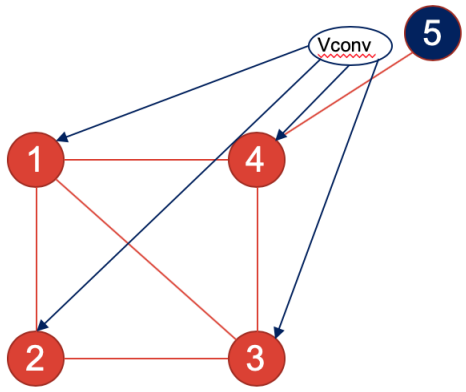


Figure 3.14: Step 5.4: Voltage convergence broadcasting.

If there is no voltage convergence, all entities know they have to continue and go back to step 2 of the algorithm (3.18). If there is, they know that the power criterion must be checked. Assuming there was voltage convergence, the ruler gathers the local power generated and demanded, checks if the power criterion is met and broadcasts back the order to stop or continue:

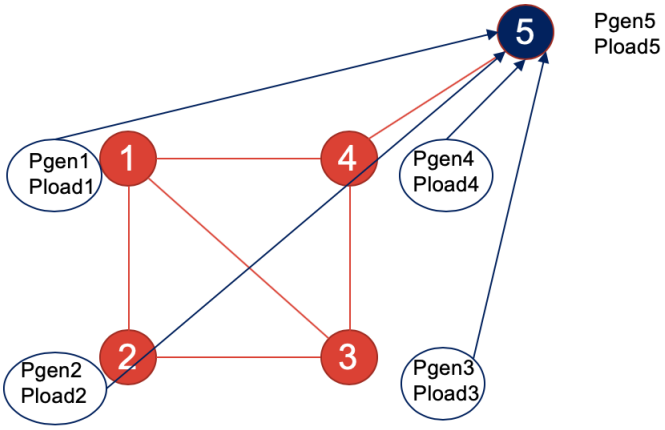


Figure 3.15: Step 5.5: Power gathering.

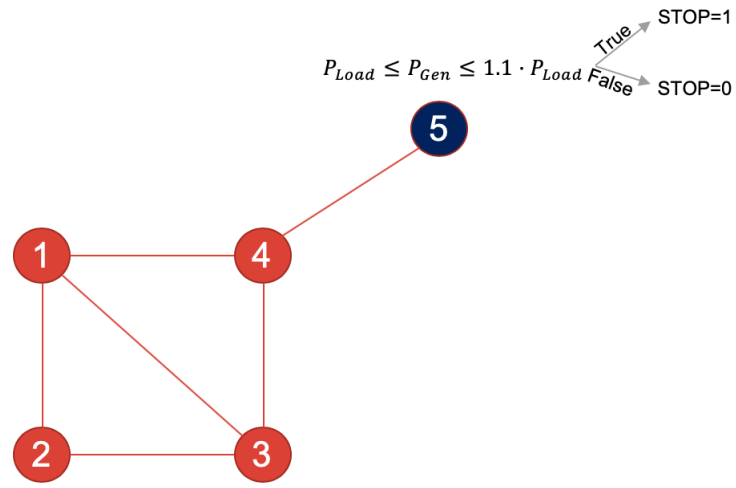


Figure 3.16: Step 5.6: Power criterion checking.

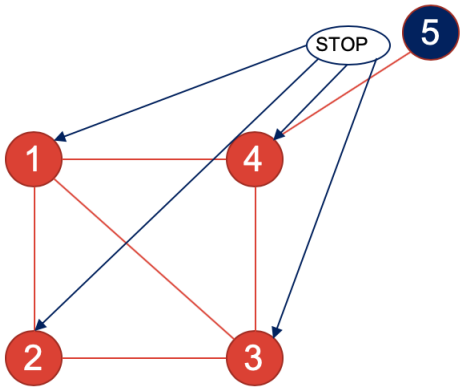


Figure 3.17: Step 5.7: Total convergence broadcasting.

The existence of a ruler entity seems to contradict the fully distributed objective of this algorithm. Furthermore, it looks like this ruler entity is introducing a single point of failure as, if the ruler entity fails, the algorithm cannot terminate. It is therefore important to remark that any entity can assume the role of ruler entity. If the entity assuming the role fails for any reason, this can be detected, and another entity will assume the role. The ruler entity can not only change from one execution of the algorithm to another but also during the same execution.

Another critical point is that the role of ruler entity assumes that communication between all entities is possible, as it needs a gather and a broadcast action for each condition of the termination criteria. Adding the fact that any entity can assume the role of ruler entity, all-with-all communication is a requirement. In section 3.1, it was explained how communication between, at least, electrical neighbors was needed for the algorithm. However, in the context of wireless communication, the possibility to have all-with-all communication does not add additional cost and gives for a much simpler, faster and more precise solution for the stopping criterion coordination problem than the alternative use of gossiping algorithms or flooding.

3.8 Full Algorithm

The complete algorithm is summed up in the following box:

PMP ADMM with Sequential Convex Approximations DOPF

Step 1 - Initiallization:

- Initialization of clone variables, net variables and dual varaiables, typically $v_k^{q(0)} = 1$, $v_{net}^{(0)} = 1$, $(\lambda)_q^{(0)} = 0$.

Step 2 - Clone voltages update:

- Each entity q solves a local OPF to find $v_k^{q(t+1)}$ that minimizes:
$\mathcal{L}_{\rho_q}(v_k^q, (E)_q v_{net}^{(t)}, (\lambda)_q^{(t)}) = f_q(v_k^q) + (\lambda)_q^{(t)T}(v_k^q - (E)_q v_{net}^{(t)})$
$+ (\rho/2)\|v_k^q - (E)_q v_{net}^{(t)}\|_2^2$.

  s.t. constraints 3.3b to 3.3l

  Applying Sequential Convex Approximations and solving the local OPF as a Quadratic Programming problem as in section 3.6.

Step 3 - Net voltages update:
Clone voltages exchange (peer-to-peer communication):

- Each entity q sends to its neighbours n_q the clone voltage they have calculated for those neighbouring entities, $v_k^{qn_q(t+1)}$.

Net voltages update, locally:

- Each entity q calculates the average value of the clone voltages they have received from the neighbours $v_k^{n_q q(t+1)}$, including their own clone voltage $v_k^{qq(t+1)}$:
$v_{net}^{q(t+1)} = (v_k^{qq(t+1)} + \sum_{j \forall n_q} v_k^{jq(t+1)})/(1 + M_q)$

Net voltages exchange (peer-to-peer communication):

- Each entity q sends to its neighbours n_q the net voltage $v_{net}^{q(t+1)}$ they have calculated.

Step 4 - Dual variable update:

- Update dual variables locally:
$(\lambda)_q^{(t+1)} = (\lambda)_q^{(t)} + \rho((v)_k^{q(t+1)} - (E)_q v_{net}^{(t+1)})$.

Step 5 - Checking stopping criterion:
(as in section 3.7. Requires ruler entity, and gathering and broadcasting communication.)

- If $(v_k^{(t)} - v_{net}^{(t)}) \leq \epsilon$ and $p_{load}^{total} \leq p_{gen}^{total} \leq \frac{110}{100} \cdot p_{load}^{total}$ then STOP, otherwise go back to Step 2.

A simplified flowchart of the algorithm is provided in 3.18:

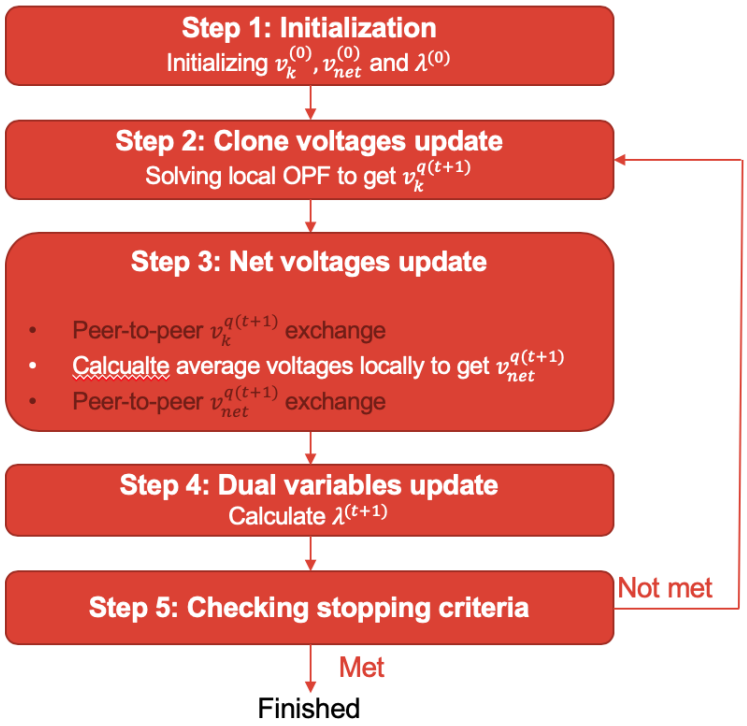


Figure 3.18: Flowchart of the ADMM DOPF algorithm.

4 Communication Failures

4.1 Types of Communication Failures

In this chapter, the different types of communication failures that can occur will be enumerated, and the strategy programmed into the algorithm to minimize the effect of them described.

The two different types of communication failures possible are:

- **Link failure**: The communication link between two entities fails, but the entities are still online and able to proceed with the algorithm and give set-points. This failure in the context of wireless communication, as we are assuming, would be infrequent. In the case of PLC, communication would be more common. As an example, the communication link between entities A and B fails in figure 4.1.

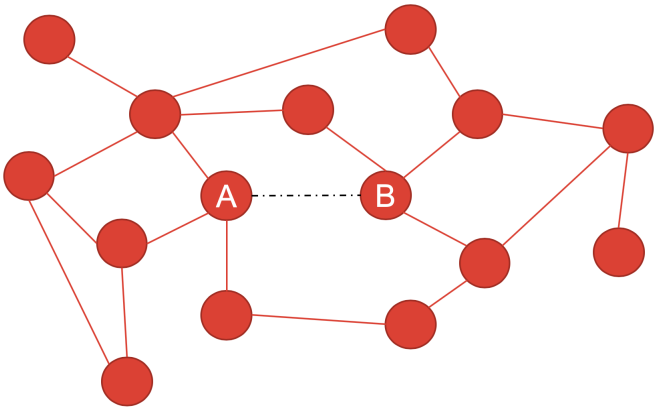


Figure 4.1: Example of link failure.

- **Entity failure**: Either the computer fails, or the program crashes. In any case, there is no possibility of sending or receiving values to or from any of the other entities, and there is no possibility to change the voltage and power set-points of the converter the entity is in charge of. The converter will have the last set-points the algorithm produced before failing and will still work. As an example, entity B fails in figure 4.2.

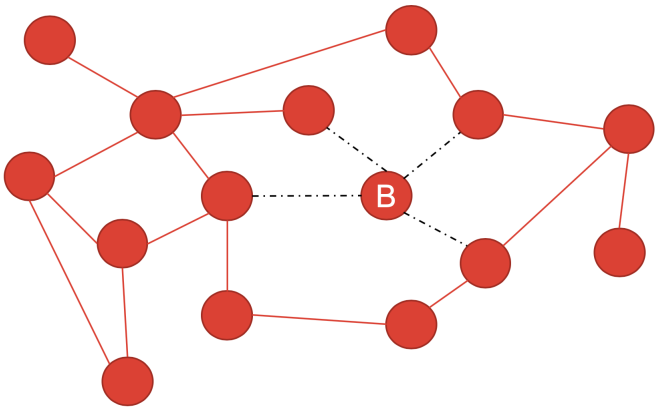


Figure 4.2: Example of entity failure.

4.2 Communication Failures Detection and Mitigation Strategy

The main concern is how to manage that the algorithm converges to a feasible stable solution in a reasonable time even if one (or more) of these failures occur. The problem is that entities have knowledge only of local values and use the clone and net voltages exchanges to "find out the needs of the neighbors" and reach a consensus. If communication between two entities fails (link failure), both entities will start converging to solutions not compatible with each other, as the voltage values they foresee for each other will be different to the actual ones. Even if the algorithm was able to converge under these circumstances, the solution would be impracticable to the real system. The reason of this impracticability is that if the link between two entities fail, from the algorithm perspective it would be like solving a problem where the electrical line between these two entities does not exist, i.e., the algorithm would be solving a problem with a different topology as the real one. In the case of an entity failure, this is even more problematic, as no new set-points can be given to the converter of the failed entity. The converter will stay with the last setpoints it had (the ones of the previous state of the system), as the entity is not able to change them. A strategy that is able to detect the communication failure and mitigate its effect is required.

The strategy designed to achieve this goal has two components, a local component and a global component:

- **On a local level**, if one entity (entity A) does not receive a message that it was expecting to receive from one of its neighbors (entity B), it will enter the *failure mode*. This mode means that entity A will fix its voltage to the value it had in the previous state of the system(which is common knowledge, all entities have in memory the voltage values of themselves and neighbors

of, at least, the previous state of the system). In other words, at the time of solving its local OPF, it will treat its own voltage as a constant parameter, not a variable anymore. This can be achieved by changing the upper and lower limits of the own voltage v_k^q to the voltage of the previous state of the system v_{ps}^q:

$$v_{ps}^q \leq v_k^q \leq v_{ps}^q \tag{4.1}$$

Entity A will also assume that the net voltage of entity B is equal to the previous state voltage of B. This is true because if the link between A and B is broken, entity B will also fix its own voltage to the previous state one. This means, for A: $v_k^{A,B} = v_{ps}^A$ and $v_{net}^B = v_{ps}^B$ in all subsequent iterations. Equally for B: $v_k^{B,A} = v_{ps}^B$ and $v_{net}^A = v_{ps}^A$. This strategy effectively replaces the need of the failed link because the entities are behaving in a way that, if the link was working, they would be transmitting the same values as the entities are artificially assuming. This is represented in figure 4.3. The solution in this case will potentially be less optimal as in the no failure case, as it is forcing two voltages, and therefore the poweflow in one line, to take a predetermined value. However, as long as the algorithm manages the reach a feasible solution, i.e., can change the generation profile and redirect the power flow through other lines to satisfy the demand, the solution will be stable.

Furthermore, if entity B failed completely, i.e., all links of B, it means that all neighbors of B will fix their voltage to the previous state voltage and assume that B also has the previous state voltage. Given that the converter of B will have the set-points of the previous state of the system (because entity B cannot change the set-points now that is down), and all neighboring nodes will also have the same voltages as in the previous state of the system due to the strategy, all the power flows of the lines connected to B will also be equal to the ones of the previous state of the system. Therefore, the power generated in B and the power flowing into B remains the same as in the prior state of the system (see figure 4.4). If the load of B was the same as in the previous state of the system, node B would be perfectly balanced and the solution given by the algorithm would be perfectly feasible and applicable to the system. As this is a real-time algorithm executed every few minutes, it is assumed that the load, although not the same, will not change much from one state of the system to the next one. The power balance in B would then not be perfect and the operation point of the system will slightly differ from the solution given by the algorithm. It must be noted that the solution of the algorithm will always slightly differ from the actual operation point due to the uncertainties regarding the system model (line parameters, control mode of the converters), the measurements, and the current state of the system which is input to the algorithm (available RES power, load). This strategy manages to keep the topology of the electrical network intact to the algorithm even if the communication network topology changes due to the communication failure, and assumes the misinformation about the load of the failed entity or

entities as one more uncertainty. The more entity failures happen the greater the uncertainty and the more the solution will deteriorate.

- **On a global level**, the ruler entity checks every iteration if any of the other entities are in failure mode via a gather action. If all entities confirm normal functioning, the ruler entity broadcasts a normal functioning flag and the algorithm proceeds normally. If at least one entity reports failure mode, this is broadcasted to all entities which will enter *emergency mode* and change the voltage convergence tolerance from $9e^{-4}$ to $9e^{-3}$ for faster convergence. If necessary, also power convergence criterion could be relaxed: as this is an emergency situation speed and feasibility are preferred over optimality or precision. If the ruler entity is the one that fails, no message reporting normal or failure mode is broadcasted to all the other entities, which will be detected. In that case, the ruler entity role will be assigned to another entity, which will automatically report the failure and the algorithm will proceed in emergency mode.

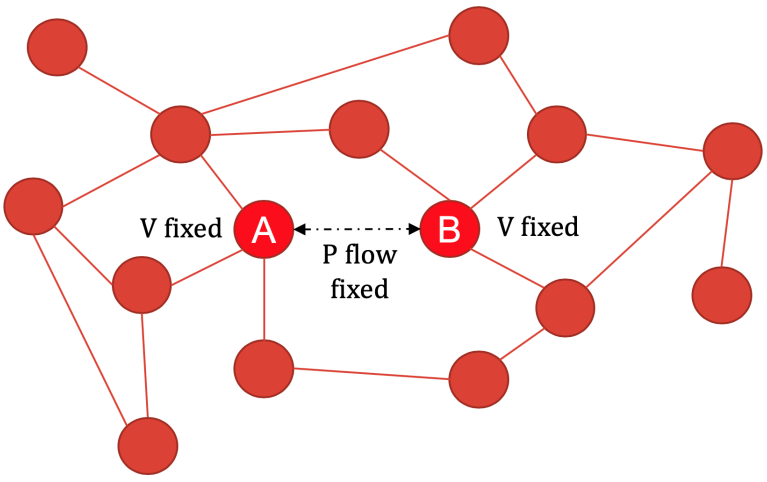


Figure 4.3: Behavior against link failure.

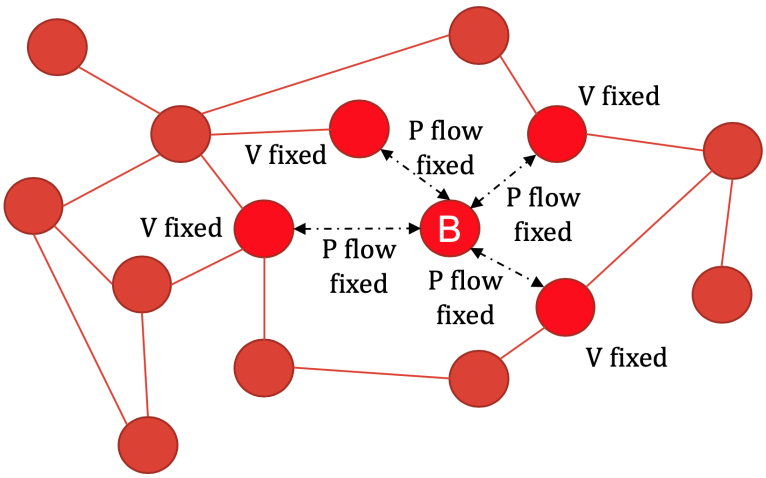


Figure 4.4: Behavior against entity failure.

The communication failures detection and mitigation strategy can be summarized as:

Communication failures detection and mitigation strategy

Communication failure check, locally:

- If any entity does not manage to establish communication with one of its neighbors:
  - Sets own voltage to previous system state voltage.
  - Assumes the net voltage of the failed neighbour is equal to the previous system state voltage of that entity.
  - Sets failure flag to 1.

Communication failure check by ruler entity:

- The ruler gathers all failure flags.
- If all are 0 then the emergency flag is also set to 0, if at least one flag is equal to 1 then the emergency flag is set to 1
- The ruler broadcasts the emergency flag.

Emergency state check, locally:

- If the emergency flag received from the ruler entity is equal to 0, proceed normally, otherwise relax voltage convergence by a factor of 10 and then proceed normally.

The complete PMP ADMM with Sequential Convex Approximations DOPF, including the aforementioned communication failures mitigation strategy, can be summarized as:

PMP ADMM with Sequential Convex Approximations DOPF
Including communication failures mitigation strategy

Step 1 - Initiallization:

- Initialization of clone variables, net variables and dual varaiables, typically $v_k^{q(0)} = 1$, $v_{net}^{(0)} = 1$, $(\lambda)_q^{(0)} = 0$.

Step 2 - Clone voltages update:

- Each entity q solves a local OPF to find $v_k^{q(t+1)}$ that minimizes:
$\mathcal{L}_{\rho_q}(v_k^q, (E)_q v_{net}^{(t)}, (\lambda)_q^{(t)}) = f_q(v_k^q) + (\lambda)_q^{(t)T}(v_k^q - (E)_q v_{net}^{(t)})$
$+ (\rho/2)\|v_k^q - (E)_q v_{net}^{(t)}\|_2^2$.

  s.t. constraints 3.3b to 3.3l

  Applying Sequential Convex Approximations and solving the local OPF as a Quadratic Programming problem as in section 3.6.

Step 3 - Net voltages update:
Clone voltages exchange (peer-to-peer communication):

- Each entity q sends to its neighbours n_q the clone voltage they have calculated for those neighbouring entities, $v_k^{qn_q(t+1)}$.

Communication failures detection and mitigation strategy
(Requires ruler entity, and gathering and broadcasting communication.)

Net voltages update, locally:

- Each entity q calculates the average value of the clone voltages they have received from the neighbours $v_k^{n_q q(t+1)}$, including their own clone voltage $v_k^{qq(t+1)}$:
$v_{net}^{q(t+1)} = (v_k^{qq(t+1)} + \sum_{j \forall n_q} v_k^{jq(t+1)})/(1 + M_q)$

Net voltages exchange (peer-to-peer communication):

- Each entity q sends to its neighbours n_q the net voltage $v_{net}^{q(t+1)}$ they have calculated.

