UNIVERSIDADE FEDERAL DO CEARÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

NATHANAEL DUQUE GADELHA

DESPACHO ÓTIMO DE MICRORREDE USANDO ALGORITMOS DE CONTROLE DISTRIBUÍDO NO ESQUEMA MATRICIAL E NO SISTEMA MULTIAGENTE

FORTALEZA

2022

NATHANAEL DUQUE GADELHA

DESPACHO ÓTIMO DE MICRORREDE USANDO ALGORITMOS DE CONTROLE DISTRIBUÍDO NO ESQUEMA MATRICIAL E NO SISTEMA MULTIAGENTE

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Elétrica do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do título de Bacharel em Engenharia Elétrica.

Área de concentração: Sistemas de Energia Elétrica.

Orientadora: Profa. Ph.D. Ruth Pastôra Saraiva Leão

Coorientador: Profa. Me. Janaína Barbosa Almada

FORTALEZA

2022

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

G12d Gadelha, Nathanael Duque.
Despacho Econômico Ótimo de Microrrede Usando Algoritmos de Controle Distribuído no Esquema Matricial e no Sistema Multiagente / Nathanael Duque Gadelha. – 2022.
84 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Centro de Tecnologia, Curso de Engenharia Elétrica, Fortaleza, 2022.
Orientação: Profa. Dra. Ruth Pastôra Saraiva Leão.
Coorientação: Profa. Ma. Janaína Barbosa Almada.

1. Microrrede. 2. Despacho Ótimo. 3. Controle Distribuído. 4. Multiagente. 5. PADE. I. Título.
CDD 621.3

NATHANAEL DUQUE GADELHA

DESPACHO ÓTIMO DE MICRORREDE USANDO ALGORITMOS DE CONTROLE DISTRIBUÍDO NO ESQUEMA MATRICIAL E NO SISTEMA MULTIAGENTE

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Elétrica da Universidade Federal do Ceará, como requisito parcial à obtenção do título de Bacharel em Engenharia Elétrica.

Aprovada em: 15/07/2022.

BANCA EXAMINADORA

Profa. PhD. Ruth Pastôra Saraiva Leão (Orientadora)

Universidade Federal do Ceará (UFC)

Profa. Me. Janaína Barbosa Almada (Coorientadora)

Universidade Federal do Ceará (UFC)

Prof. Dr. Raimundo Furtado Sampaio.

Universidade Federal do Ceará (UFC)

Prof. Dr. Lucas Silveira Melo

Universidade Federal do Ceará (UFC)

AGRADECIMENTOS

À minha mãe: Francilinda, pelo apoio, cuidado e dedicação em todos os momentos da minha vida.

Às minhas Avós: Laura e Olinda, sempre preocupadas com minha saúde e bem estar.

Aos meus Avôs: Francisco e Antônio, que foram verdadeiros pais para mim.

Às minhas tias: Margarida, Maristella e Anni, que foram uma verdadeira inspiração de luta e conquista .

Aos meus amigos da Faculdade: Carolina, Raul, Bruno, Annalyanne e Alice, que estiveram comigo durante essa caminhada.

Aos meu colegas do GREI e a professora Ruth, por me auxiliar na elaboração desse trabalho.

Ao MEC e a CNPQ pelo apoio financeiro nas bolsas de garduação .

“Os que se encantam com a prática sem a ciência são como os timoneiros que entram no navio sem timão nem bússola, nunca tendo certeza do seu destino”.

(Leonardo da Vinci)

RESUMO

O crescimento da geração distribuída tem sido responsável por mudanças nas características dos sistemas de distribuição de energia elétrica (SDE) tais como: de rede passiva para ativa, de fluxo de potência unidirecional para bidirecional e com consumidores e prossumidores. Os múltiplos recursos energéticos dos SDEs são melhor gerenciados na forma de microrredes definidas como sistemas de potência de pequeno porte formados por fontes, dispositivos de armazenamento de energia e cargas controláveis. Para o gerenciamento de uma microrrede são usados métodos de despacho econômico ótimo (DEO), que visam obter o menor custo total de operação da microrrede, atendendo às restrições técnicas e normativas. O objetivo deste trabalho é apresentar o desenvolvimento de um sistema de despacho econômico ótimo para microrredes com algoritmos de otimização distribuídos em uma arquitetura de gerenciamento distrivuiído, a microrrede utilizada para os casos de estudo será composta de fontes despacháveis, fontes não despacháveis, baterias e cargas, serão utilizados os algoritmos de otimização distribuída de Consenso, Difusão, Difusão Exata e por fim, representa-se o algoritmo de consenso em um Sistemas Multiagentes (SMA). A plataforma Python Agent Development (PADE) foi usada para criação e execução dos agentes que participam da otimização da microrrede. O desempenho dos métodos é comparado quanto ao custo computacional e resultados obtidos do DEO. Dentre os métodos de otimização com modelagem matricial, a Difusão Exata mostrou ser o método mais rápido e de menor custo computacional total em relação aos seus similares. Adicionalmente, o método de Difusão Exata é o método que mais respeita a privacidade dos dados dos agentes participantes da microrrede, sendo necessário transmitir entre agentes somente a variável de difusão, respeitando o sigilo de outros dados. Já para aaplicação de SMA na plataforma PADE, foi desenvolvido um algoritmo para usar a API do mosaik possibilitando comunicação sincronizada entre os agentes da microrrede. O algoritmo de Consenso foi aplicado no SMA realizando com sucesso o DEO da microrrede.Por fim, foi feito um caso extra exclusivo do SMA para mostrar a ferramenta do *Plug and Play*.