Step 4 - Dual variable update:

- Update dual variables locally:
  $(\lambda)_q^{(t+1)} = (\lambda)_q^{(t)} + \rho((v)_k^{q(t+1)} - (E)_q v_{net}^{(t+1)})$.

Step 5 - Checking stopping criterion:
(as in section 3.7. Requires ruler entity, and gathering and broadcasting communication.)

- If $(v_k^{(t)} - v_{net}^{(t)}) \leq \epsilon$ and $p_{load}^{total} \leq p_{gen}^{total} \leq \frac{110}{100} \cdot p_{load}^{total}$ then STOP, otherwise go back to Step 2.

Other failures such as converter failures are not considered in this thesis for not being communication failures and because, from the algorithm perspective, if the power unit fails before or after the execution of the algorithm this would be taken into account in the next calculation round and the algorithm would proceed normally. If the failure occurs during the execution of the algorithm, a warning signal could make the algorithm reset, taking into account the missing power unit.

5 Implementation: Software and Hardware

In this chapter, the software and hardware used to implement the algorithm will be described.

5.1 Description of the Software

In this section, the software used in the development of this thesis will be described. There are two subsections. The first one explains why Python was chosen as the programming language. It then goes on introducing the different relevant packages, being the most important the MPI implementation for Python in section 5.1.1. In the second subsection, the simulation tool is presented.

5.1.1 Python

Python [35] is the coding language that was selected for the development of the simulation tool. Python is an object-oriented, interpreter-based programming language with dynamic typing and dynamic binding. It is oriented to programmer's productivity and code readability, allowing faster code development and clearer syntax in comparison to other languages. It is also a highly portable language along different hardware architectures and operative systems. It can easily be extended with new functions and data types implemented in C or C++. Besides, it is one of the most commonly used languages in scientific research. For that reason, many open-source packages are available, that work as easy-to-implement libraries for new projects. Modularity and code reuse is encouraged. Two of these packages are of particular relevance for this project: NumPy and cvxopt. Last but not least, Python is also very well suited for driving simulations in parallel architectures.

NumPy Package

NumPy [36] is a python library whose main characteristic is the addition of a potent N-dimensional array object. This arrays can be used as numerical arrays or as multi-dimensional containers for generic data. Python has the capability of handling list-objects natively, but NumPy brings much more efficient memory usage. For example, a 6-element vector containing number 1 to 6 occupies more than three times less memory as a numpy object [37]. However, the optimized memory allocation is not

the only advantage of using NumPy. The main reason for using NumPy is the multiple functions for creating and modifying N-dimensional arrays and the ability to perform mathematical operations to them, also much faster than the plane Python methods. The functions included in the package are practical and easy to use with a straightforward syntax and extensive documentation and community support. This functions include: creation of arrays of any size and dimensions full of zeros, full of ones, full of random numbers etc.; reshaping of arrays from one set of dimensions to another; matrix transposition or inversion; find maximum or minimum value; summation of elements; element-wise operations; slicing of arrays and many more. In a nutshell, the use of NumPy is not only convenient for the programmer, but it also helps to make much faster and efficient codes.

Convex Optimization Solver for Python (CVXOPT)

CVXOPT [38] is a free software package for Python that includes several solvers for cone programming. The user's manual can be found online in [39]. A discussion of the interior-point algorithms used in the solvers can be found in [40]. In this thesis, we were interested in the quadratic programming solver. It has a very simple syntax where, given the standard form of any quadratic problem:

$$\min_{x} \quad \frac{1}{2}x^T Px + qTx \tag{5.1a}$$

$$\text{s.t.} \quad Gx \leq h \tag{5.1b}$$

$$Ax = b \tag{5.1c}$$

where

- x is the vector of optimization variables.
- P, G, A are coefficient matrices.
- q, h, b are coefficient vectors.
- $\frac{1}{2}x^T Px + qTx$ is the OF in matrix form.
- $Gx \leq h$ represents the inequality constraints in matrix form.
- $Ax = b$ represents the equality constraints in matrix form.

the call to the solver consist in $sol = solvers.qp(P, q, G, h, A, b)$.

This was the solver chosen to solve the small local quadratic problems in each entity.

MPI Package for Python (mpi4py)

MPI stands for *Mesage Passing Interface*, which is a standard that defines the syntax of library routines that allows the parallelization of programs in the main scientific programming languages. [41] mpi4py is a package that brings the MPI-2 C++ bindings to Python. [42]. There are many functions and communication modes defined in the MPI standard, but the ones relevant for this thesis are:

- Regarding the number of senders and receivers:

  Point-to-point communication: One sender to one receiver. In this thesis: when entities send particular values to concrete neighbors before and after the average voltage calculation.

  Gather: One receiver, many senders. In this thesis: when all entities send convergence information to the ruler entity.

  Broadcast: One sender, many receivers. In this thesis: when the ruler entity sends to all the others convergence results.

- Regarding the overlapping of computation and communication:

  Blocking communication: once an entity sends a message, it blocks till it receives confirmation that the receiver entity has got the message. This method doesn't allow for the overlapping of computation and communication. Not used in this thesis.

  Non-blocking communication: sending and receiving requests are issued, but the entity can continue working until the moment it needs a value form the receiving requests when it will wait until the value is received. This gives better performance and avoids dead-locks. All communications in this thesis are non-blocking.

- Regarding the type of data transferred:

  Generic Python data objects: Take more memory space and yield slower communication times. Use only recommended when buffer-like objects are not an option (for example, while communicating dictionaries).

  Buffer-like objects (NumPy arrays): Faster and more efficient than the transmission of generic python data objects. The arrays sent need to be homogeneous in type. As in this thesis we deal with the transmission of numerical values or numerical arrays, this does not pose a problem. All communications in the thesis are of buffer-like objects.

Each of the parallel programs that MPI launches is called a *process*. MPI tries to optimize the process allocation so that the computation time and communication speed are maximized and the waiting time is minimized. Generally, if there are fewer processes than cores available in the machine, one process will go to one core. To which core each process goes is, if not forced otherwise, decided by MPI, again,

in an optimized way. If there are more processes than cores, *overbooking* occurs, where several processes are assigned to the same core and the processes "take turns" to get computation time in the core.

5.1.2 Simulation Tool

One of the contributions of this thesis is the development of a simulation tool to analyze the effects of communication failures was developed. This tool is written in Python and is used through a user interface in the Terminal/Command Window. It has been tested in MacOS and Linux systems, and it could be easily adapted for use in Windows systems. The algorithm is executed in parallel form: the tool launches as many MPI processes as entities in the network and, in case of running it in the HPC cluster, it lets the user choose how many nodes of the cluster to use. This tool allows to choose one of the different test networks presented in section 6.1 and run the algorithm with or without failures in the communication. The number of networks that the tool is able to simulate could be easily expanded. The communication failures can be failures in links (communication between two specific entities) or failures in entities (one entity fails to communicate with all others). They can also be single failures or multiple failures and, in case of multiple failures, occur all at the same time (same iteration of the algorithm) or each at different times. In case of wanting to implement a link failure, as chances are that the user does not know by heart which nodes are electrically connected to which ones, the tool gives the option to the user to ask it what neighbors any given entity has. The tool also has a checking mechanism that is continuously checking the input of the user to see if it is valid or not, and guides the user in case of wrong input. A brief results report is shown in the command window once the algorithm finishes, and the complete results are stored as *.csv* files with automatic naming and classification of the files to identify the network and failures simulated. Additionally, the full results can be automatically graphed using Matlab and saved as *.fig* and *.png* if so desired.

Code, File Structure and User Interface

The tool was designed with reusability and expandability in mind. It was programmed in a modular form, where modules can be activated, deactivated, changed or added. This way, it allows for future improvements and could be easily adapted for different projects. There are seven main files or modules:

- **start.py**: is the starting module. This is the file that has to be called to start the tool. It is called by typing "python3.6 start.py" in the command window when located in the same folder as the Python files. It asks the user what network to simulate, what entity should be the ruler and if the program is being executed in the HPC cluster or not. If it is being executed in the cluster, it will ask how many computing nodes to use. Then it will call *Main.py* with the proper MPI command, which has already been configured by *start.py* to

load the selected network. A demonstration of the working of this module can be found in figure 5.1.

- **Main.py**: is the core module. It loads the appropriate case into *Uvar.py* and asks the user if communication failures should be implemented. If affirmative, *embyUI.py* will be called to let the user input the failures that should happen. Then *Main.py* calls one of the functions of *EntityModule.py*: EM_start. This function prepares all the matrices that will be needed to store the data of each entity according to the network size, the maximum number of iterations and the number of neighbors each entity has. Then, the algorithm finally starts by calling the second function of *EntityModule.py*: EM_main. In EM_main the PMP ADMM SCA algorithm is implemented, which includes the local OPF solving by linearizing the power constraints and calling the quadratic programming CVXOPT solver and all the communications involved in the PMP method and the stopping criterion check routine using the MPI commands. In other words, steps 2 to 5 shown in figure 3.18 are implemented in EM_main. Also in EM_main, once the stopping criterion is met, the brief report is printed to the command window, the whole results saved as .csv and the Matlab script *plotcsv.m* called to plot the results and store them. A demonstration of the brief report can be seen in figure 5.3. In figure 5.4 the saving and plotting take place.

- **EntityModule.py**: is the entity module. Contains EM_start and EM_main.

- **EMem.py**: is the entity memory module. It contains the memory matrices of the entities.

- **Uvar.py**: is the simulation parameters module. It contains the simulation parameters. It is a copy of *UvarX.py*, being X the case selected.

- **embyUI.py**: is the communication failure module. It is in charge of asking the user what failures to implement and modify the necessary matrices (the communication matrices that define which entity communicates with which and at what time) so that the desired failures occur. It asks first if the user wants to implement link failures or entity failures. If the user selects link failures, it gives the possibility to check the neighbors of any desired entity. Once the user has finished checking or if the selection was entity failures, it asks the user to introduce the failure or failures and the time of the algorithm at which they should happen. Then it sets up the failures, checks if the ruler entity is affected by them or not and changes the ruler entity if necessary and prepared the unique ID of the scenario so that it can be assigned to the names of the result files to identify them easily. For example, ID: n30_EF_[6,28]it[1200,300]r1 means: 30 nodes (n30), entity failure (EF), entities 6 and 28 fail at iteration 1200 and 300 respectively ([6,28]it[1200,300]), entity 1 is the ruler (r1). A demonstration of how communication failure modules works can be seen in figure 5.2.

- **UvarX.py**: is the parameters file for network X. It contains all the simulation parameters for that network. This comprises all the network parameters (admittance matrix, generation and line limits, generation costs and loading) and all the algorithm parameters (penalty parameter, convergence tolerance and maximum number of iterations). New *UvarX* can be created following the format and naming of the existing ones to simulate new networks.

```
(base) 083-121:Parallel miguel$ python3.6 start.py

                WELCOME TO THE USER INTERFACE FOR "DCG-FDOP"
                ----------------------------------------
                        created by Miguel Berbegal

        -Please select the case number (5, 14, 30, 57, 118, 300):_ 14
        -What entity should assume the role of ruler?:_ 2
        -Are you executing this in the swarm HPC cluster?(yes/no):_ no

mpiexec -n 14 python3.6 Main.py


Loading case...
```

Figure 5.1: Example of UI: start.py.

```
Case loaded successfully

Initializing algorithm, please wait, this may take a couple of minutes...

        -Would you like to implement any communication/entity error?(yes/no):_ yes
        -What type of error woud you like: entity failures(ef) or link failures(lf)?:_ lf
        -Would you like to check first what neighbours a particular entity has?(yes/no):_ yes
        -What entity would you like to check?:_ 3
        The neighbours of entity 3 are:
         [2, 4]
        -Would you like to check another entity's neighbours?(yes/no):_ yes
        -What entity would you like to check?:_ 10
        The neighbours of entity 10 are:
         [9, 11]
        -Would you like to check another entity's neighbours?(yes/no):_ no
        -Please write a list of the links you want to fail.
        (example: "2 3, 6 7, 1 14"):_ 2 3, 10 11

The links that will fail are:
['2 3', '10 11']
Note: All the links that don't exist in the actual network will be ignored.

        -At what iteration would you like the failures to occur?
        If you introduce one value all will happen simultaneously, if not, you
        will have que introduce a vector of the same length as the ef vector:_ 30 50

Starting main algorithm...
```

Figure 5.2: Example of UI: embyUI.py.

```
----------------------------------------------------------------------
FINAL RESULTS BRIEF REPORT:



        The final voltages are:
         [1.0120042173679786, 1.0129793716123316, 0.9875419834600243, 0.9954968488858742, 1
.0005626859316803, 1.0124375856534518, 0.9992758757728216, 1.0207692087414597, 0.9877874894
292578, 0.9890052591364464, 0.997977024653547, 1.0028910688158568, 0.9983180622845129, 0.98
10698088752824]

        Power:
         [0.16679092843221857, 1.3631420260510048, 0.2887665478851985, 1.3417390130765207e-
08, 1.0292448155779776e-07, 0.6681447667177427, -1.0655871325224346e-07, 0.2878488149255686
3, -4.564853894506486e-08, -1.9944793741412276e-07, -2.3239817079510096e-08, -1.69984761104
1348e-08, -3.936143405314673e-08, -5.63151764415932e-09]
        OFV:
         [-1525682.1728530964, -2508243.0131143127, -1495184.122099662, -2983981.783308169,
 -2533661.487114, -2512572.9462423343, -2003758.9897855266, -1020236.3020430985, -2453617.6
32079852, -1474283.4027897648, -1504289.3012390889, -1513814.883424867, -1994969.274918464,
 -1467659.280909834]

        Power generated:  2.774692763467171
        Total demand:  2.59

        Convergence was set to:  0.0009000000000000001
        Max iterations was set to:  10000
        Iterations runned:  53
        Voltage convergence reached: yes
        Power convergence reached: yes
        time/iteration [seconds]: 0.02584745792242197


----------------------------------------------------------------------
```

Figure 5.3: Example of UI: EntityModule.py. Brief results report.

```
Saving the results as .csv...
Directory  14nodes/failure_cases  created, results will be stored there...

Results saved.

Calling Matlab...


                        < M A T L A B (R) >
                Copyright 1984-2018 The MathWorks, Inc.
                 R2018b (9.5.0.944444) 64-bit (maci64)
                            August 28, 2018


To get started, type doc.
For product information, visit www.mathworks.com.

Importing .csv files to Matlab...
Ploting and saving...
Saved. Exiting Matlab...
Removing all temporal files...


Finished.
```

Figure 5.4: Example of UI: EntityModule.py. Saving and plotting results.

5.2 Description of the Hardware

In this project, two different hardware configurations were used. The first one, a MacBook Pro early 2015, was mainly involved in the development and first trials of the code. The second, the Swarm High-Performance Computing Cluster at the E.ON Energy Research Center, was the one mainly used for the simulations, especially the ones of bigger systems. Details of both computers will be given in the following two subsections.

5.2.1 SWARM HPC Cluster at the E.ON ERC

The SWARM cluster has the following main characteristics:

Number of nodes	4
Cores per node	24
Processor per node	Intel Xeon CPU E5-2658 (v3)
RAM per node	128 GB
Operating system	Centos 7

Table 5.1: Swarm cluster specifications.

Each node of the cluster has 24 cores organized in two sockets. Each socket has its own shared memory. It is important to note that in spite of the shared memory, when we execute the parallel algorithm the shared memory is divided into as many parts as processes (or entities) and each core has its own area of private memory that no other core can access. This means that to share values, there has to be communication, not like in the sequential execution where the memory is truly shared and all entities can access the whole memory, as explained in section 6.2. A graphic depiction of the architecture of the node is given in figure 5.5.

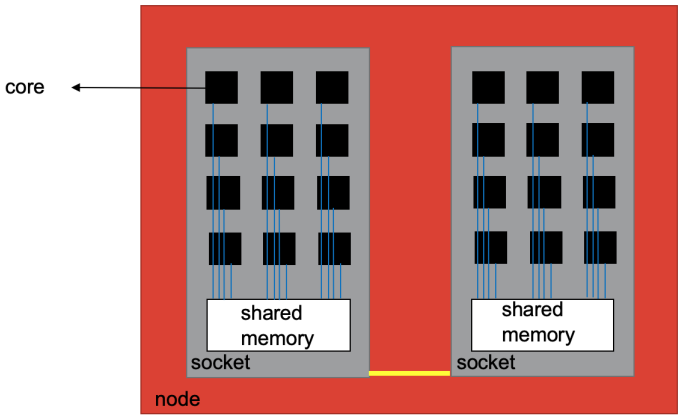


Figure 5.5: Swarm node.

The SWARM cluster has four identical nodes communicated via InfiniBand, which means 96 cores available.

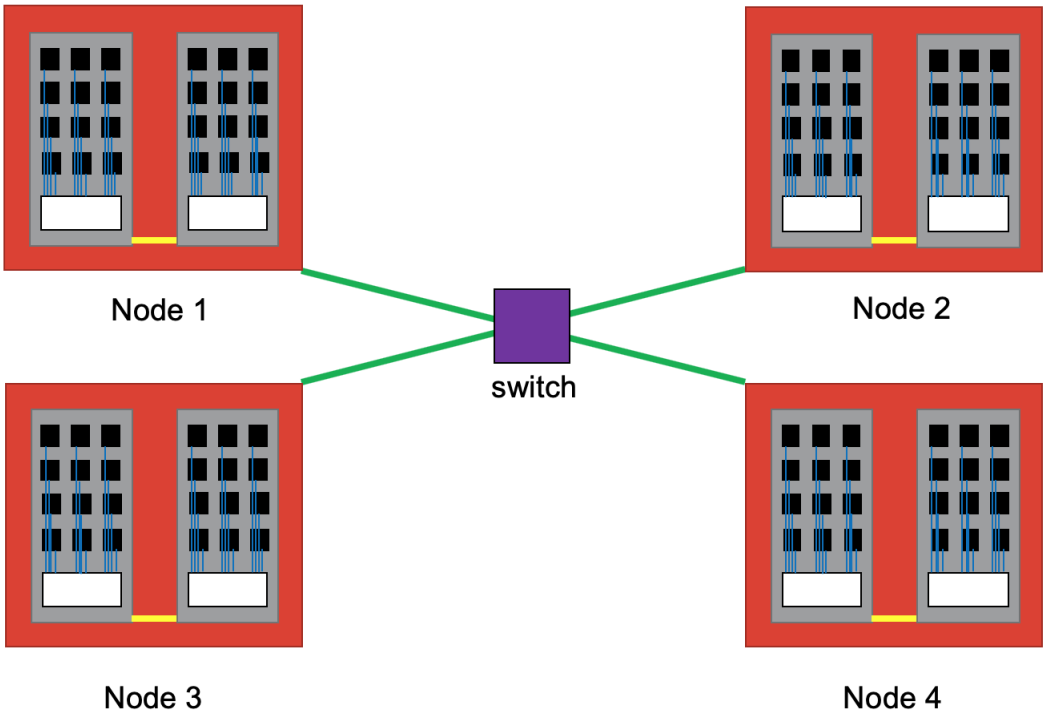


Figure 5.6: SWARM cluster. (Blue lines: communication through shared memory, yellow lines: inter-socket communication, green lines: InfiniBand).

Regarding how the cores will communicate, depending on where the two cores that have to communicate are allocated, the communication will be achieved in different ways. Cores within one socket will communicate through the **shared memory** (blue lines), cores between sockets will have to go through the **inter-socket path** (yellow). Cores between two nodes will communicate via **infiniband** (green). The speed of communication is as follows: Contrary to what could think InfiniBand is the fastest because it is a hardware implementation and it skips the OS. Then shared memory access is in second place. In theory, all 12 cores of a socket can access in parallel the shared memory, but in reality, all this goes through the memory manager and bottlenecks can happen here. The biggest limitation will be the inter-socket communication. This different mixed communication technolgoies, the hardware limitations and the MPI usage make this implementation not an accurate representation of what would happend in the field, with geographically separated computers communicating via wireless technology, but can nonetheless be usefull to study the challenges that come together with a fully distributed algorithm.

5.2.2 Personal Computer

The main characteristics of the PC are summarized in table 5.2.

Cores	2
Processor	Intel Core i5-5257U 2,7GHz
RAM	8 GB 1866 MHz LPDDR3 SDRAM
Operating system	macOS Mojave

Table 5.2: MacBook Pro specifications.

It is a single-socket dual-core machine with one shared memory, as represented in figure 5.7.

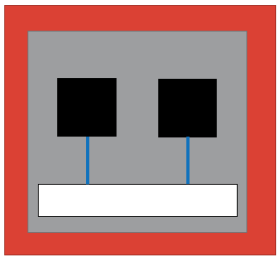


Figure 5.7: MacBook Pro structure.

All communication is done via shared memory access. The fact that only two cores are avialble means that all networks will suffer from overbooking in this machine.

6 Results and Discussion

In this chapter, the three main results of the thesis are presented. Before the results themselves, section 6.1 presents the main characteristics of the test networks. Then, in section 6.2, a distinction between the two different modes of execution of the algorithm, sequential and parallel, is made, which is important for section 6.5. In section 6.3, the different metrics used in the results are explained.

Regarding the results, in section 6.4, the computational scalability of the algorithm is studied by means of the sequential version of the algorithm. In section 6.5 a comparison between the sequential and parallel execution of the algorithm is made in order to study the communication scalability. In the last section of the chapter, section 6.6, the main focus of the thesis is tackled: the effect of communication failures on the algorithm.

6.1 Description of the Test Networks

Five IEEE test networks of different sizes were used in this thesis: 14 node network, 30 node network, 57 node network, 118 node network and 300 node network. Additionally, a simple 5 node network case was used for quick code tests. The network parameters were extracted from [43]. As the systems are AC systems, the line parameters are given in terms of resistance and reactance. In some cases, zero resistance values are provided. In those cases, a resistance value is calculated by calculating the average r/x relationship for that network and multiplying this value by the reactances of the lines where no resistance is given. The schematics for the networks were found in [44].

Next, the main characteristics of each network are presented. For the remaining parameters not covered in this section such as resistance values, operational constraints and algorithm parameters, please refer to Appendix A. The nodal generation limits, generation costs and loading can be found in [43].

6.1.1 5 Node Network

Main characteristics	
Nodes	5
Lines	6
Generators	4
Generating capacity	15.3 p.u.
Loads	3
Total load	10 p.u.

Table 6.1: 5 node network main characteristics.

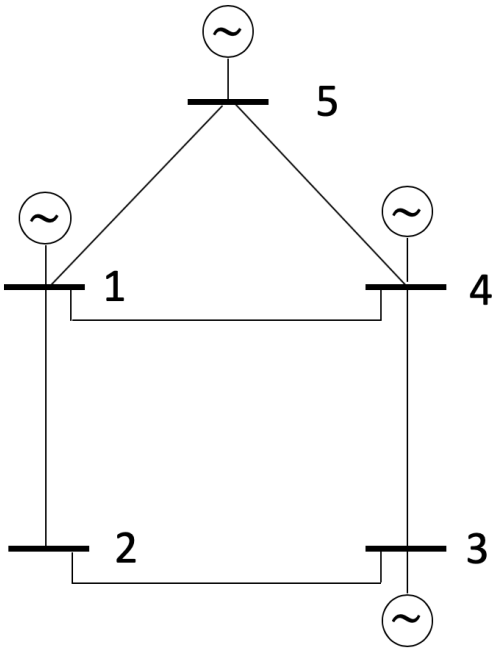


Figure 6.1: 5 node network.

6.1.2 14 Node Network

Main characteristics	
Nodes	14
Lines	20
Generators	5
Generating capacity	7.724 p.u.
Loads	11
Total load	2.59 p.u.

Table 6.2: 14 node network main characteristics.

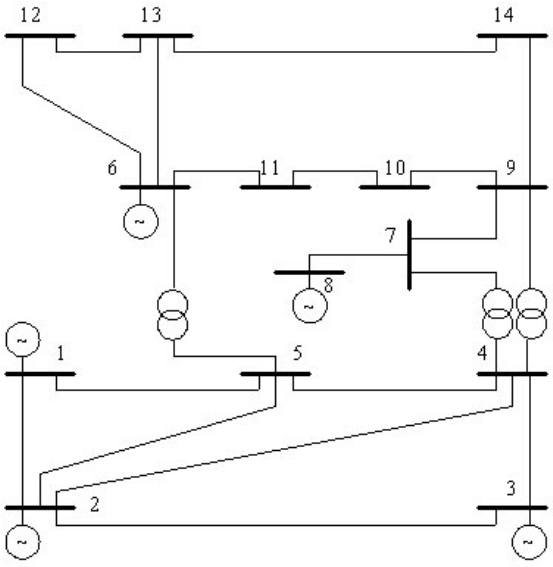


Рис.1. IEEE тестовая схема, состоящая из 14 узлов

Figure 6.2: 14 node network[44].

6.1.3 30 Node Network

Main characteristics	
Nodes	30
Lines	41
Generators	6
Generating capacity	9.002 p.u.
Loads	21
Total load	2.834 p.u.

Table 6.3: 30 node network main characteristics.

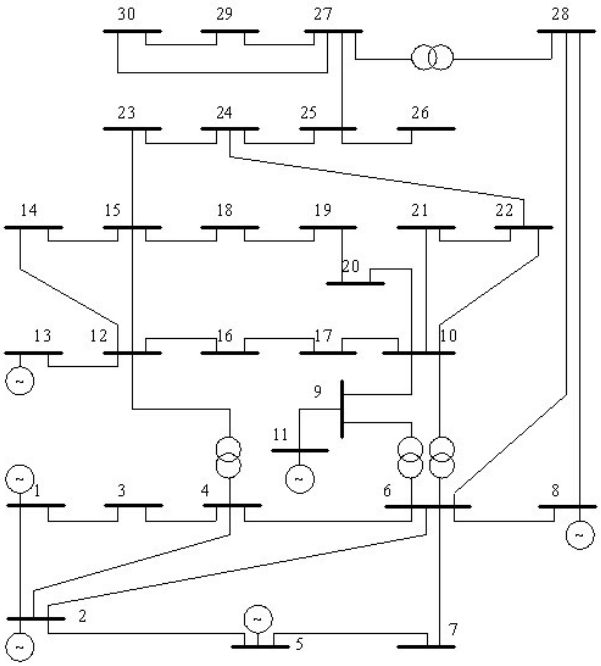


Рис.1. IEEE тестовая схема, состоящая из 30 узлов

Figure 6.3: 30 node network[44].

6.1.4 57 Node Network

Main characteristics	
Nodes	57
Lines	78
Generators	7
Generating capacity	19.7588 p.u.
Loads	42
Total load	12.508 p.u.

Table 6.4: 57 node network main characteristics.

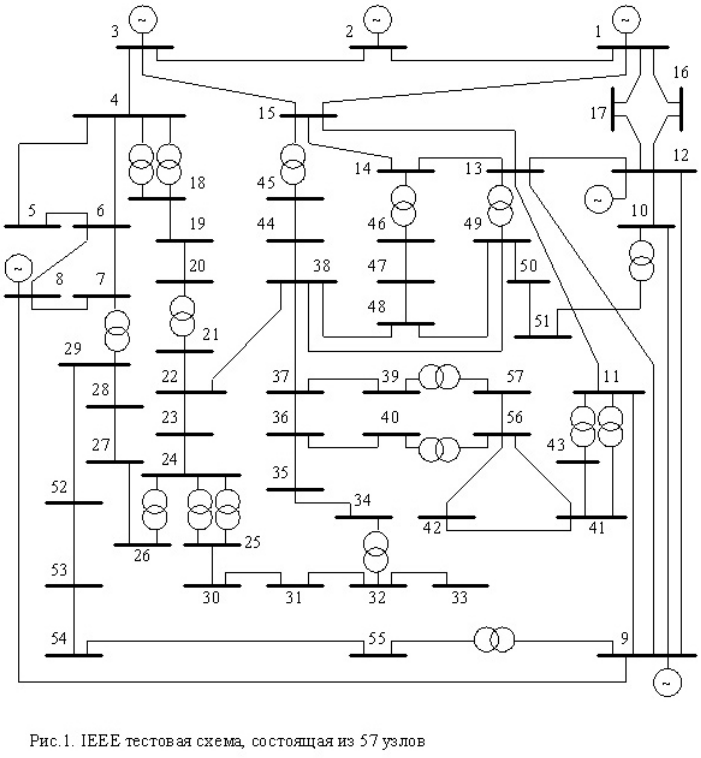


Figure 6.4: 57 node network[44].

6.1.5 118 Node Network

Main characteristics	
Nodes	118
Lines	179
Generators	54
Generating capacity	99.662 p.u.
Loads	99
Total load	42.42 p.u.

Table 6.5: 118 node network main characteristics.

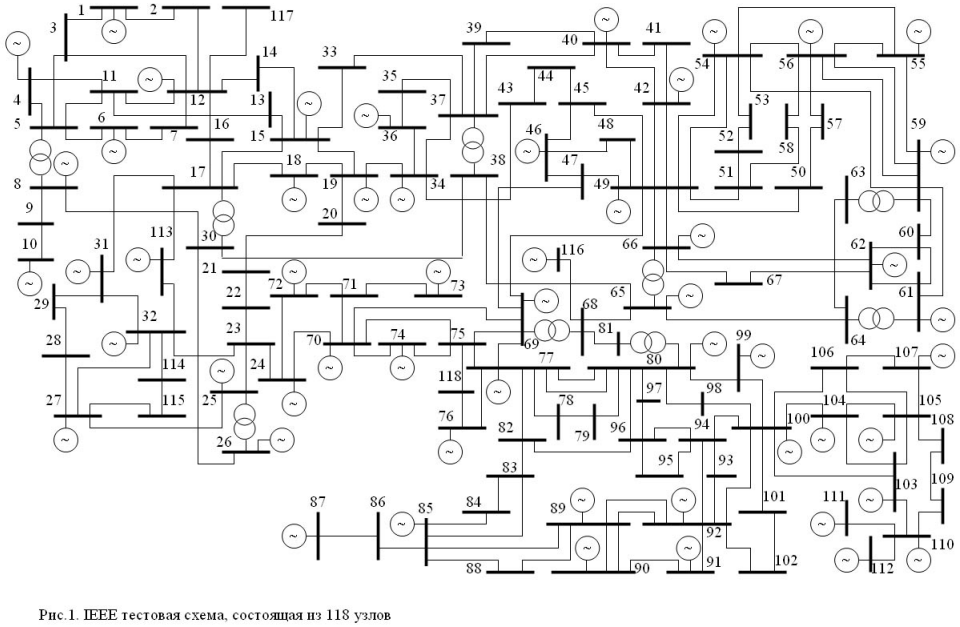


Figure 6.5: 118 node network[44].

6.1.6 300 Node Network

Main characteristics	
Nodes	300
Lines	409
Generators	69
Generating capacity	326.78435 p.u.
Loads	199
Total load	235.2585 p.u.

Table 6.6: 300 node network main characteristics.

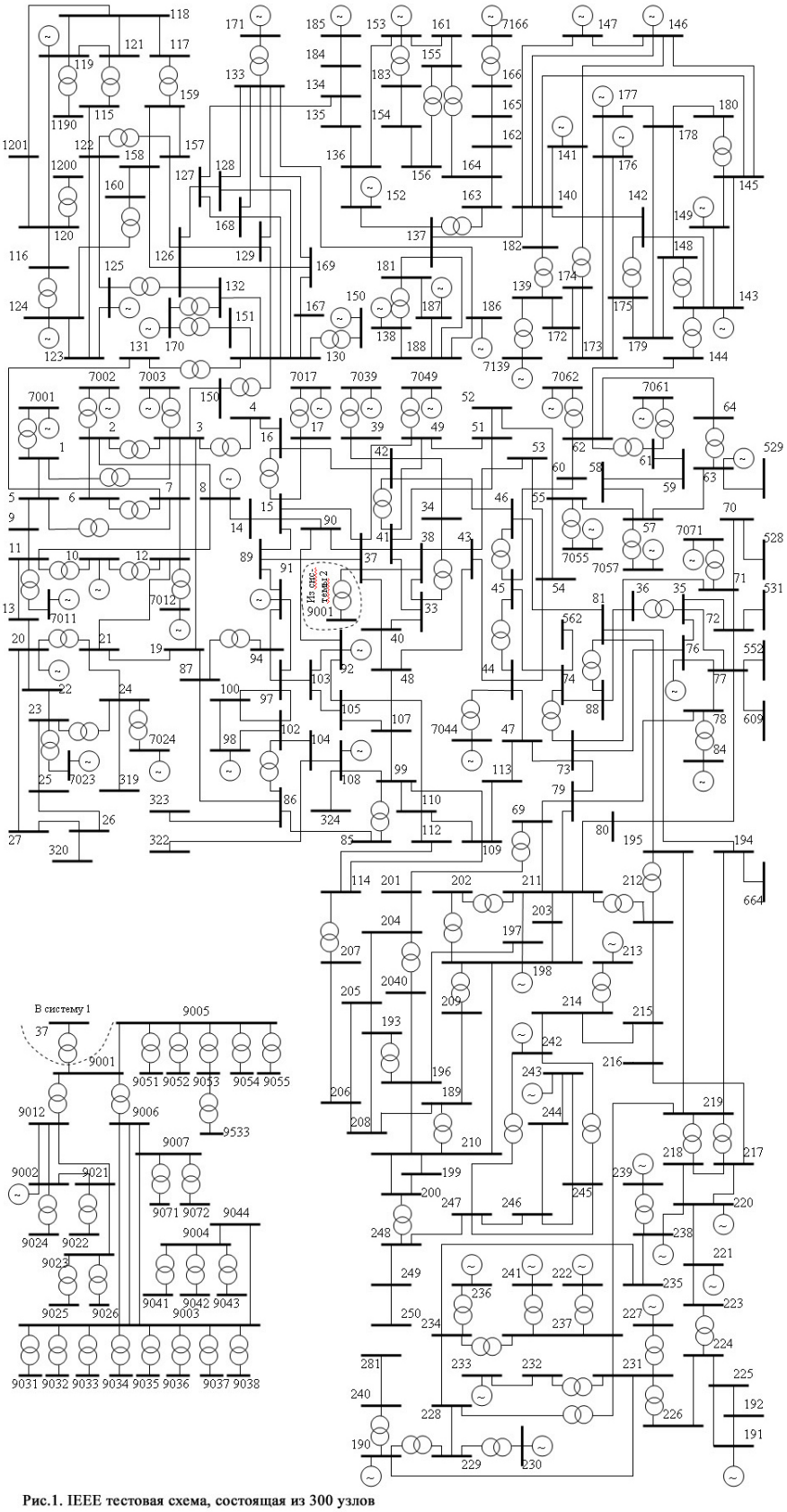


Figure 6.6: 300 node network[44].

6.2 Sequential vs. Parallel Execution

The PMP ADMM with SCA has the potential to be executed in a fully distributed way where all entities run the algorithm in parallel in different computers (or computation nodes). However, the algorithm can also run sequentially, where all entities take turns and run one after the other in one single computer. There are some essential differences between both types of execution.

Recuperating the 5 node example used in section 3.5, the sequential execution flowchart is as represented in figure 6.7.

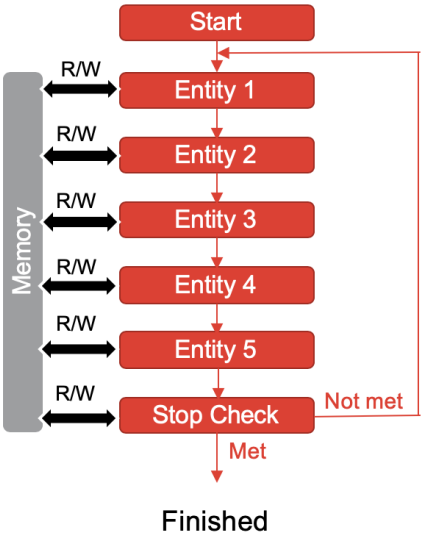


Figure 6.7: Sequential execution.

And the parallel execution flowachart is presented in figure 6.8.

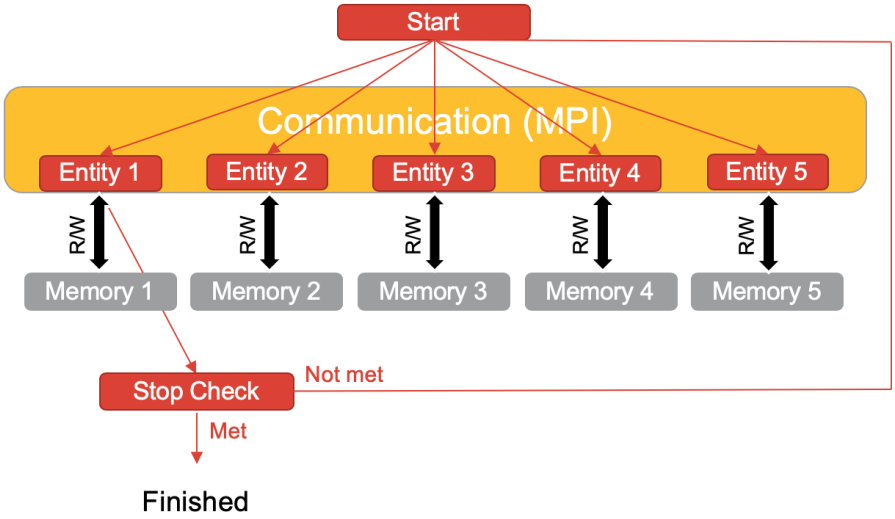


Figure 6.8: Parallel execution.

The evident difference between the two modes of execution is that in sequential mode (SM) only one entity at a time computes while in parallel mode, all entities can compute in parallel(PM). However, the fact that in SM the whole algorithm runs in one computer while in PM in (in principle) as many computers as entities have two main implications. The first one is that in SM, there is only one big memory shared by all the entities. This means that all entities can access the data of any other. The second implication is a consequence of the first one: there is no real communication between entities. Instead, the exchange of values is done by reading and writing to the common memory. In PM, each entity has its private memory no other entities have access to and, therefore, real communication must take place in order to exchange information. The relevance of these differences will become apparent in section 6.5. The communication when in parallel mode is carried out via MPI, using the *mpi4py* package introduced in section 5.1.1.

A sequential version of the algorithm was developed as a first stage to achieve the fully distributed parallel version.

6.3 Metrics

The different metrics that will be used to interpret the results are the following:

- To measure the **convergence speed**, given that the time to convergence is dependent on the hardware and software implementation, but the **number of iterations to convergence** is independent of this factors, the latter is chosen as a metric to compare convergence in different failure scenarios.
- To compare the algorithms results in different communication failures scenarios, the solution without failure is taken as reference. The **number of**

iterations till convergence, the **total power generated** and the **power and voltage profiles** of both, the solution in case of no failure and the solution in case of failure, are compared. Additionally, a comparison between the previous state of the system and the current solution can be made. In some failure cases, the algorithm may not be able to converge. To compare failure scenarios among each other, the **number of iterations required to convergence after the failure** and the **power losses deviation** with respect to the reference case are also calculated.

- On the timing side, three time measurements are relevant:

  Total execution time in seconds t_{total}, which is the total time the algorithm takes from start to finish. This is an important metric for each specific case but is not a useful metric to compare different cases, as the number of iterations to convergence for each system differs significantly, thus not giving a measure of the speed at which the entities are realising their tasks.

  Average time per iteration in seconds $t_{it} = \frac{t_{total}}{total\ iterations}$. Which is the average time the algorithm takes for each iteration. This is the best metric to compare sequential vs. parallel execution of the same system.

  Average time per iteration and entity $t_{it,ent} = \frac{t_{total}}{total\ iterations \cdot number\ of\ entities}$. Which is the average time each entity needs to finish one iteration. In parallel execution $t_{it} = t_{it,ent}$ as all entities are working in parallel. This is the best metric to compare differently sized systems against each other and extract scalability conclusions.

- Additionally, while timing the parallel algorithm, a distinction between **computation time** and **communication time** is made. Computation time t_{comp} is defined of all the time spent in mathematical operations and communication time t_{comm} all the tasks related with MPI (**Clone voltages exchange**, **Net voltages exchange**, **Failure flags gathering**, **Failure state broadcast**, **Voltage converge flags gathering**, **Voltage convergence broadcast**, **Power gathering** and **Stop flag broadcast**).

6.4 Computational Scalability Results

One of the main theoretical advantages of distributed algorithms is that they are scalable. When talking about distributed algorithms scalability can be divided in two categories: computational scalability and communication scalability. The first one refers to how much does the computation time increase when the problem size increases and the second how much does the communication requirements increase when the problem size increases. In order to measure the computational scalability, the sequential version of the algorithm was used, due to the lack of real communication in this algorithm form. A computationally scalable algorithm should show ideally a flat curve, where computation time does not grow at all with the size of

the problem. On the contrary, a non-scallable algorithm would show a cuadratic or exponential behavior.

The algorithm was run in the PC (referred as Mac) and in one of the computers of the SWARM HPC cluster (Swarm0). All networks were run and timed, and the total time, time per iteration and time per iteration and entity extracted. The results are presented in table 6.7.

Time metrics of the sequential algorithm [s]						
Network nodes	**5**	**14**	**30**	**57**	**118**	**300**
Iterations	162	120	1379	4004	544	5018
Sequential Mac	2.07	4.37	115	600	155	4772
Sequential HPC	1.16	2.8	80	446	135	4495
t_{it} Mac	0.0128	0.0364	0.0834	0.1499	0.2849	0.9510
t_{it} HPC	0.0072	0.0233	0.0580	0.1114	0.2482	0.8958
$t_{it,ent}$ Mac	0.0026	0.0026	0.0028	0.0026	0.0024	0.0032
$t_{it,ent}$ HPC	0.0014	0.0017	0.0019	0.0020	0.0021	0.0030

Table 6.7: Time metrics of all networks in PC and HPC for the sequential execution of the algorithm.

The first interesting result has to do with the number of iterations each network needed to converge. The 118 network needs almost 8 times fewer iterations than the 57 network and about half the iterations of network 30. The reason behind this is that one of the main factors that determine the number of iterations (next to the size of the problem itself) is the number of generators compared to the number of nodes a network has. The 30 node network has only 6 nodes with generation (20%) and the 57 node network only 7 (12%), while the 118 has 54 well-distributed generators (46%). The 300 node network goes back to 23% generation nodes. This iterations till convergence disparity makes the comparison between total execution times of little significance for our study, demonstrating an erratic trend as seen in figure 6.9.

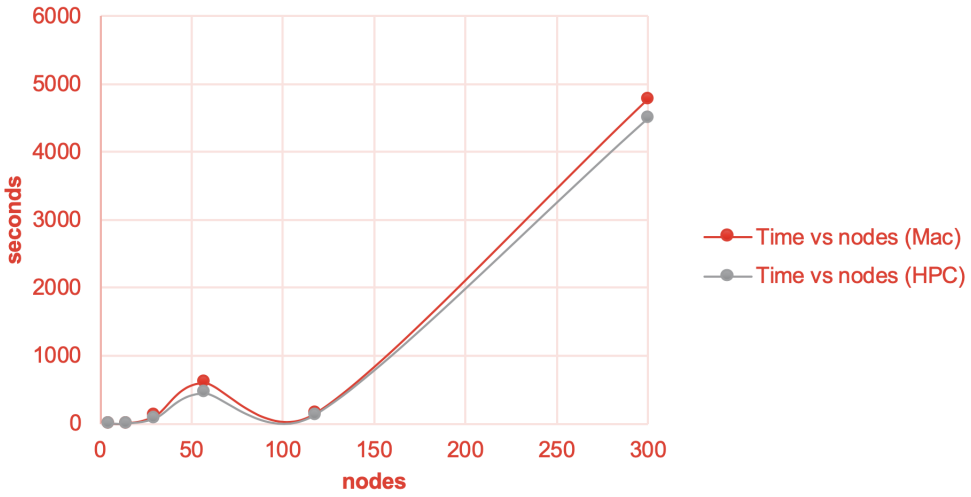


Figure 6.9: Total time for all networks in sequential execution.

However, when we calculate the time per iteration, a linear trend can be clearly seen in figure 6.10. Given that the algorithm is sequential, in each iteration, all the entities have to do their computations one after the other. Therefore, if the algorithm is computationaly scalable and each entity takes approximatly the same amount of time to solve one subproblem, regardless of how many other entities there are in the system, a linear increase of time per iteration is expected.

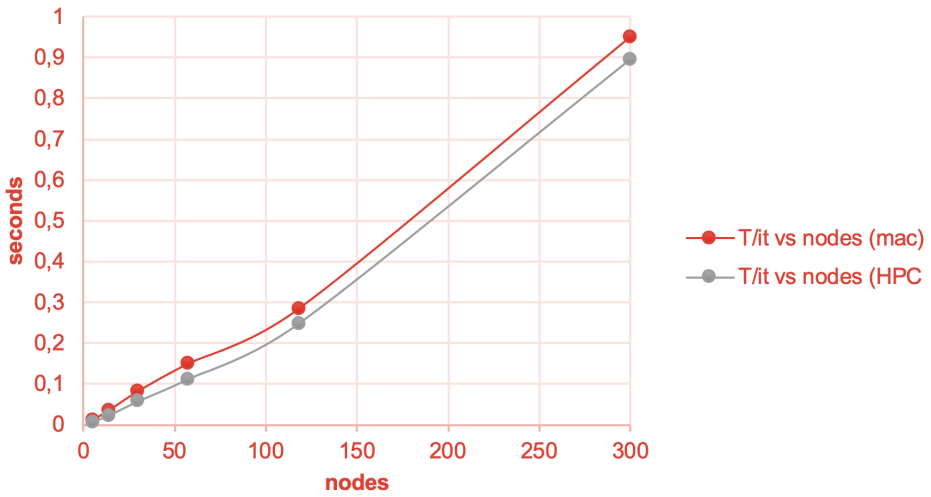


Figure 6.10: Time per iteration for all networks in sequential execution.

If we then divided the time per iteration also by the number of entities that are computing sequentially in each iteration we get the time each entity takes in each iteration to compute, i.e., the amount of time that each entity takes to solve the local

OPF. This is represented in figure 6.11, obtaining, as a result, a very flat profile. This proves that each entity takes arround 2.5ms to solve the local OPF regardless of if teh entity is in a 30 node system or a 300 node system. Therefore, we can conclude that the algorithm is, at least, computationally scalable.

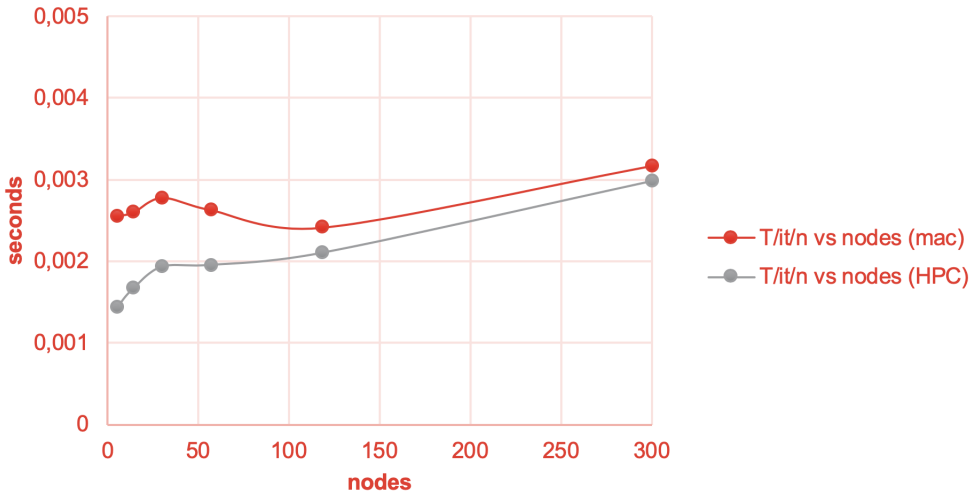


Figure 6.11: Time per iteration and number of entities for all networks in sequential execution.

6.5 Communication Scalability Results

In order to study the communication scalability of the algorithm, the parallel version of it is timed and compared to the sequential version.

In table 6.8, the different total execution times for networks 5 to 118 are presented. Several runs were timed and the average times calculated. In the PC it was not possible to launch 118 or 300 parallel processes due to hardware and software restrictions. In the HPC it was also not possible to luanch 118 processes in one single computing node and also not 300 processes, even if using the 4 computing nodes of the SWARM cluster.

Sequential vs. parallel execution total times [s]					
Network nodes	**5**	**14**	**30**	**57**	**118**
Iterations	162	120	1379	4004	544
Sequential Mac	2.07	4.37	115	600	155
Sequential HPC	1.16	2.8	80	446	135
Parallel Mac (2 cores)	1.4	3.21	82.5	471	NI
Parallel 1 HPCn (24 cores)	0.49	12.460	471.50	2652.5	NI
Parallel 2 HPCn (48 cores)	0.335	6.285	178.65	1352.0	510
Parallel 3 HPCn (72 cores)	0.345	2.815	129.47	797.0	368
Parallel 4 HPCn (96 cores)	0.335	1.300	107.61	598.0	255

Table 6.8: Sequntial vs. parallel execution times for all networks in PC and HPC using different amount of computing nodes.

The first striking result is that the sequential algorithm is generally faster than the parallel execution, registering the sequential algorithm executed in one of the nodes of the HPC cluster the best times in the bigger networks. The second is that the parallel execution in the PC (with 2 cores) is always faster than the parallel execution in one node of the HPC cluster (24 cores). The third is the fact that adding more computing nodes to the execution improves the execution time (with the exception of the 5 node network where 2,3 and 4 computing nodes yield the same time).

As explained in section 6.3, the total execution time is not a useful metric to compare different networks, as the number if iterations till convergence varies significantly. If we divide the total execution time by the number of iterations till convergence, we get the average time per iteration. These times are presented in table 6.9.

Sequential vs. parallel execution times per iteration [s]					
Network nodes	**5**	**14**	**30**	**57**	**118**
Iterations	162	120	1379	4004	544
Sequential Mac	0.0128	0.0364	0.0834	0.1499	0.2849
Sequential Swarm0	0.0072	0.0233	0.0580	0.1114	0.2482
Parallel Mac	0.0086	0.0268	0.0598	0.1176	NI
Parallel 1 HPCn (24 cores)	0.0030	0.1038	0.3419	0.6625	NI
Parallel 2 HPCn (48 cores)	0.0021	0.0524	0.1295	0.3377	0.9375
Parallel 3 HPCn (72 cores)	0.0021	0.0235	0.0939	0.1991	0.6765
Parallel 4 HPCn (96 cores)	0.0021	0.0108	0.0780	0.1494	0.4688

Table 6.9: Sequential vs. parallel execution times per iteration for all networks in PC and HPC using different amount of computing nodes.

A graphical representation of table 6.9 is given in figure 6.12.

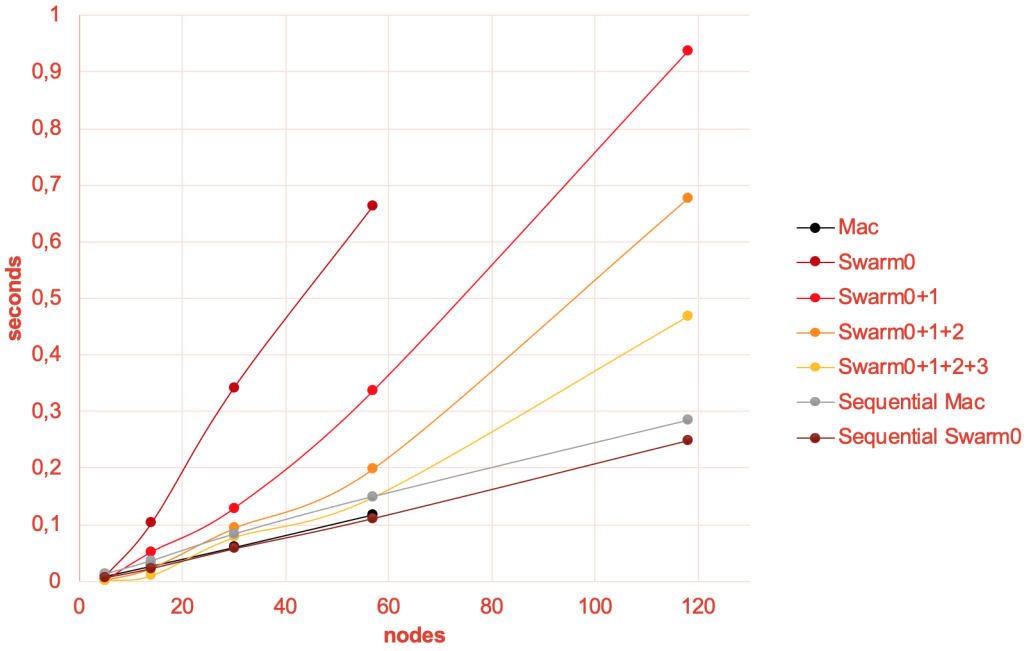


Figure 6.12: Sequential vs parallel execution times per iteration [s].

In 6.12 it can be clearly seen that the sequential version of the algorithm, where no real communication occurs, is more scalable than the parallel version (the slope is flatter). This shows that the problem is communication bound, which means that the communication time is higher than the computation time, to the point that solving all problems one after the other is more efficient than solving the problems in parallel and having them communicate with each other. In terms of speed, this sequential implementation is faster than the parallel one. However, a sequential implementation requires one computer to have all the data, which is not compatible with a distributed control strategy, going against the data privacy philosophy.

It can also be observed that the times in the HPC significantly improve when adding more computation nodes. This is a consequence of the problem being communication bound. In subsection 5.2.1 it was explained how three different communication paths go into play in the HPC cluster depending on where the cores that have to communicate are situated respect to one another. If the problem is communication bound, the speed of communication will significantly impact the total time.

MPI optimizes the allocation of processes and, the more degrees of freedom it has, the better it can optimize communication so that as much information as possible goes through the faster path (Infiniband). For example, focusing on the14 node network, if only one computational node of the HPC cluster is used (24 cores, 2 sockets), Infiniband is not an option and all communications are either shared memory access or inter-socket communication. Inter-socket communication is the slowest

one so MPI will try to avoid it and use as many cores of one socket as possible, so that the majority of the communication occurs via shared memory access, but 14 processes do not fit in one socket (at leat 2 have to be placed in another socket), meaning that the use of inter-socket communication is unavoidable. The memory manager of the computer may also be overloaded and not all 12 cores within one socket be able to access the shared memory in parallel (figure 6.13). If another computing node is added, MPI is able to use 7 cores of one machine and 7 of another and communicate them through InfiniBand, and avoid inter-socket communication. Still the communication between cores of the same socket will happen through shared memory access, but in this case, less occupied cores per socket also mean an improvement in the performance of the shared memory access (figure 6.14). Adding more computational nodes allow for more Infiniband communication and less shared memory acces, thus the continuous improvement in execution times till a point that the 14 node network with 3 and 4 computing nodes in the HPC runs faster as the PC and even slightly faster than the sequential algorithm(figures 6.15 and 6.16).

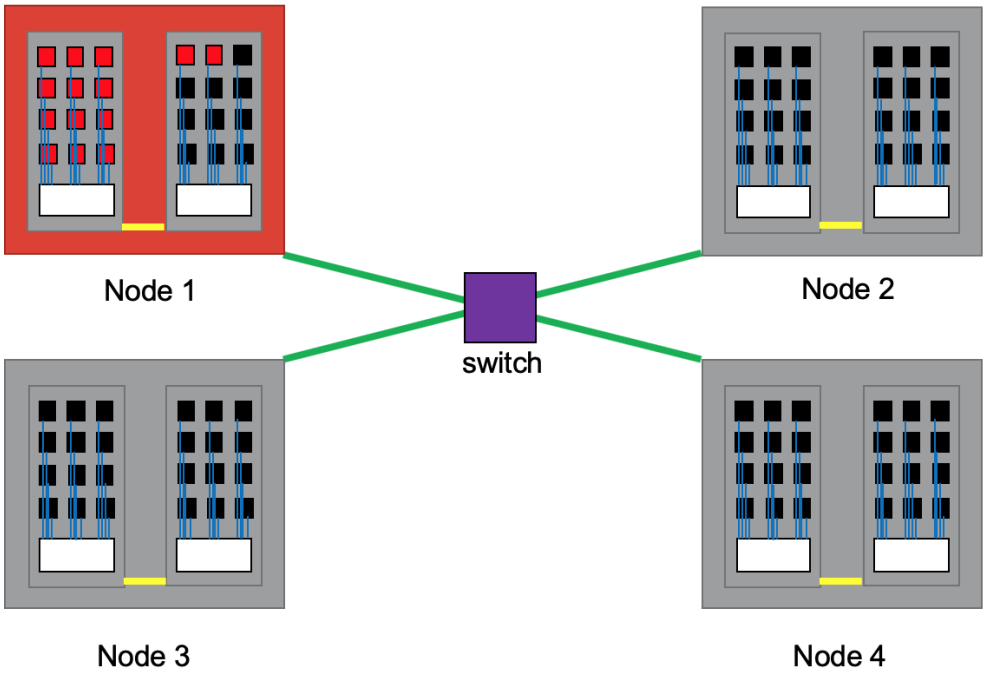


Figure 6.13: Example of allocation of 14 processes in 1 computing node.

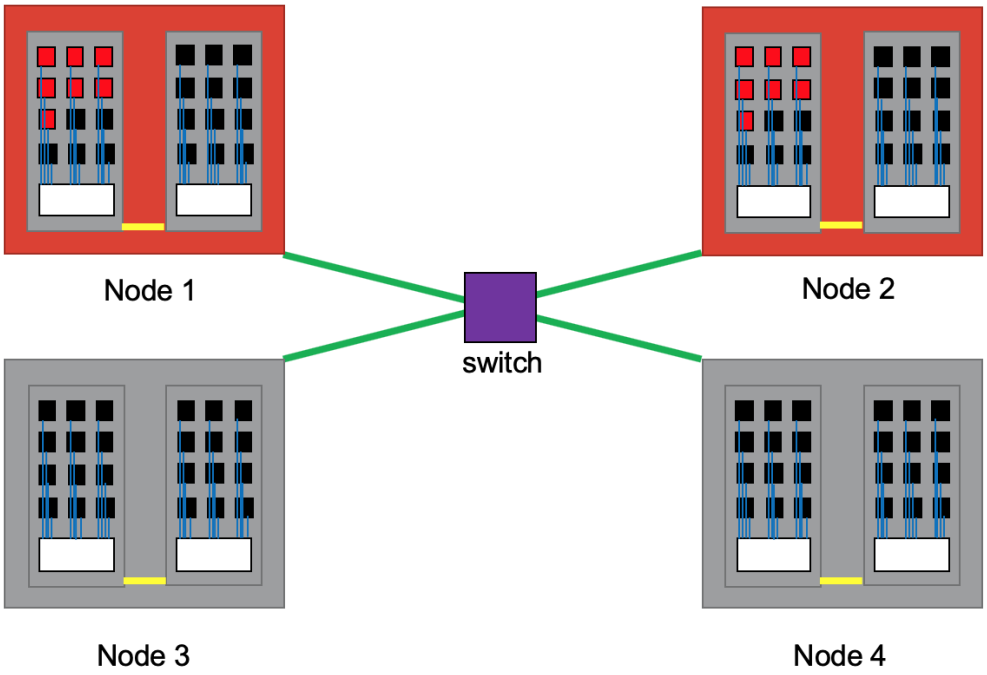


Figure 6.14: Example of allocation of 14 processes in 2 computing nodes.

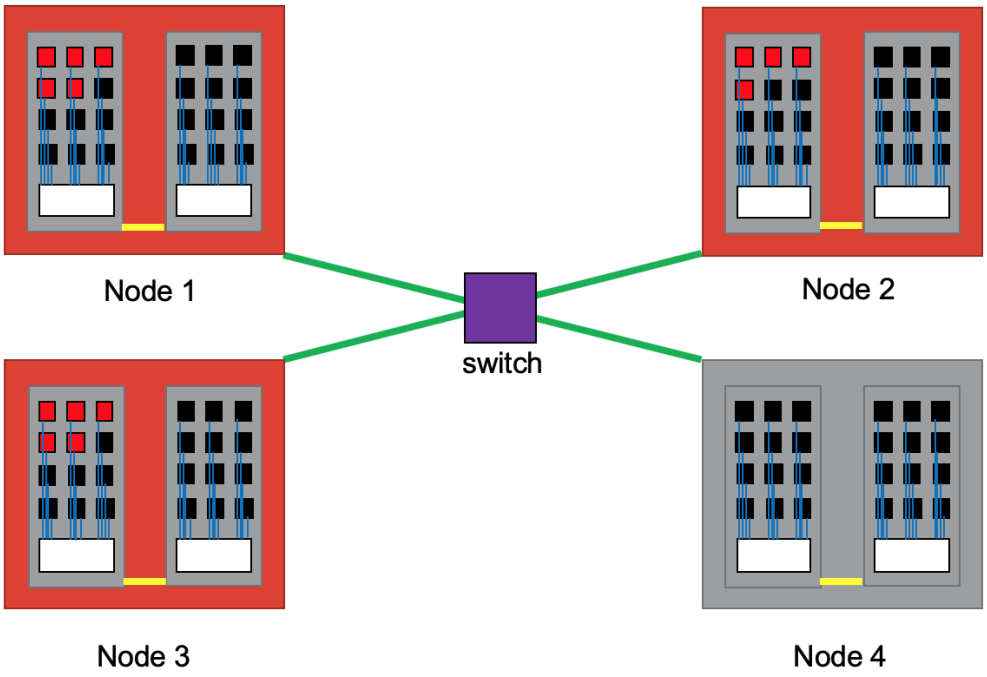


Figure 6.15: Example of allocation of 14 processes in 3 computing nodes.

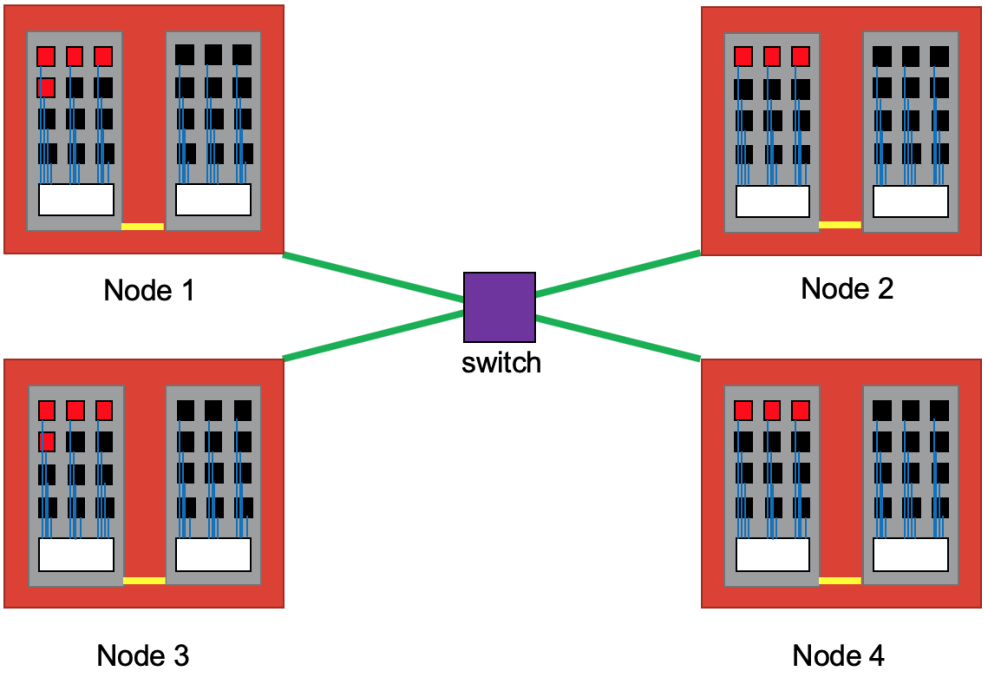


Figure 6.16: Example of allocation of 14 processes in 4 computing nodes.

In the 5 node network, there is a saturation of this effect, where more computing nodes do not improve the overall time due to the small number of processes. In this case, where communication is optimal, the parallel algorithm is faster than the sequential one. It can also be seen in figure 6.12 that the higher the number of computing nodes, i.e., the quicker and more consistent the communication technology is, the more the scalability of the parallel algorithm approaches the scalability of the sequential one. All this hints that, under ideal communication conditions, the parallel algorithm may be as scalable and even faster than the sequential counterpart. In the remaining networks, the trend of the 14 node network continues, but the times in the parallel algorithm in the PC and the sequential times are still faster.

The fact that the PC with only 2 cores but fast shared memory access manages better time than the HPC cluster only reasserts the fact that the problem is communication bound, leaving the enormous amount of overbooking in a 2 core machine with 57 processes aside as a negligible effect.

To confirm the claims above, the algorithm was timed differentiating between communication and computation time for various networks and the number of computing nodes. The communication time was also split into the different communication steps of the algorithms: Clone voltages exchange, Net voltages exchange, Failure flags gathering, Failure state broadcast, Voltage converge flags gathering, Voltage convergence broadcast, Power gathering and Stop flag broadcast. The times are **total times** and given in seconds.

Time of each communication task [s]					
14 node network	**Mac**	**1 HPCn**	**2 HPCn**	**3 HPCn**	**4 HPCn**
Clone voltages ex.	1.65	5.70	1.90	0.73	0.43
Net voltages ex.	0.061	1.78	0.35	0.034	0.0081
Failure flags gath.	0.36	5.59	2.75	1.48	0.97
Failure state bcast.	0.0034	0.0021	0.0017	0.0021	0.0016
Voltage flags gath.	0.12	3.23	2.37	0.78	0.020
Voltage conv. bcast.	0.016	0.0041	0.0035	0.0040	0.0030
Power gath.	0.00043	0.026	0.024	0.000025	0.000020
Stop flag bcast.	0.000025	0.000025	0.000094	0.000014	0.000011
Communication time	2.22	16.33	7.40	3.03	1.43
Computation time	1.06	0.69	0.16	0.13	0.13
Total time	3.28	17.02	7.56	3.16	1.56

Table 6.10: Time of each communicaiton task in the 14 node network case in PC and 1, 2, 3 and 4 computing nodes of the HPC.

Time of each communication task [s]					
30 node network	**Mac**	**1 HPCn**	**2 HPCn**	**3 HPCn**	**4 HPCn**
Clone voltages ex.	38.19	103.59	46.30	26.91	14.14
Net voltages ex.	1.75	8.80	12.54	10.66	7.62
Failure flags gath.	14.86	327.50	122.08	80.59	54.38
Failure state bcast.	0.041	0.025	0.031	0.046	0.031
Voltage flags gath.	4.36	52.78	40.28	49.87	28.81
Voltage conv. bcast.	0.10	0.042	0.043	0.060	0.049
Power gath.	0.0012	0.028	0.012	0.027	0.012
Stop flag bcast.	0.000027	0.000013	0.000021	0.000014	0.000013
Communication time	59.30	492.77	221.28	168.17	105.04
Computing time	30.57	51.94	15.71	4.77	3.19
Total time	89.87	544.71	236.99	172.94	108.23

Table 6.11: Time of each communicaiton task in the 30 node network case in PC and 1, 2, 3 and 4 computing nodes of the HPC.

Time of each communication task [s]					
57 node network	**Mac**	**1 HPCn**	**2 HPCn**	**3 HPCn**	**4 HPCn**
Clone voltages ex.	176.91	740.31	446.02	231.25	158.85
Net voltages ex.	31.54	172.46	99.51	69.12	78.68
Failure flags gath.	73.98	1204.03	490.01	296.13	259.37
Failure state bcast.	0.12	0.092	0.16	0.14	0.13
Voltage flags gath.	12.47	214.31	109.51	95.79	71.78
Voltage conv. bcast.	0.29	0.15	0.19	0.15	0.21
Power gath.	0.0026	0.069	0.025	0.0086	0.013
Stop flag bcast.	0.000031	0.000030	0.000015	0.000015	0.000014
Communication time	295.32	2331.43	1145.42	692.70	569.04
Computing time	266.47	712.03	349.23	242.3	183.16
Total time	561.79	3043.46	1494.66	935.01	752.20

Table 6.12: Time of each communicaiton task in the 57 node network case in PC and 1, 2, 3 and 4 computing nodes of the HPC.

Time of each communication task [s]					
118 node network	**Mac**	**1 HPCn**	**2 HPCn**	**3 HPCn**	**4 HPCn**
Clone voltages ex.			88.39	92.11	57.47
Net voltages ex.			7.66	4.01	2.14
Failure flags gath.			310.76	212.48	139.02
Failure state bcast.			0.016	0.029	0.011
Voltage flags gath.			51.38	37.44	30.27
Voltage conv. bcast.			0.025	0.025	0.017
Power gath.			0.067	0.10	0.064
Stop flag bcast.			0.000024	0.000029	0.000015
Communication time			458.30	346.28	228.99
Computing time			48.98	27.09	25.77
Total time			507.28	373.37	254.77

Table 6.13: Time of each communicaiton task in the 118 node network case in PC and 1, 2, 3 and 4 computing nodes of the HPC.

As we suspected, the communication time with respect to the computation time is much longer, especially in the HPC. We also learn from these results that the most demanding communication steps are the clone voltages exchange, the failure flags gathering and the voltage flags gathering. These are the most burdensome communication tasks and the ones happening at every iteration. The communication steps involving the power criterion check, which occurs only if there is voltage convergence, have a negligible impact on the total communication time. The broad-

casting actions are also less time consuming than the gathering actions. This gives insight into what steps should be prioritized in order to optimize the overall time.

In conclusion, the parallel algorithm showed some of the challenges of real communication and gave insight into what the most demanding communication tasks are. In this section, we also learned that the problem is communication bound, so much so that, with our implementation, the sequential version is actually faster and more scalable. The results also point in the direction that, with a consistent, homogeneous and fast communication technology such as 5G, the parallel algorithm could become as scalable and fast, if not more scalable and faster, than the sequential algorithm.

6.6 Communication Failure Results

This section will deal with the communication failures. First, in subsection 6.6.1, the objectives of the simulations are enumerated, i.e., what factors we aim to investigate through the simulations. Second, in subsection 6.6.2, the different test cases will be specified and explained. In subsection 6.6.3, the output plots of the simulation tools are presented and explained. Lastly, in subsection 6.6.4 the results are presented and discussed.

6.6.1 Influencing Factors

The aim of the simulation cases presented in subsection 6.6.2 is to test the effectiveness of the communication failure detection and mitigation strategy.

Additionally, we also want to investigate which of the following factors have a greater influence on the convergence of the algorithm under failure with the implemented mitigation strategy.

- Iteration at which the failure happens.
- Size of the generator of the entity that fails.
- Number of neighbors of the entity that fails.
- Loading of the previous state of the system.
- Number of entities/links that fail.

We expect that other more difficult to isolate factors, such as the topology and size of the grid, the number, distribution and size of the generators, and the loading of the current state of the system, also influence the convergence. Even if each individual grid will have particular characteristics, the aim of these simulations is to find out if there are general trends that can be separated from the peculiarities of each network.

6.6.2 Test Cases

In this section, the test cases design to test the effectiveness of the communication failure detection and mitigation strategy and the factors introduced in subsection 6.6.1 are described. The 30, 57 and 118 networks were used. The 5 and 14 node networks were discarded due to their small size and the 300 node network due to hardware limitations.

In first place, the following **single entity failures** were simulated:

- Single entity failure with the maximum number of neighbors.
- Single entity failure with the minimum number of neighbors.
- Single entity failure with average number of neighbors.
- Single entity failure with the biggest generator.
- Single entity failure with the smallest generator.

Regarding the iteration at which the failure occurs, two subcases were simulated: one at the beginning of the algorithm and one at the end. The iterations till convergence of the algorithm without failure for each network were taken as reference. For example, the 30 node network needs 1379 iterations without failure. The failures are simulated at iteration 100 and at iteration 1200 (approximately in the first and last tenth of the iterations).

This leaves a total of 30 cases, summed up in table 6.14.

The goal of these cases was to detect if the characteristics of the entity that fails (generation or number of neighbors)or the moment of the failure (at the beginning or at the end) have any particular effect on the convergence of the algorithm.

Second, **multiple entities failures** were tested. The 30 node network and the 118 node network were used. The 57 network, due to the small number of generators, has more convergence problems and was discarded from these tests. The failures were tested again at the beginning and at the end of the algorithm. A list of the cases tested is presented in table 6.15.

Additionally, the 30 node network was used to study the effect of the previous state of the system on the convergence of the algorithm under failure. In all the previous cases a previous state of the system with a load equal to 90% of the current load was assumed. To asses the effect of the previous state of the system, all 30 node network cases already mentioned were repeated assuming the previous load of the system was 50% the current one and 150% the current one.

This makes a total of 75 different simulations of entity failures desgined to explore all the factors that affect the convergence of the algorithm under communication failures.

On the **link failures** side, they are less critical (entity failures can be seen as the failure of all links of an entity) and less likely to happen in a wireless communication

context. Therefore, a series of simulations were carried out in the 30 node network, where the number of links that fail increases until convergence is not achieved. These cases are summed up in table 6.16.

Single entity failures		
Network	**Failed entity**	**Iteration**
30 node network	6 (max. Neighbours (7))	100
	2,4,7,8,9,10,28	1200
	11 (min. Neighbours (1))	100
	9	1200
	28 (avrg. Neighbours (3))	100
	6,8,27	1200
	1 (biggest generator (3.602p.u.))	100
		1200
	5 (smallest generator (1p.u.))	100
		1200
57 node network	9 (max. Neighbours (6))	400
	8,10,11,12,13,55	3600
	33 (min. Neighbours (1))	400
	32	3600
	48 (avrg. Neighbours (3))	400
	38,47,49	3600
	1 (biggest generator (5.76p.u.))	400
		3600
	2 (smallest generator (1p.u.))	400
		3600
118 node network	49 (max. Neighbours (9))	50
	42,45,47,48,50,51,54,66,69	450
	10 (min. Neighbours (1))	50
	9	450
	11 (avrg. Neighbours (4))	50
	4,5,12,13	450
	69 (biggest generator (8.052p.u.))	50
		450
	1 (smallest generator (1p.u.))	50
		450

Table 6.14: Single entity failure cases.

Multiple entities failures			
Network	**Failed entities**	**Iteration**	**Case name**
30 node network	6,28 (2)	100,1200	MEF1-30
	6,28,1 (3)	100,1200	MEF2-30
	15,12,10,6,8 (5)	100,1200	MEF3-30
	14,15,18,19,20 (5)	100,1200	MEF4-30
	3,14,16,18,19 (5)	100,1200	MEF5-30
	3,4,6,7,9,12,28 (7)	100,1200	MEF6-30
118 node network	49,10,11,69,1 (5)	50,450	MEF1-118
	8,17,20,34,47,49,62,63 (8)	50,450	MEF2-118
	1,10,20,30,40,50,60,70,80,90,100,110 (12)	50,450	MEF3-118
	1-35 and 89-118 (65)	50,450	MEF4-118
	1-59 (59)	50,450	MEF5-118
	1-89 (89)	50,450	MEF6-118

Table 6.15: Multiple entities failure cases.

Link failures			
Network	**Failed links**	**Iteration**	**Case name**
30 node network	1-3 (1)	500	LF1
	1-3,5-7 (2)	500	LF2
	1-3,5-7,2-4,-2-6 (4)	500	LF3
	11-3,5-7,2-4,2-6,8-6,8-28 (6)	500	LF4
	1-3,5-7,2-4,2-6,8-6,8-28,11-9 (7)	500	LF5
	1-3,5-7,2-4,2-6,8-6,8-28,11-9,13-12 (8)	500	LF6

Table 6.16: Link failures cases.

6.6.3 Simulation Plots

Each simulation gives as a result, apart from the raw numerical data, 8 different plots.

An exmaple from a simulation of the 14 node network, were entities 4 and 8 fail in iteration 100 is presented:

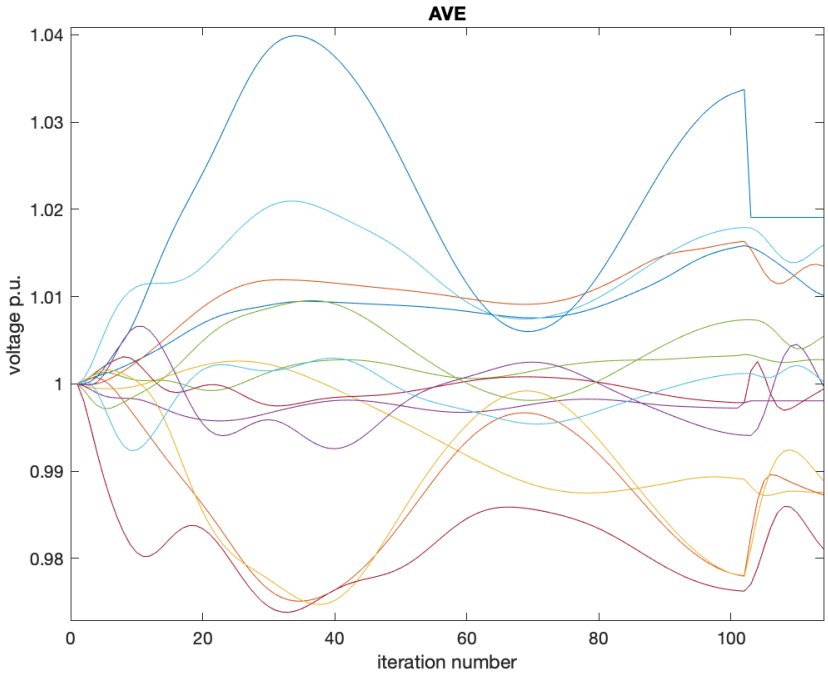


Figure 6.17: Evolution of all net voltages through the iterations of the algorithm.

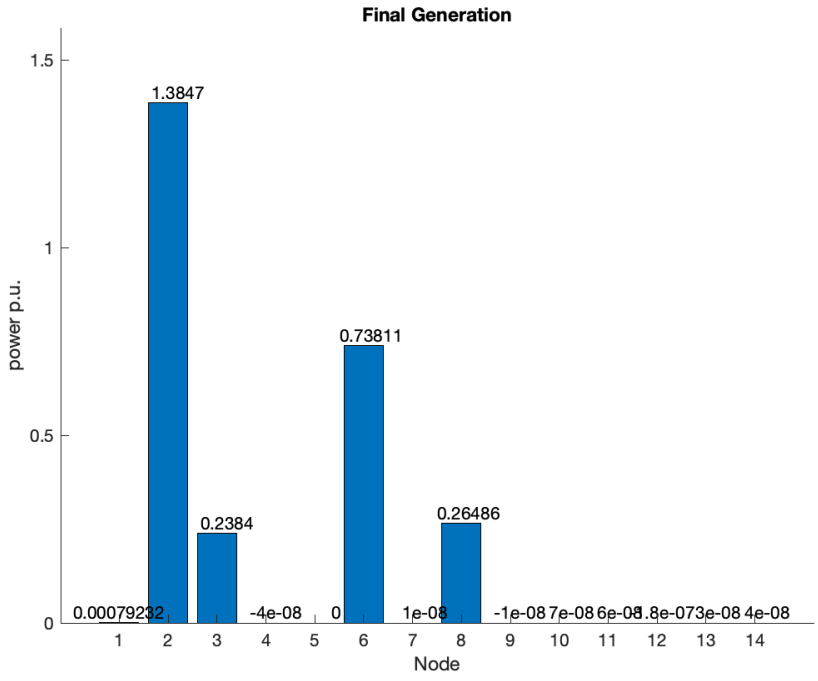


Figure 6.18: Power generation per node.

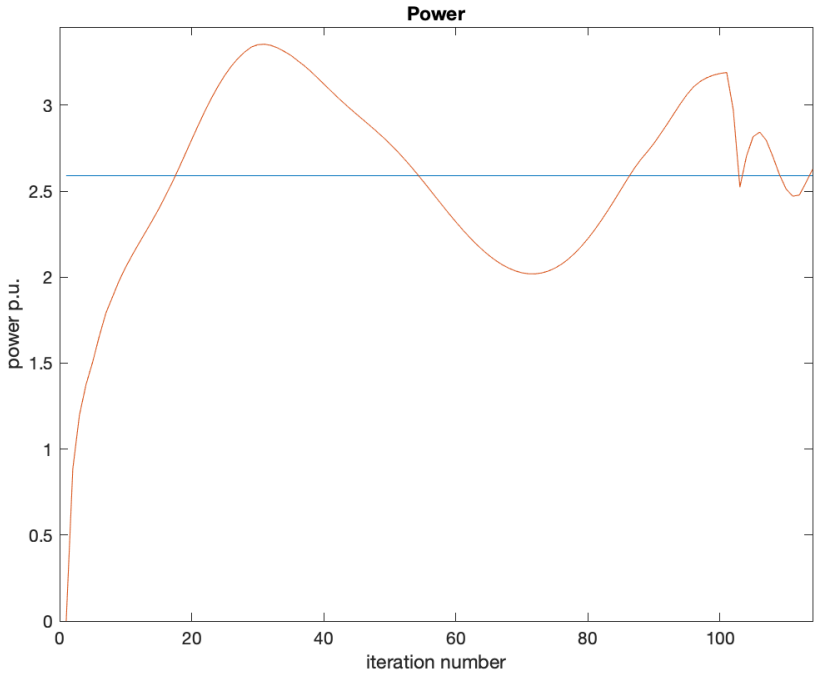


Figure 6.19: Total power generated in the system in each iteration and total demand.

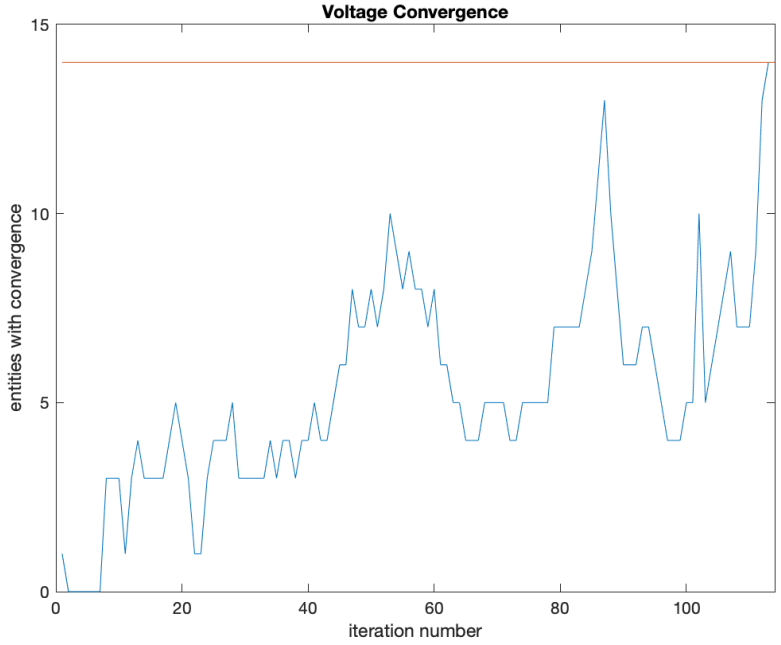


Figure 6.20: Amount of entities with voltage convergence with respect to the total number of entities in each iteration.

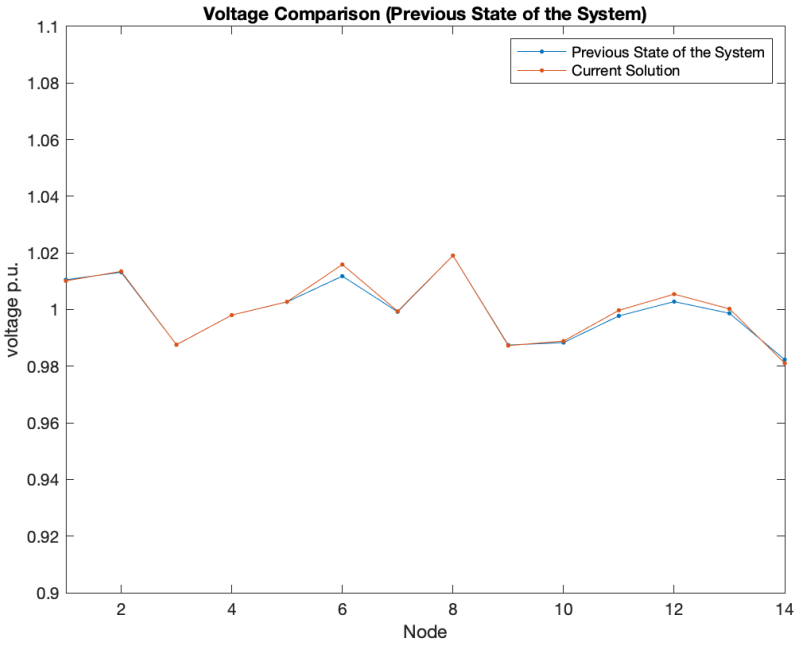


Figure 6.21: Voltage profile comparison between previous state of the system and solution with failure.

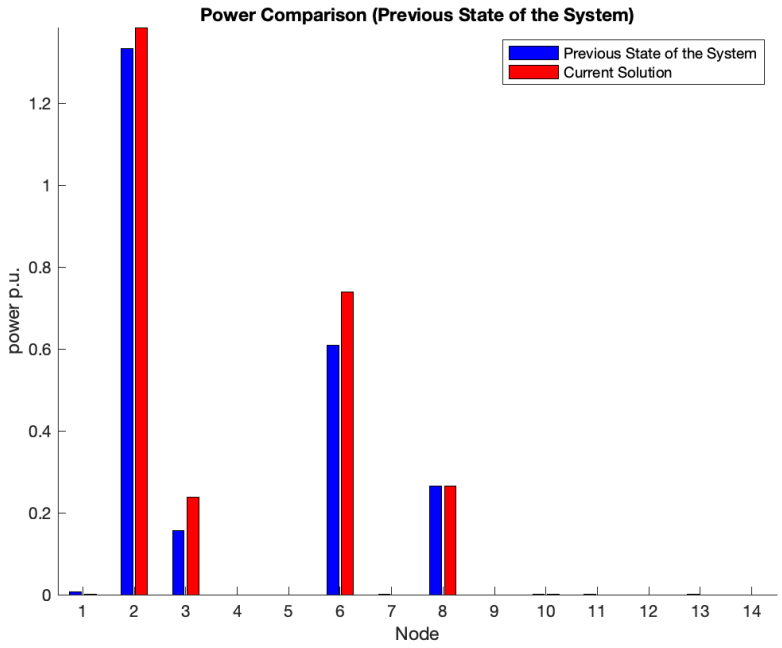


Figure 6.22: Nodal power generation comparison between previous state of the system and solution with failure.

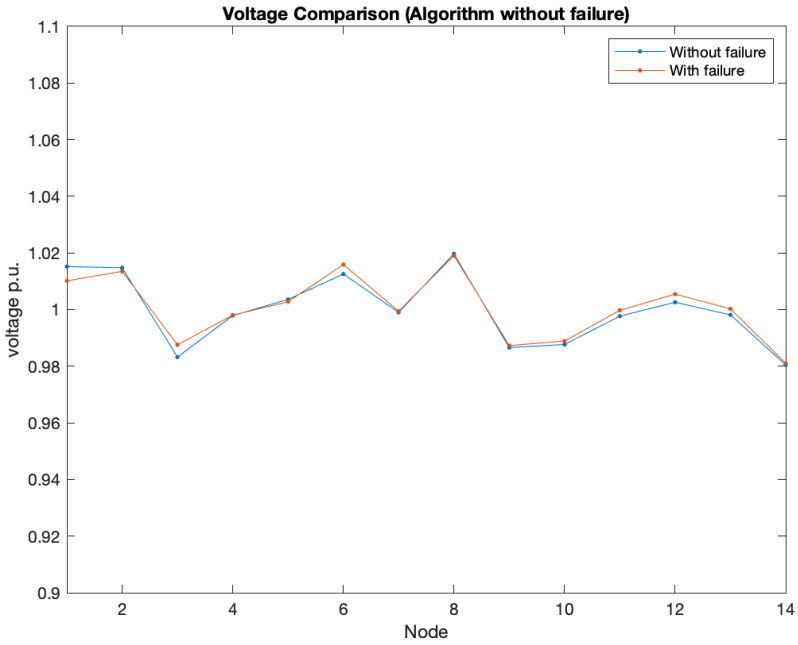


Figure 6.23: Voltage profile comparison between solution without failure and solution with failure.

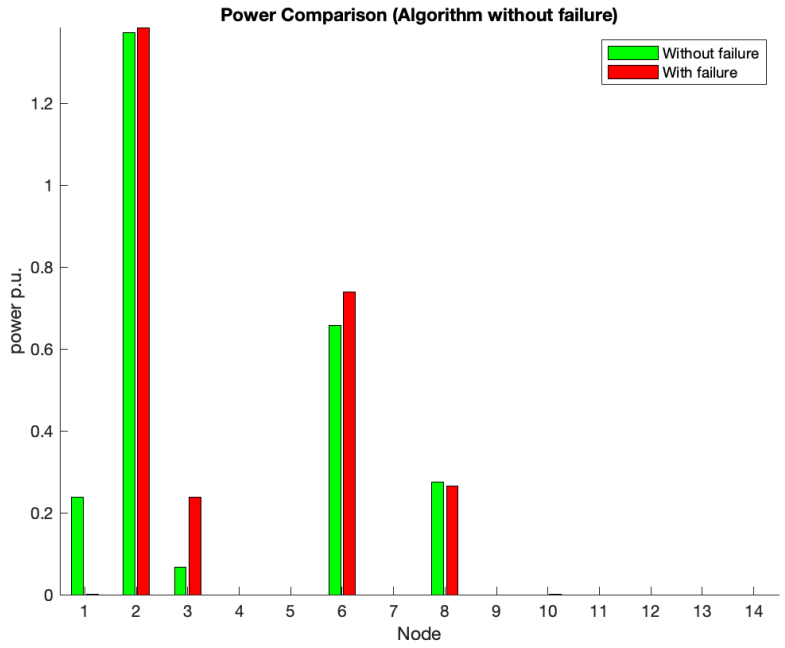


Figure 6.24: Nodal power generation comparison between solution without failure and solution with failure.

- Figure 6.17 shows the evolution of the net voltages with respect to the iterations of the algorithm. The two very pronounced steps in two of the signals correspond to the entities that fail and fix their voltage to the previous state voltage. How this affects the other entities is also noticeable in the signals and the closer the entities are to the failed ones the more significant the disturbance is.

- Figure 6.18 shows the final generation in each node. The values close to zero are not exactly zero due to numerical tolerances in the algorithm.

- Figure 6.19 shows the porgresion of the total power generation with respect to the iterations (in orange). It also displays the total demand for reference (straight blue line). The disturbance caused by the failure in this plot is also very noticeable.

- Figure 6.20 shows the number of entities that have reached voltage convergence at each given iteration (in blue) and gives the total amount of entities as reference (orange line).

- Figure 6.21 compares the final voltage profiles (voltage per node) of the previous state of the system and the current one (with failure). The behavior of the algorithm against failures can be clearly seen in this plot; entities 4 and 8 failed, therefore, voltages in these entities have the same values as in the previous state of the system as well as the voltages of the neighbors of those entities (entities 2,3,5,7,9). The other entities are free to adjust their voltages so that the demand is met.

- In a similar fashion, figure 6.22 compares the power generated per node in the previous state of the system and in the current solution of the algorithm with failure. Entity 4 has no generator, so the generation stays at 0 but entity 8 generates precisely the same amount as in the previous state of the system. The rest of the generators increase their generation to meet the demand.

- Parallel to figures 6.21 and 6.22, a comparison of voltage and generation profiles is made in figures 6.23 and 6.24, but this time against the voltage and generation of the solution of the algorithm if there was no communication failure. This way, the deviation due to the failure is graphically represented.

6.6.4 Results and Discussion

The 81 cases simulated give a total of more than 600 plots. For practical reasons, tables will be used to synthesize the main results and discuss them.

Single Entity Failure

Single entity failure results - I					
30 node network	**Reference**	**Biggest generator**		**Smallest generator**	
Iteration of failure	-	100	1200	100	1200
Entity failed	-	1		5	
Neighbours	-	(2)		(2)	
Generation (p.u.)	-	3.602		1	
Voltage convergence	yes	yes	yes	yes	yes
Power convergence	yes	yes	yes	yes	yes
Total generation (p.u.)	2.97	3.11	3.109	3.109	3.117
Total demand (p.u.)	2.83				
Power losses deviation (p.u.)	-	0.14	0.139	0.139	0.147
Max iterations	3000				
Iterations till convergence	1379	197	1223	193	1229
Iterations after failure	-	97	23	93	29

Table 6.17: Simulation results of single entity failure in Network 30 (generator size).

Single entity failure results - II						
30 node network	**Max. neighbours**		**Min. neighbours**		**Avrg. neighbours**	
Iteration of failure	100	1200	100	1200	100	1200
Entity failed	6		11		28	
Neighbours	(7)		(1)		(3)	
Generation (p.u.)	0		1		0	
Voltage convergence	yes	yes	yes	yes	yes	yes
Power convergence	yes	yes	yes	yes	yes	yes
Total generation (p.u.)	3.105	3.008	3.107	3.051	2.85	3.096
Total demand (p.u.)	2.83					
Power losses deviation (p.u.)	0.135	0.038	0.137	0.081	-0.12	0.126
Max iterations	3000					
Iterations till convergence	162	1221	210	1206	296	1222
Iterations after failure	62	21	110	6	196	22

Table 6.18: Simulation results of single entity failure in Network 30 (number of neighbors).

Single entity failure results - III					
57 node network	**Reference**	**Biggest generator**		**Smallest generator**	
Iteration of failure	-	400	3600	400	3600
Entity failed	-	1		2	
Neighbours	-	(4)		(2)	
Generation (p.u.)	-	5.76		1	
Voltage convergence	yes	yes	yes	yes	yes
Power convergence	yes	yes	yes	yes	yes
Total generation (p.u.)	12.78	13.03	12.59	13.71	12.74
Total demand (p.u.)	12.51				
Power losses deviation (p.u.)		0.25	-0.19	0.93	-0.04
Max iterations	8000				
Iterations till convergence	4004	411	3627	428	3619
Iterations after failure	-	11	27	28	19

Table 6.19: Simulation results of single entity failure in Network 57 (generator size).

Single entity failure results - IV						
57 node network	**Max. neighbours**		**Min. neighbours**		**Avrg. neighbours**	
Iteration of failure	400	3600	400	3600	400	3600
Entity failed	9		33		48	
Neighbours	(6)		(1)		(3)	
Generation (p.u.)	1		0		0	
Voltage convergence	yes	yes	yes	yes	yes	yes
Power convergence	yes	yes	yes	yes	yes	yes
Total generation (p.u.)	13.64	12.67	12.51	12.7	12.95	13.41
Total demand (p.u.)	12.51					
Power losses deviation (p.u.)	0.86	-0.11	-0.27	-0.08	0.17	0.63
Max iterations	8000					
Iterations till convergence	413	3620	681	3601	584	3617
Iterations after failure	13	20	281	1	184	17

Table 6.20: Simulation results of single entity failure in Network 57 (number of neighbors).

Single entity failure results - V					
118 node network	**Reference**	**Biggest generator**		**Smallest generator**	
Iteration of failure	-	50	450	50	450
Entity failed	-	69		1	
Neighbours	-	(6)		(2)	
Generation (p.u.)	-	8.052		1	
Voltage convergence	yes	yes	yes	yes	yes
Power convergence	yes	yes	yes	yes	yes
Total generation (p.u.)	43.09	42.59	42.59	42.52	43.89
Total demand (p.u.)	42.42				
Power losses deviation (p.u.)		-0.5	-0.5	-0.57	0.8
Max iterations	1500				
Iterations till convergence	544	225	453	171	453
Iterations after failure	-	175	3	121	3

Table 6.21: Simulation results of single entity failure in Network 118 (generator size).

Single entity failure results- VI						
118 node network	**Max. neighbours**		**Min. neighbours**		**Avrg. neighbours**	
Iteration of failure	50	450	50	450	50	450
Entity failed	49		10		11	
Neighbours	(9)		(1)		(4)	
Generation (p.u.)	3.04		5.5		0	
Voltage convergence	yes	yes	yes	yes	yes	yes
Power convergence	yes	yes	yes	yes	yes	yes
Total generation (p.u.)	42.55	43.99	42.48	43.3	42.44	44.03
Total demand (p.u.)	42.42					
Power losses dev. (p.u.)	-0.54	0.9	-0.61	0.21	-0.65	0.94
Max iterations	1500					
Iterations till converg.	212	451	175	454	177	451
Iterations after failure	162	1	125	4	127	1

Table 6.22: Simulation results of single entity failure in Network 118 (number of neighbors).

All 30 single entity failures converge to a feasible solution. In all cases, the algorithm finishes faster as the algorithm without failure thanks to the voltage convergence criterion relaxation, achieving our goal of a fast emergency solution. In the 30 and 118 Network in all cases the algorithm needs noticably more iterations after the failure when the failure happens at the beginning compared to when the failure happens at the end. This also applies to the "min. neighbors" and "avg. neighbors" case of the 57 Network, but is contradicted by the other three cases. Regarding the extra losses of the solutions with failures, the 30 Network tends to give slighlty more

lossy results, while in the 57 and 118 Networks the results are mixed, sometimes being the solution more lossy than the solution with no failure and sometimes less. There is no discernible pattern regarding the size of the generators or the number of neighbors. In 30 Network the "min." and "avg. neighbors" are the cases that need more iterations after failure, and the biggest and smallest generator cases need around the same iterations. The same is also true for the 57 Network, while in the 118 Network the biggest generator and the max neighbors are the ones that need more iterations after failure.

The overall conclusion is that none of the factors seem to have a particular influence in the convergence and no general trend can be seen across all the networks, apart from the fact that if the failure occurs at the beginning the convergence takes longer, which is to be expected as the algorithm is at the first stages of convergence at that point. None of the failures represent a threat to the system as in every case the algorithm converges even before as it would typically do.

Effect of the Loading Condition of the System

Load of previous state: 50% of current load- I					
30 node network	**Reference**	**Biggest generator**		**Smallest generator**	
Iteration of failure	-	100	1200	100	1200
Entity failed	-	1		5	
Neighbours	-	2,3(2)		2,7(2)	
Generation (p.u.)	-	3.602		1	
Voltage convergence	yes	no	no	no	no
Power convergence	yes	-	-	-	-
Total generation (p.u.)	2.97	-	-	-	-
Total demand (p.u.)	2.83				
Power losses deviation (p.u.)	-	-	-	-	-
Max iterations	3000				
Iterations till convergence	1379	3000	3000	3000	3000
Iterations after failure	-	-	-	-	-

Table 6.23: Simulation results in case of 50% of the current load in the previous system state in Network 30 (generator size).

Load of previous state: 50% of current load- II						
30 node network	**Max. neighbours**		**Min. neighbours**		**Avrg. neighbours**	
Iteration of failure	100	1200	100	1200	100	1200
Entity failed	6		11		28	
Neighbours	2,4,7,8,9,10,28(7)		9(1)		6,8,27(3)	
Generation (p.u.)	0		1		0	
Voltage convergence	no	no	yes	yes	no	no
Power convergence	-	-	yes	yes	-	-
Total generation (p.u.)	-	-	3.08	3.11	-	-
Total demand (p.u.)	2.83					
Power losses dev. (p.u.)	-	-	0.11	0.14	-	-
Max iterations	3000					
Iterations till converg.	3000	3000	232	1263	3000	3000
Iterations after failure	-	-	132	63	-	-

Table 6.24: Simulation results in case of 50% of the current load in the previous system state in Network 30 (number of neighbors).

Load of previous state: 150% of current load- I					
30 node network	**Reference**	**Biggest generator**		**Smallest generator**	
Iteration of failure	-	100	1200	100	1200
Entity failed	-	1		5	
Neighbours	-	2,3(2)		2,7(2)	
Generation (p.u.)	-	3.602		1	
Voltage convergence	yes	yes	yes	yes	no
Power convergence	yes	yes	yes	yes	-
Total generation (p.u.)	2.97	3.03	3.03	3.04	-
Total demand (p.u.)	2.83				
Power losses deviation (p.u.)	-	0.06	0.06	0.07	-
Max iterations	3000				
Iterations till convergence	1379	189	1224	147	3000
Iterations after failure	-	89	24	47	-

Table 6.25: Simulation results in case of 150% of the current load in the previous system state in Network 30 (generator size).

Load of previous state: 150% of current load- II						
30 node network	**Max. neighbours**		**Min. neighbours**		**Avrg. neighbours**	
Iteration of failure	100	1200	100	1200	100	1200
Entity failed	6		11		28	
Neighbours	2,4,7,8,9,10,28(7)		9(1)		6,8,27(3)	
Generation (p.u.)	0		1		0	
Voltage convergence	yes	yes	yes	yes	yes	yes
Power convergence	yes	yes	yes	yes	yes	yes
Total generation (p.u.)	3.06	2.84	3.1	2.99	2.84	2.86
Total demand (p.u.)	2.83					
Power losses dev. (p.u.)	0.09	-0.13	0.13	0.02	-0.13	-0.11
Max iterations	3000					
Iterations till converg.	188	1373	251	1208	313	1369
Iterations after failure	88	173	151	8	213	169

Table 6.26: Simulation results in case of 150% of the current load in the previous system state in Network 30 (number of neighbors).

In this case, the results are evident. The convergence is much better in the 150% previous load case in comparison to the 50% previous load case. In fact, while only the case with the minimum neighbors converges in the 50% case, only the smallest generator failure at the end of the algorithm does not converge in the 150% case. This is what we expected, as, coming from a more loaded state, if one entity fails, convergence is much easier due to the 10% losses allowed by the power stopping criterion. However, coming from a less loaded state, the generated power will have to meet at least the demand, which gives a lot less play to the algorithm. This could be summed up in that the algorithm is more fault resistant when the load is going down than when the load is going up. These results may also suggest that in extreme cases (such as the 50% load case) the number of neighbors plays a greater rol, as the only case that managed to converge in that situation was the min. neighbors case.

Multiple Entity Failures

Multiple entities failures results- I							
30 node network	Reference	MEF1		MEF2		MEF3	
Iteration of failure	-	100	1200	100	1200	100	1200
Entity failed	-	6,28		6,28,1		15,12,10,6,8	
Voltage convergence	yes	yes	yes	yes	yes	yes	yes
Power convergence	yes	yes	yes	yes	yes	yes	yes
Total generation (p.u.)	2.97	3.11	2.94	3.11	2.89	2.84	2.9
Total demand (p.u.)	2.83						
Power losses deviation (p.u.)	-	0.14	-0.03	0.14	-0.08	-0.13	-0.07
Iterations till convergence	1379	161	1221	159	1221	183	1216
Iterations after failure	-	61	21	59	21	83	16

Table 6.27: Simulation results of multiple entity failures in Network 30-I.

Multiple entities failures results- II							
30 node network	Reference	MEF4		MEF5		MEF6	
Iteration of failure	-	100	1200	100	1200	100	1200
Entity failed	-	14,15,18,19,20		3,14,16,18,19		3,4,6,7,9,12,28	
Voltage convergence	yes	yes	yes	no	no	yes	yes
Power convergence	yes	yes	yes	-	-	no	no
Total generation (p.u.)	2.97	2.85	3.072	-	-	-	-
Total demand (p.u.)	2.83						
Power losses deviation (p.u.)	-	-0.12	0.102	-	-	-	-
Iterations till convergence	1379	167	1217	-	-	-	-
Iterations after failure	-	67	17	-	-	-	-

Table 6.28: Simulation results of multiple entity failures in Network 30-II.

Multiple entities failures results- III					
118 node network	Reference	MEF1		MEF2	
Iteration of failure	-	50	450	50	450
Entity failed	-	49,10,11,69,1		8,17,20,34,47,49,62,63	
Voltage convergence	yes	yes	yes	yes	yes
Power convergence	yes	yes	yes	yes	yes
Total generation (p.u.)	43.09	42.43	42.44	42.42	42.42
Total demand (p.u.)	42.42				
Power losses deviation (p.u.)	-	-0.66	-0.65	-0.67	-0.67
Iterations till converg.	544	288	477	258	510
Iterations after failure	-	238	27	208	60

Table 6.29: Simulation results of multiple entity failures in Network 118-I.

Multiple entities failures results - IV									
118 node network	Reference	MEF3		MEF4		MEF5		MEF6	
Iteration of failure	-	50	450	50	450	50	450	50	450
Entity failed	-	1,10,20,...,110		1-35 and 89-118		1-59		1-89	
Voltage convergence	yes	yes	yes	yes	yes	yes	yes	yes	yes
Power convergence	yes	yes	no	yes	no	yes	no	no	no
Total generation (p.u.)	43.09	42.47	-	42.5	-	42.48	-	-	-
Total demand (p.u.)	42.42								
Power losses dev. (p.u.)	-	-0.62	-	-0.59	-	-0.61	-	-	-
Iterations till converg.	544	103	-	135	-	233	-	-	-
Iterations after failure	-	53	-	85	-	183	-	-	-

Table 6.30: Simulation results of multiple entity failures in Network 118-II.

With the multiple entity failures, we wanted to test the limits of convergence and see if we could, together with the other simulations, identify any more trends. In the 30 Network up to 5 entities could fail without convergence problems. With 6 failures, a case where the entities were close to each other and with no generators converged (MEF4) and a case where the entities were more spread out and one generator was affected didn't converge (MEF5). MEF6 is a case that affected all the generators and was anticipated wouldn't converge, as none of the generators is controllable. In the 118 network cases with up to 8 entities didn't have trouble converging. MEF3, with 11 entity failures, could not converge when the failure was at the end of the algorithm. In MEF5 with 59 entity failures all relatively close and MEF4 with 65 entity failures spread in two groups, the same story repeats itself: convergence if the failure happens at the beginning and no convergence if it happens at the end. MEF6 with 89 failures does not manage to converge.

Link Failures

Link failures results							
30 node network	Reference	LF1	LF2	LF3	LF4	LF5	LF6
Iteration of failure	-	500	500	500	500	500	500
links failed	-	1	2	4	6	7	8
Voltage convergence	yes	yes	yes	yes	yes	yes	yes
Power convergence	yes	yes	yes	yes	yes	yes	no
Total power (p.u.)	2.97	2.86	2.85	2.84	2.84	2.84	-
Total generation (p.u.)	2.83						
Power losses deviation (p.u.)	-	-0.11	-0.12	-0.13	-0.13	-0.13	-
Iterations till convergence	1379	544	575	583	587	709	-
Iterations after failure	-	44	75	83	87	209	-

Table 6.31: Simulation results of link failures in Network 30.

While the 30 node network was starting to have troubles converging with 5 entity failures, it manages to converse with up to 7 link failures. This confirms our claim that the link failures are less critical than the entity failures.

7 Conclusion and Future Work

The communication failure detection and mitigation strategy developed in this thesis proved to be effective in allowing the algorithm to reach fast and feasible solutions under many different communication failure scenarios. Single entity and link failure scenarios always converged and many of the multiple entity and link failure scenarios too.

Nevertheless, the communication failure detection and mitigation strategy has one shortcoming. In the case of entity failures, the information about the current load of those entities that fail is lost. The best the algorithm can do is to assume that the load did not change much with respect to the last known state of the system, which is a realistic assumption given that the algorithm is real-time and executed every few minutes. The algorithm also assumes that the generation of the failed entities will remain the same as in the previous state of the system, which, if the generation units are renewable, may not be possible. This means that the solution the algorithm gives has an error produced by the load of the failed entities and the potential shortage of generation, which will produce slight local power inbalance. This power inbalance will be taken care by the local controllers of the converters and will depend on their Input State Stability (ISS). However, it must be noted that the solution of the algorithm will always slightly differ from the actual operating point due to the uncertainties regarding the system model (line parameters, control mode of the converters), the measurements, and the current state of the system which is input to the algorithm (available RES power, load). This strategy manages to keep the topology of the electrical network intact to the algorithm even if the communication network topology changes due to the communication failure, and assumes the misinformation about the load of the failed entity or entities as one more uncertainty. Of course, this means that, regardless of the convergence of the algorithm, the more entity failures there are in the system the more will the solution stray away from reality. As future work, the impact of this phenomenon could be analyzed through uncertainties and probabilistic OPF. This could run in real time if it is fast enough, or the analysis could be used to modify the parameters of the constraint of the algorithm.

Regarding the communication failures, through the more than 80 different scenarios simulated, the following conclusions cloud be drawn:

- The size of the generator that fails does not seem to have any special influence regarding the convergence of the algorithm.

- When the load of the previous state of the system is greater than the load of the current one, the solution of the previous state of the system in terms of generation is already enough to cover the current demand, while when the load of the previous state of the system is lower than the current one, then the algorithm has to make more efforts to meet the power criterion. Therefore, the algorithm is more fault resistant when the load is in a decreasing trend compared to when the load is in an increasing trend.
- The number of neighbors of the entity that fails do not show any particular influence in our results. In the cases of very low demand in the previous state of the system (50% load 30 node network), the only case that converges is the one with minimum amount of neighbors, which suggests that only in cases of difficulties in convergence achievement the number of neighbors becomes relevant.
- The time of the failure also has in general no influence in the convergence of the algorithm. Only in the multiple entity failure cases in the 118 network show that if the failures occurs at the beginning of the algorithm it has more chances to converge than if it occurs at the end. This also suggests that in cases of difficulties in convergence achievement the time of the failures play a bigger role.
- Spread out failures seem to be more critical than localized failures.
- The algorithm can handle multiple entity or link failures. The number of failures is network-case sensitive and can not be generalized.
- Bigger networks can handle more failures.
- Networks with more distributed generation are more failure resistant and have better convergence.

However, more research in this direction should be carried out. More simulation scenarios and more and different test networks would be useful to get more general and precise conclusions. The simulation tool developed in this thesis would significantly help in this endeavor, as it was designed with reusability and expandability in mind.

With the goal of developing a simulation tool that allowed to test the behavior of DOPF algorithms when communication failures occur, a fully distributed PMP ADMM based algorithm with parallel computation and real communication was implemented. With the comparison between the sequential prototype and the final parallel version the challenges of real communication became apparent: while the algorithm is numerically, i.e., computationally scalable, introducing real communication showed that the problem is communication bound and, therefore, communication latency plays a crucial role in the overall performance of the algorithm. While

with the implementation used in this thesis the total parallel execution times are in the few minutes range, with a consistent, homogeneous and fast communication technology such as 5G, it is expected that the algorithm would become scalable also from the communications point of view and yield total execution times below the one minute mark.

As explained in the paragraph above, the fully distributed algorithm, where to each node of the electrical network an entity is assigned and each entity runs in its own computer, yields a communication bound problem, meaning that the computation time each entity needs to solve the local OPF is minimal in comparison to the time they need to carry out the communication tasks. A future direction of research could be trying to balance out the computation and communication weight of the algorithm. This could be done by grouping the entities in a way that, within the group, the algorithm would work sequentially and without real communication between those entities, and between groups parallelization and real communication would take place. If we call each group of entities a *macro-entity*, each macro-entity would have increased computational load and the overall communication tasks would be decreased, as entities within one macro-entity would not need to communicate, they would just read values from the shared memory. The author suspects this would allow for much better scalability and result in an even better and faster simulation tool able to simulate much bigger cases with fewer hardware requirements. Not only could this make for a useful tool for simulation, but this macro-entity approach of the algorithm could be an alternative to other not fully distributed control strategies (like hierarchical structures), where instead of one computer/entity per electrical node, a group of nodes will depend on a single computer hosting the macro-entity for that group. This would present an intermediate approach between fully centralized and fully distributed control, where data privacy is lost among entities in one macro-entity but could still be mantained between macro-entities.

Appendix

A Test Networks Parameters

A.1 5 Node Network

Algorithm parameters	
Penalty parameter	10^6
Max. PMP ADMM iterations	$5 \cdot 10^3$
Max. SCA iterations	1
Voltage convergence tolerance	$9/10^5$
Taylor points tolerance	$9/10^7$

Table A.1: 5 node network algorithm parameters.

Operational constraints	min	max
Injected current	-5	5
Injected power	-5	5
Current flow	-5	5
Power flow	-5	5
Voltage	0.9	1.1

Table A.2: 5 node network operational constraints.

Line resistances		
from node:	to node:	r(p.u.)
1	2	0.0028
1	4	0.0030
1	5	0.0006
2	3	0.0011
3	4	0.0030
4	5	0.0030

Table A.3: 5 node network line parameters.

A.2 14 Node Network

Algorithm parameters	
Penalty parameter	10^6
Max. PMP ADMM iterations	10^4
Max. SCA iterations	1
Voltage convergence tolerance	$9/10^5$
Taylor points tolerance	$9/10^7$

Table A.4: 14 node network algorithm parameters.

Operational constraints	min	max
Injected current	-5	5
Injected power	-5	5
Current flow	-5	5
Power flow	-5	5
Voltage	0.9	1.1

Table A.5: 14 node network operational constraints.

Line resistances		
from node:	to node:	r(p.u.)
1	2	0.0194
1	5	0.054
2	3	0.047
2	4	0.0581
2	5	0.057
3	4	0.067
4	5	0.0134
4	7	0.0907
4	9	0.2413
5	6	0.1094
6	11	0.095
6	12	0.1229
6	13	0.0662
7	8	0.0764
7	9	0.0477
9	10	0.0318
9	14	0.1271
10	11	0.0821
12	13	0.2209

13	14	0.1709

Table A.6: 14 node network line parameters.

A.3 30 Node Network

Algorithm parameters	
Penalty parameter	10^6
Max. PMP ADMM iterations	$5 \cdot 10^3$
Max. SCA iterations	1
Voltage convergence tolerance	$9/10^5$
Taylor points tolerance	$9/10^7$

Table A.7: 30 node network algorithm parameters.

Operational constraints	min	max
Injected current	-5	5
Injected power	-5	5
Current flow	-5	5
Power flow	-5	5
Voltage	0.9	1.1

Table A.8: 30 node network operational constraints.

Line resistances		
from node:	to node:	r(p.u.)
1	2	0.0192
1	3	0.0452
2	4	0.057
3	4	0.0132
2	5	0.0472
2	6	0.0581
4	6	0.0119
5	7	0.046
6	7	0.0267
6	8	0.012
6	9	0.0938
6	10	0.2507
9	11	0.0938
9	10	0.0496
4	12	0.1154
12	13	0.0631
12	14	0.1231
12	15	0.0662
12	16	0.0945

14	15	0.221
16	17	0.0524
15	18	0.1073
18	19	0.0639
19	20	0.034
10	20	0.0936
10	17	0.0324
10	21	0.0348
10	22	0.0727
21	22	0.0116
15	23	0.1
22	24	0.115
23	24	0.132
24	25	0.1885
25	26	0.2544
25	27	0.1093
28	27	0.1786
27	29	0.2198
27	30	0.3202
29	30	0.2399
8	28	0.0636
6	28	0.0169

Table A.9: 30 node network line parameters.

A.4 57 Node Network

Algorithm parameters	
Penalty parameter	10^6
Max. PMP ADMM iterations	$8 \cdot 10^3$
Max. SCA iterations	1
Voltage convergence tolerance	$9/10^5$
Taylor points tolerance	$9/10^7$

Table A.10: 57 node network algorithm parameters.

Operational constraints	min	max
Injected current	-18.5	18.5
Injected power	-18.5	18.5
Current flow	-18.5	18.5
Power flow	-18.5	18.5
Voltage	0.9	1.1

Table A.11: 57 node network operational constraints.

Line resistances		
from node:	to node:	r(p.u.)
1	2	0.0083
2	3	0.0298
3	4	0.0112
4	5	0.0625
4	6	0.043
6	7	0.02
6	8	0.0339
8	9	0.0099
9	10	0.0369
9	11	0.0258
9	12	0.0648
9	13	0.0481
13	14	0.0132
13	15	0.0269
1	15	0.0178
1	16	0.0454
1	17	0.0238
3	15	0.0162
4	18	0.2832

4	18	0.2194
5	6	0.0302
7	8	0.0139
10	12	0.0277
11	13	0.0223
12	13	0.0178
12	16	0.018
12	17	0.0397
14	15	0.0171
18	19	0.461
19	20	0.283
21	20	0.3964
21	22	0.0736
22	23	0.0099
23	24	0.166
24	25	0.6032
24	25	0.6277
24	26	0.0241
26	27	0.165
27	28	0.0618
28	29	0.0418
7	29	0.0331
25	30	0.135
30	31	0.326
31	32	0.507
32	33	0.0392
34	32	0.4863
34	35	0.052
35	36	0.043
36	37	0.029
37	38	0.0651
37	39	0.0239
36	40	0.03
22	38	0.0192
11	41	0.3822
41	42	0.207
41	43	0.2103
38	44	0.0289
15	45	0.0532
14	46	0.0375
46	47	0.023

47	48	0.0182
48	49	0.0834
49	50	0.0801
50	51	0.1386
10	51	0.0363
13	49	0.0975
29	52	0.1442
52	53	0.0762
53	54	0.1878
54	55	0.1732
11	43	0.0781
44	45	0.0624
40	56	0.6098
56	41	0.553
56	42	0.2125
39	57	0.6915
57	56	0.174
38	49	0.115
38	48	0.0312
9	55	0.0615

Table A.12: 57 node network line parameters.

A.5 118 Node Network

Algorithm parameters	
Penalty parameter	10^6
Max. PMP ADMM iterations	$1.5 \cdot 10^3$
Max. SCA iterations	1
Voltage convergence tolerance	$9/10^5$
Taylor points tolerance	$9/10^7$

Table A.13: 118 node network algorithm parameters.

Operational constraints	min	max
Injected current	-50	50
Injected power	-50	50
Current flow	-50	50
Power flow	-50	50
Voltage	0.9	1.1

Table A.14: 118 node network operational constraints.

Line resistances		
from node:	to node:	r(p.u.)
1	2	0.0303
1	3	0.0129
4	5	0.0018
3	5	0.0241
5	6	0.0119
6	7	0.0046
8	9	0.0024
8	5	0.007
9	10	0.0026
4	11	0.0209
5	11	0.0203
11	12	0.006
2	12	0.0187
3	12	0.0484
7	12	0.0086
11	13	0.0223
12	14	0.0215
13	15	0.0744
14	15	0.0595

12	16	0.0212
15	17	0.0132
16	17	0.0454
17	18	0.0123
18	19	0.0112
19	20	0.0252
15	19	0.012
20	21	0.0183
21	22	0.0209
22	23	0.0342
23	24	0.0135
23	25	0.0156
26	25	0.01
25	27	0.0318
27	28	0.0191
28	29	0.0237
30	17	0.0101
8	30	0.0043
26	30	0.008
17	31	0.0474
29	31	0.0108
23	32	0.0317
31	32	0.0298
27	32	0.0229
15	33	0.038
19	34	0.0752
35	36	0.0022
35	37	0.011
33	37	0.0415
34	36	0.0087
34	37	0.0026
38	37	0.0098
37	39	0.0321
37	40	0.0593
30	38	0.0046
39	40	0.0184
40	41	0.0145
40	42	0.0555
41	42	0.041
43	44	0.0608
34	43	0.0413

44	45	0.0224
45	46	0.04
46	47	0.038
46	48	0.0601
47	49	0.0191
42	49	0.0715
42	49	0.0715
45	49	0.0684
48	49	0.0179
49	50	0.0267
49	51	0.0486
51	52	0.0203
52	53	0.0405
53	54	0.0263
49	54	0.073
49	54	0.0869
54	55	0.0169
54	56	0.0028
55	56	0.0049
56	57	0.0343
50	57	0.0474
56	58	0.0343
51	58	0.0255
54	59	0.0503
56	59	0.0825
56	59	0.0803
55	59	0.0474
59	60	0.0317
59	61	0.0328
60	61	0.0026
60	62	0.0123
61	62	0.0082
63	59	0.0101
63	64	0.0017
64	61	0.007
38	65	0.009
64	65	0.0027
49	66	0.018
49	66	0.018
62	66	0.0482
62	67	0.0258

65	66	0.0096
66	67	0.0224
65	68	0.0014
47	69	0.0844
49	69	0.0985
68	69	0.0096
69	70	0.03
24	70	0.0022
70	71	0.0088
24	72	0.0488
71	72	0.0446
71	73	0.0087
70	74	0.0401
70	75	0.0428
69	75	0.0405
74	75	0.0123
76	77	0.0444
69	77	0.0309
75	77	0.0601
77	78	0.0038
78	79	0.0055
77	80	0.017
77	80	0.0294
79	80	0.0156
68	81	0.0018
81	80	0.0096
77	82	0.0298
82	83	0.0112
83	84	0.0625
83	85	0.043
84	85	0.0302
85	86	0.035
86	87	0.0283
85	88	0.02
85	89	0.0239
88	89	0.0139
89	90	0.0518
89	90	0.0238
90	91	0.0254
89	92	0.0099
89	92	0.0393

91	92	0.0387
92	93	0.0258
92	94	0.0481
93	94	0.0223
94	95	0.0132
80	96	0.0356
82	96	0.0162
94	96	0.0269
80	97	0.0183
80	98	0.0238
80	99	0.0454
92	100	0.0648
94	100	0.0178
95	96	0.0171
96	97	0.0173
98	100	0.0397
99	100	0.018
100	101	0.0277
92	102	0.0123
101	102	0.0246
100	103	0.016
100	104	0.0451
103	104	0.0466
103	105	0.0535
100	106	0.0605
104	105	0.0099
105	106	0.014
105	107	0.053
105	108	0.0261
106	107	0.053
108	109	0.0105
103	110	0.0391
109	110	0.0278
110	111	0.022
110	112	0.0247
17	113	0.0091
32	113	0.0615
32	114	0.0135
27	115	0.0164
114	115	0.0023
68	116	0.0003

12	117	0.0329
75	118	0.0145
76	118	0.0164

Table A.15: 118 node network line parameters.

A.6 300 Node Network

Algorithm parameters	
Penalty parameter	10^6
Max. PMP ADMM iterations	10^4
Max. SCA iterations	1
Voltage convergence tolerance	$9/10^5$
Taylor points tolerance	$9/10^7$

Table A.16: 300 node network algorithm parameters.

Operational constraints	min	max
Injected current	-100	100
Injected power	-100	100
Current flow	-100	100
Power flow	-100	100
Voltage	0.9	1.1

Table A.17: 300 node network operational constraints.

Line resistances		
from node:	to node:	r(p.u.)
1	5	0.0001
2	6	0.0008
2	8	0.0244
3	7	0.0362
3	18	0.0158
3	129	0.0158
4	16	0.0160
5	9	0.0326
7	12	0.1713
7	110	0.0556
8	11	0.1112
8	14	0.1112
9	11	0.0762
11	13	0.0762
12	20	0.0537
13	19	1.1068
14	15	0.4436
15	31	0.5075
15	74	0.6669

15	75	0.6113
16	36	0.4412
18	20	0.3079
18	72	0.0558
19	21	0.7363
19	26	0.7698
20	23	0.7573
21	22	0.0738
22	24	0.0383
23	231	0.3661
24	25	1.0593
25	26	0.1567
25	232	0.1301
27	28	0.5448
27	32	0.1543
27	34	0.3849
27	35	0.4412
28	36	0.2355
29	60	0.1606
29	63	0.0010
29	64	0.0010
30	73	0.0060
31	32	0.0006
31	34	0.0080
31	35	0.0010
31	43	0.0020
31	74	0.0060
31	75	0.0010
32	35	0.0010
32	37	0.0130
33	36	0.0130
34	42	0.0060
35	36	0.0080
35	43	0.0020
35	44	0.0060
36	40	0.0140
37	38	0.0650
37	42	0.0990
37	46	0.0960
38	41	0.0020
38	47	0.0020

39	52	0.0130
39	62	0.0160
40	68	0.0690
41	61	0.0040
41	92	0.0520
42	87	0.0190
43	44	0.0070
44	45	0.0360
45	48	0.0450
46	47	0.0430
47	48	0.0135
48	49	0.0025
49	50	0.0060
49	55	0.0070
50	51	0.0010
51	53	0.0120
52	54	0.0060
54	56	0.0100
54	123	0.0040
55	236	0.0080
57	190	0.0220
57	66	0.0100
58	59	0.0170
58	237	0.1020
59	60	0.0470
59	61	0.0080
60	64	0.0320
60	238	0.0006
61	63	0.0260
61	66	0.0062
62	73	0.0650
62	240	0.0310
63	64	0.0020
64	65	0.0260
64	67	0.0950
64	239	0.0130
64	241	0.0270
65	66	0.0280
65	69	0.0070
66	190	0.0090
67	190	0.0050

68	173	0.0520
68	174	0.0430
70	71	0.0250
71	72	0.0310
71	234	0.0370
74	76	0.0270
75	77	0.0250
76	78	0.0350
76	79	0.0650
77	84	0.0460
77	86	0.1590
78	79	0.0090
79	82	0.0020
79	83	0.0090
79	84	0.0160
80	82	0.0010
80	83	0.0265
81	87	0.0510
81	88	0.0510
81	89	0.0320
81	90	0.0200
82	83	0.0360
83	85	0.0340
84	86	0.0180
85	88	0.0256
85	233	0.0210
86	87	0.0180
86	90	0.0040
88	235	0.0286
89	90	0.0160
89	92	0.0010
89	93	0.0140
90	91	0.0891
91	93	0.0782
94	101	0.0060
95	99	0.0077
96	97	0.0990
97	98	0.0220
97	245	0.0035
245	99	0.0035
97	100	0.0080

98	99	0.0120
98	100	0.0060
101	102	0.0470
101	104	0.0320
102	103	0.1000
102	104	0.0220
104	105	0.0190
105	106	0.0170
105	108	0.2780
105	111	0.0220
105	136	0.0380
105	137	0.0480
105	148	0.0240
106	107	0.0340
106	113	0.0530
106	147	0.0020
107	109	0.0450
107	112	0.0500
108	109	0.0160
108	112	0.0430
109	111	0.0190
109	130	0.0760
109	146	0.0440
109	147	0.0120
112	116	0.1570
112	147	0.0740
112	148	0.0700
112	150	0.1000
113	114	0.1090
113	163	0.1420
114	115	0.0170
115	116	0.0036
115	131	0.0020
116	119	0.0001
116	160	0.0058
116	165	0.1320
116	167	0.0792
118	151	0.0022
119	120	0.0073
119	121	0.0125
119	124	0.0808

119	125	0.0965
119	126	0.0360
119	161	0.0476
120	125	0.0006
121	122	0.0059
122	124	0.0115
122	128	0.0198
124	125	0.0050
124	128	0.0077
125	126	0.0165
127	157	0.0059
127	158	0.0049
131	132	0.0059
132	140	0.0078
133	135	0.0026
133	162	0.0076
134	140	0.0021
136	138	0.0016
137	138	0.0017
137	139	0.0079
141	143	0.0078
141	144	0.0017
142	143	0.0026
144	145	0.0021
146	148	0.0002
151	152	0.0043
151	153	0.0039
152	153	0.0091
152	154	0.0125
152	155	0.0056
154	155	0.0015
154	158	0.0005
155	156	0.0007
156	157	0.0005
157	158	0.0562
157	159	0.0120
160	117	0.0152
160	166	0.0468
163	164	0.0430
165	167	0.0489
166	167	0.0013

167	117	0.0291
168	187	0.0060
168	188	0.0075
169	210	0.0127
169	219	0.0085
170	171	0.0218
171	204	0.0073
172	184	0.0523
172	187	0.1371
173	198	0.0137
173	242	0.0055
174	198	0.1746
175	176	0.0804
175	189	0.0110
176	177	0.0008
176	190	0.0029
177	181	0.0066
177	182	0.0024
177	189	0.0018
177	190	0.0044
178	179	0.0002
178	189	0.0018
179	189	0.0669
180	183	0.0558
182	190	0.0807
183	184	0.0739
184	185	0.1799
185	186	0.0904
185	187	0.0770
191	194	0.0251
192	193	0.0222
193	194	0.0498
193	221	0.0061
194	195	0.0004
195	196	0.0004
196	197	0.0025
196	198	0.0007
196	199	0.0007
198	216	0.0004
199	197	0.0330
199	200	0.0460

199	217	0.0004
200	202	0.0059
201	216	0.0030
203	204	0.0020
203	205	0.0450
204	170	0.0480
205	210	0.0031
206	210	0.0024
207	208	0.0031
207	210	0.0140
207	213	0.0300
208	169	0.0100
210	211	0.0150
210	216	0.3320
211	212	0.0090
213	214	0.0200
213	216	0.0340
214	217	0.0760
220	216	0.0400
219	230	0.0810
221	224	0.1240
221	226	0.0100
222	223	0.0460
222	224	0.3020
223	225	0.0730
224	225	0.2400
224	226	0.0139
225	226	0.0025
226	227	0.0017
227	228	0.0015
228	229	0.0045
3	1	0.0040
3	2	0.0098
3	4	0.0005
7	5	0.0027
7	6	0.0003
10	11	0.0037
12	10	0.0010
15	17	0.0016
16	15	0.0003
20	19	0.0014

23	22	0.0100
30	29	0.0019
39	38	0.0010
39	40	0.0005
54	53	0.0009
55	56	0.0019
61	62	0.0026
68	73	0.0013
70	81	0.0134
71	83	0.0002
72	78	0.0001
93	186	0.0017
95	103	0.0002
100	94	0.0006
101	136	0.0002
109	110	0.0005
109	129	0.0003
111	149	0.0082
120	153	0.0112
121	154	0.0127
122	123	0.0326
122	127	0.0195
124	159	0.0157
130	149	0.0360
132	162	0.0268
134	135	0.0428
138	96	0.0351
139	103	0.0616
142	116	0.0111
143	134	0.0111
161	118	0.0011
168	189	0.0084
172	175	0.0084
174	191	0.0191
179	227	0.0114
180	57	0.0194
181	190	0.0010
183	246	0.0030
188	177	0.0137
190	191	0.0101
197	198	0.0043

202	203	0.0045
208	209	0.0126
213	215	0.0081
217	218	0.0052
175	246	0.0043
98	243	0.0103
99	244	0.0103
248	2	0.0099
249	3	0.0319
260	53	0.0052
261	54	0.0060
265	145	0.0005
254	23	0.0039
247	1	0.0030
263	109	0.0010
250	11	0.0024
253	22	0.0024
257	43	0.0178
264	118	0.0013
251	12	0.0005
252	17	0.0010
255	33	0.0027
259	49	0.0008
256	38	0.0034
258	48	0.0012
262	59	0.0013
31	266	0.0009
266	270	0.0003
266	271	0.0540
266	273	0.0508
270	292	0.0008
270	293	0.0471
270	294	0.0210
270	295	0.0274
270	296	0.0200
271	272	0.0260
271	268	0.0030
271	268	0.0010
273	267	0.0012
273	267	0.0010
267	274	0.0005

274	276	0.0005
274	275	0.0001
267	277	0.0010
276	278	0.0049
276	279	0.0010
272	297	0.0023
272	298	0.0051
272	268	0.0069
268	280	0.0033
268	281	0.0062
268	282	0.0042
268	291	0.0041
291	269	0.0041
269	288	0.0049
269	289	0.0027
269	290	0.0036
268	283	0.0067
268	284	0.0035
268	285	0.0068
268	286	0.0115
268	287	0.0389
273	299	0.0420
294	300	0.0148

Table A.18: 300 node network line parameters.

List of Figures

1.1 Control strategies: (a) centralized (b) decentralized (c) distributed. . 3

2.1 Complex decomposition structure. 11

3.1 Electrical grid vs. entity network. 22
3.2 Clone voltages: calculation. 28
3.3 Clone voltages: sending. 29
3.4 Clone voltages: receiving. 29
3.5 Final situation after first communication round. 30
3.6 Net voltages: calculation. 30
3.7 Net voltages: sending. 31
3.8 Net voltages: receiving. 31
3.9 Final result after second communication round. 32
3.10 Step 5: Checking stopping criteria. 36
3.11 Step 5.1: Local voltage convergence check. 37
3.12 Step 5.2: Flag gathering. 37
3.13 Step 5.3: Flag checking. 38
3.14 Step 5.4: Voltage convergence broadcasting. 38
3.15 Step 5.5: Power gathering. 39
3.16 Step 5.6: Power criterion checking. 39
3.17 Step 5.7: Total convergence broadcasting. 40
3.18 Flowchart of the ADMM DOPF algorithm. 42

4.1 Example of link failure. 43
4.2 Example of entity failure. 44
4.3 Behavior against link failure. 46
4.4 Behavior against entity failure. 47

5.1 Example of UI: start.py. 56
5.2 Example of UI: embyUI.py. 56
5.3 Example of UI: EntityModule.py. Brief results report. 57
5.4 Example of UI: EntityModule.py. Saving and plotting results. . . . 57
5.5 Swarm node. 58
5.6 SWARM cluster. (Blue lines: communication through shared memory, yellow lines: inter-socket communication, green lines: InfiniBand). 59
5.7 MacBook Pro structure. 60

6.1 5 node network. 62

6.2 14 node network [44]. 63
6.3 30 node network [44]. 64
6.4 57 node network [44]. 65
6.5 118 node network [44]. 66
6.6 300 node network [44]. 67
6.7 Sequential execution. 68
6.8 Parallel execution. 69
6.9 Total time for all networks in sequential execution. 72
6.10 Time per iteration for all networks in sequential execution. 72
6.11 Time per iteration and number of entities for all networks in sequential execution. 73
6.12 Sequential vs parallel execution times per iteration [s]. 75
6.13 Example of allocation of 14 processes in 1 computing node. 76
6.14 Example of allocation of 14 processes in 2 computing nodes. 77
6.15 Example of allocation of 14 processes in 3 computing nodes. 77
6.16 Example of allocation of 14 processes in 4 computing nodes. 78
6.17 Evolution of all net voltages through the iterations of the algorithm. . 85
6.18 Power generation per node. 85
6.19 Total power generated in the system in each iteration and total demand. 86
6.20 Amount of entities with voltage convergence with respect to the total number of entities in each iteration. 86
6.21 Voltage profile comparison between previous state of the system and solution with failure. 87
6.22 Nodal power generation comparison between previous state of the system and solution with failure. 87
6.23 Voltage profile comparison between solution without failure and solution with failure. 88
6.24 Nodal power generation comparison between solution without failure and solution with failure. 88

List of Tables

5.1 Swarm cluster specifications. 58
5.2 MacBook Pro specifications. 60

6.1 5 node network main characteristics. 62
6.2 14 node network main characteristics. 63
6.3 30 node network main characteristics. 64
6.4 57 node network main characteristics. 65
6.5 118 node network main characteristics. 66
6.6 300 node network main characteristics. 66
6.7 Time metrics of all networks in PC and HPC for the sequential execution of the algorithm. 71
6.8 Sequntial vs. parallel execution times for all networks in PC and HPC using different amount of computing nodes. 74
6.9 Sequential vs. parallel execution times per iteration for all networks in PC and HPC using different amount of computing nodes. 74
6.10 Time of each communicaiton task in the 14 node network case in PC and 1, 2, 3 and 4 computing nodes of the HPC. 79
6.11 Time of each communicaiton task in the 30 node network case in PC and 1, 2, 3 and 4 computing nodes of the HPC. 79
6.12 Time of each communicaiton task in the 57 node network case in PC and 1, 2, 3 and 4 computing nodes of the HPC. 80
6.13 Time of each communicaiton task in the 118 node network case in PC and 1, 2, 3 and 4 computing nodes of the HPC. 80
6.14 Single entity failure cases. 83
6.15 Multiple entities failure cases. 84
6.16 Link failures cases. 84
6.17 Simulation results of single entity failure in Network 30 (generator size). 90
6.18 Simulation results of single entity failure in Network 30 (number of neighbors). 90
6.19 Simulation results of single entity failure in Network 57 (generator size). 91
6.20 Simulation results of single entity failure in Network 57 (number of neighbors). 91
6.21 Simulation results of single entity failure in Network 118 (generator size). 92
6.22 Simulation results of single entity failure in Network 118 (number of neighbors). 92

6.23 Simulation results in case of 50% of the current load in the previous system state in Network 30 (generator size). 93
6.24 Simulation results in case of 50% of the current load in the previous system state in Network 30 (number of neighbors). 94
6.25 Simulation results in case of 150% of the current load in the previous system state in Network 30 (generator size). 94
6.26 Simulation results in case of 150% of the current load in the previous system state in Network 30 (number of neighbors). 95
6.27 Simulation results of multiple entity failures in Network 30-I. 96
6.28 Simulation results of multiple entity failures in Network 30-II. 96
6.29 Simulation results of multiple entity failures in Network 118-I. 97
6.30 Simulation results of multiple entity failures in Network 118-II. 97
6.31 Simulation results of link failures in Network 30. 98

A.1 5 node network algorithm parameters. 105
A.2 5 node network operational constraints. 105
A.3 5 node network line parameters. 105
A.4 14 node network algorithm parameters. 106
A.5 14 node network operational constraints. 106
A.6 14 node network line parameters. 107
A.7 30 node network algorithm parameters. 108
A.8 30 node network operational constraints. 108
A.9 30 node network line parameters. 109
A.10 57 node network algorithm parameters. 110
A.11 57 node network operational constraints. 110
A.12 57 node network line parameters. 112
A.13 118 node network algorithm parameters. 113
A.14 118 node network operational constraints. 113
A.15 118 node network line parameters. 118
A.16 300 node network algorithm parameters. 119
A.17 300 node network operational constraints. 119
A.18 300 node network line parameters. 129

Bibliography

[1] T. Morstyn, B. Hredzak, and V. G. Agelidis, "Control strategies for microgrids with distributed energy storage systems: An overview", *IEEE Transactions on Smart Grid*, vol. 9, no. 4, pp. 3652–3666, 2018, ISSN: 1949-3053. DOI: 10.1109/TSG.2016.2637958.

[2] D. K. Molzahn, F. Dörfler, H. Sandberg, S. H. Low, S. Chakrabarti, R. Baldick, and J. Lavaei, "A survey of distributed optimization and control algorithms for electric power systems", *IEEE Transactions on Smart Grid*, vol. 8, no. 6, pp. 2941–2962, 2017, ISSN: 1949-3053. DOI: 10.1109/TSG.2017.2720471.

[3] B. H. Kim and R. Baldick, "A comparison of distributed optimal power flow algorithms", *IEEE Transactions on Power Systems*, vol. 15, no. 2, pp. 599–604, 2000, ISSN: 0885-8950. DOI: 10.1109/59.867147.

[4] W. Lu, M. Liu, S. Lin, and L. Li, "Fully decentralized optimal power flow of multi-area interconnected power systems based on distributed interior point method", *IEEE Transactions on Power Systems*, vol. 33, no. 1, pp. 901–910, 2018, ISSN: 0885-8950. DOI: 10.1109/TPWRS.2017.2694860.

[5] T. Erseghe, "Distributed optimal power flow using admm", *IEEE Transactions on Power Systems*, vol. 29, no. 5, pp. 2370–2380, 2014, ISSN: 0885-8950. DOI: 10.1109/TPWRS.2014.2306495.

[6] M. Hong, "A distributed, asynchronous, and incremental algorithm for non-convex optimization: An admm approach", *IEEE Transactions on Control of Network Systems*, vol. 5, no. 3, pp. 935–945, 2018, ISSN: 2325-5870. DOI: 10.1109/TCNS.2017.2657460.

[7] J. Jian, C. Zhang, L. Yang, and K. Meng, "A hierarchical alternating direction method of multipliers for fully distributed unit commitment", *International Journal of Electrical Power and Energy Systems*, vol. 108, pp. 204 –217, 2019, ISSN: 0142-0615. DOI: https://doi.org/10.1016/j.ijepes.2018.12.043. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0142061518320891.

[8] S. Magnússon, P. C. Weeraddana, and C. Fischione, "A distributed approach for the optimal power-flow problem based on admm and sequential convex approximations", *IEEE Transactions on Control of Network Systems*, vol. 2, no. 3, pp. 238–253, 2015, ISSN: 2325-5870. DOI: 10.1109/TCNS.2015.2399192.

[9] J. Guo, G. Hug, and O. Tonguz, "Impact of partitioning on the performance of decomposition methods for ac optimal power flow", in *2015 IEEE Power Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, 2015, pp. 1–5. DOI: 10.1109/ISGT.2015.7131832.

[10] J. Mohammadi, J. Zhang, S. Kar, G. Hug, and J. M. F. Moura, "Multilevel distributed approach for dc optimal power flow", in *2015 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, 2015, pp. 1121–1125. DOI: 10.1109/GlobalSIP.2015.7418372.

[11] M. Lu and C. L. P. Chen, "The design of multi-agent based distributed energy system", in *2009 IEEE International Conference on Systems, Man and Cybernetics*, 2009, pp. 2001–2006. DOI: 10.1109/ICSMC.2009.5346133.

[12] Y. Wang, L. Wu, and S. Wang, "A fully-decentralized consensus-based admm approach for dc-opf with demand response", *IEEE Transactions on Smart Grid*, vol. 8, no. 6, pp. 2637–2647, 2017, ISSN: 1949-3053. DOI: 10.1109/TSG.2016.2532467.

[13] Q. Ling, Y. Liu, W. Shi, and Z. Tian, "Communication-efficient weighted admm for decentralized network optimization", in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016, pp. 4821–4825. DOI: 10.1109/ICASSP.2016.7472593.

[14] A. Krkoleva, V. Borozan, and P. Romanos, "Implementation of gossip algorithms in power systems", vol. 54, Oct. 2010, pp. 20–28. DOI: 10.1007/978-3-642-19322-4_3.

[15] K. Huang, D. A. Cartes, and S. K. Srivastava, "A multiagent-based algorithm for ring-structured shipboard power system reconfiguration", *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 37, no. 5, pp. 1016–1021, 2007, ISSN: 1094-6977. DOI: 10.1109/TSMCC.2007.900643.

[16] R. Sherer, A. Gupta, and M. Hybinette, "Adaptive message clustering for distributed agent-based systems", in *2011 IEEE Workshop on Principles of Advanced and Distributed Simulation*, 2011, pp. 1–6. DOI: 10.1109/PADS.2011.5936756.

[17] T. Wu, K. Yuan, Q. Ling, W. Yin, and A. H. Sayed, "Decentralized consensus optimization with asynchrony and delays", *IEEE Transactions on Signal and Information Processing over Networks*, vol. 4, no. 2, pp. 293–307, 2018, ISSN: 2373-776X. DOI: 10.1109/TSIPN.2017.2695121.

[18] F. Xiao and L. Wang, "Asynchronous consensus in continuous-time multi-agent systems with switching topology and time-varying delays", *IEEE Transactions on Automatic Control*, vol. 53, no. 8, pp. 1804–1816, 2008, ISSN: 0018-9286. DOI: 10.1109/TAC.2008.929381.

[19] R. Olfati-Saber and R. M. Murray, "Consensus problems in networks of agents with switching topology and time-delays", *IEEE Transactions on Automatic Control*, vol. 49, no. 9, pp. 1520–1533, 2004, ISSN: 0018-9286. DOI: 10.1109/TAC.2004.834113.

[20] J. Hespanha, P. Naghshtabrizi, and Y. Xu, "A survey of recent results in networked control systems", *Proceedings of the IEEE*, vol. 95, pp. 138 –162, Feb. 2007. DOI: 10.1109/JPROC.2006.887288.

[21] R. A. Gupta and M. Chow, "Networked control system: Overview and research trends", *IEEE Transactions on Industrial Electronics*, vol. 57, no. 7, pp. 2527–2535, 2010, ISSN: 0278-0046. DOI: 10.1109/TIE.2009.2035462.

[22] J. Baillieul and P. J. Antsaklis, "Control and communication challenges in networked real-time systems", *Proceedings of the IEEE*, vol. 95, no. 1, pp. 9–28, 2007, ISSN: 0018-9219. DOI: 10.1109/JPROC.2006.887290.

[23] W. Wang, Y. Xu, and M. Khanna, "A survey on the communication architectures in smart grid", *Computer Networks*, vol. 55, no. 15, pp. 3604 –3629, 2011, ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2011.07.010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S138912861100260X.

[24] V. C. Gungor, D. Sahin, T. Kocak, S. Ergut, C. Buccella, C. Cecati, and G. P. Hancke, "A survey on smart grid potential applications and communication requirements", *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, pp. 28–42, 2013, ISSN: 1551-3203. DOI: 10.1109/TII.2012.2218253.

[25] J. Liang, B. Shen, H. Dong, and J. Lam, "Robust distributed state estimation for sensor networks with multiple stochastic communication delays", *International Journal of Systems Science*, vol. 42, no. 9, pp. 1459–1471, 2011. DOI: 10.1080/00207721.2010.550402. eprint: https://doi.org/10.1080/00207721.2010.550402. [Online]. Available: https://doi.org/10.1080/00207721.2010.550402.

[26] D. M. Falcao, F. F. Wu, and L. Murphy, "Parallel and distributed state estimation", *IEEE Transactions on Power Systems*, vol. 10, no. 2, pp. 724–730, 1995, ISSN: 0885-8950. DOI: 10.1109/59.387909.

[27] S. Kar, J. M. F. Moura, and K. Ramanan, "Distributed parameter estimation in sensor networks: Nonlinear observation models and imperfect communication", *IEEE Transactions on Information Theory*, vol. 58, no. 6, pp. 3575–3605, 2012, ISSN: 0018-9448. DOI: 10.1109/TIT.2012.2191450.

[28] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004, ISBN: 0521833787.

[29] 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, pp. 1–122, Jan. 2011, ISSN: 1935-8237. DOI: 10.1561/2200000016. [Online]. Available: http://dx.doi.org/10.1561/2200000016.

[30] S. Boyd, "Notes on decomposition methods", *Notes for EE364B*, pp. 1–36, Jan. 2007.

[31] A. Kargarian, J. Mohammadi, J. Guo, S. Chakrabarti, M. Barati, G. Hug, S. Kar, and R. Baldick, "Toward distributed/decentralized dc optimal power flow implementation in future electric power systems", *IEEE Transactions on Smart Grid*, vol. 9, no. 4, pp. 2574–2594, 2018, ISSN: 1949-3053. DOI: 10.1109/TSG.2016.2614904.

[32] M. Kraning, "Dynamic network energy management via proximal message passing", vol. 1, pp. 73–126, Jan. 2014. DOI: 10.1561/2400000002.

[33] S. Chakrabarti, M. Kraning, E. Chu, R. Baldick, and S. Boyd, "Security constrained optimal power flow via proximal message passing", in *2014 Clemson University Power Systems Conference*, 2014, pp. 1–8. DOI: 10.1109/PSC.2014.6808131.

[34] M. Andersen, J. Dahl, Z. Liu, and L. Vandenberghe, "Interior-point methods for large-scale cone programming", English, in *Optimization For Machine Learning*. MIT Press, 2011, ISBN: 9780262016469.

[35] *Python*, https://www.python.org/, Accessed: 2018-09-23.

[36] *Numpy*, https://www.numpy.org/, Accessed: 2018-10-10.

[37] *A hitchhiker guide to python numpy arrays*, https://towardsdatascience.com/a-hitchhiker-guide-to-python-numpy-arrays-9358de570121, Accessed: 2019-05-14.

[38] *Cvxopt: Python software for convex optimization*, https://cvxopt.org/, Accessed: 2018-10-10.

[39] *Cvxopt user's manual*, https://cvxopt.org/userguide/intro.html, Accessed: 2018-11-25.

[40] L Vandenberghe, "The cvxopt linear and quadratic cone program solvers", May 2019.

[41] *Mpi forum: Documents*, https://www.mpi-forum.org/docs/, Accessed: 2018-11-25.

[42] L. Dalcín, R. Paz, and M. Storti, "Mpi for python", *J. Parallel Distrib. Comput.*, vol. 65, no. 9, pp. 1108–1115, Sep. 2005, ISSN: 0743-7315. DOI: 10.1016/j.jpdc.2005.03.010. [Online]. Available: http://dx.doi.org/10.1016/j.jpdc.2005.03.010.

[43] *Matpower: Free, open-source electric power system simulation and optimization tools for matlab and octave*, http://www.pserc.cornell.edu/matpower/, Accessed: 2018-09-13.

[44] *Electric power systems analysis and nature-inspired optimization algorithms*, https://www.al-roomi.org/power-flow, Accessed: 2018-09-14.