Palavras-chave: Microrrede, Despacho Ótimo, Controle Distribuído, Consenso,Difusão, Difusão Exata, Sistema Multiagente, PADE, Sistemas de Distribuição de Energia Elétrica.

ABSTRACT

The growth of distributed generation has been accountable for changes in characteristics of the power distribution systems (SDEs), from passive to active networks, from unidirectional to bidirectional power flow and with consumers and prosumers. The multiple energy resources in the SDEs are best managed in the form of microgrids, which are small power systems made up of sources, energy storage devices and controllable loads. For the management of a microgrid, optimal economic dispatch (DEO) methods are used, which aim to obtain the lowest total operating cost of the microgrid, taking into account technical and regulatory restrictions. The objective of this work is to develop and simulate an optimal dispatch of a microgrid that comprises dispatchable sources, non-dispatchable sources, batteries, and controllable loads, using distributed optimization algorithms such as Consensus, Diffusion, Exact Diffusion, and Multi-agent systems (MAS). The Python Agent Development (PADE) platform was used to create and run the agents that participate in the optimal dispatch of the microgrid. The performance of the methods is compared in terms of computational cost and results obtained from the DEO. Among the optimization methods with matrix modeling, exact diffusion proved to be the fastest method with the lowest total computational cost when compared to its counterparts. Additionally, the Exact Diffusion method is the method that most respects the privacy of the data of the agents, with the diffusion parameter the only variable interchanged between agents, respecting the confidentiality of other data. As for the application of a Multiagent system on the PADE platform, an algorithm was developed to use the mosaik API enabling synchronized and successful communication between the microgrid agents. The Consensus algorithm was applied in the SMA, successfully performing the DEO of the microgrid.Finally, an extra SMA exclusive case was made to show the Plug and Play tool.

Keywords: Microgrid, Optimal Dispatch, Distributed Control, Consensus, Diffusion, Exact Diffusion, Multi-agent System, PADE, Power Distribution Systems.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

GD	Geração Distribuída
SDE	Sistema de Distribuição de Energia
PAC	Ponto de Acoplamento Comum
DEO	Despacho Econômico Ótimo
SMA	Sistema Multiagente
PADE	*Python Agent DEvelopment*
FIPA	*Foundation for Intelligent Physical Agents*
IoT	Internet of Things
AG	Algoritmo Genético
ACC	Adaptar, Corrigir, Comunicar
ACL	*Agent Comunication Language*
PA	Plataforma de Agentes
FD	Facilitador Diretório
STM	Serviço de Transporte de Mensagem
PTM	Protocolos de Transporte de Mensagem

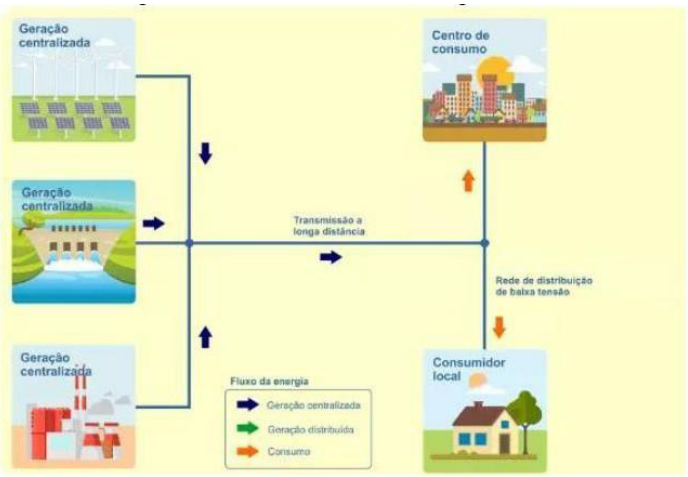
LISTA DE TABELAS

SUMÁRIO

1. Introdução

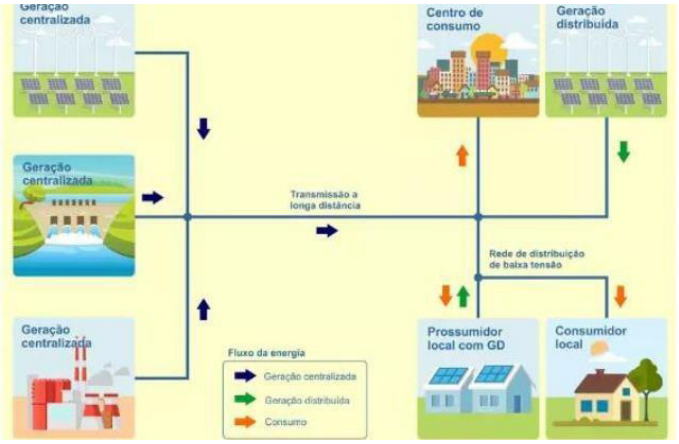
O crescimento da geração distribuída (GD) tem sido responsável por grandes transformações nos sistemas elétricos de distribuição, uma vez que, cada vez mais os consumidores migram de uma postura passiva para exercer uma postura mais ativa, em que não só consomem como também produzem e negociam a energia excedente com o operador da rede de distribuição. A presença de prossumidores contribui na mudança da direção do fluxo de energia nas redes de distribuição, podendo fluir dos prossumidores para a rede elétrica (TAVEIRA, 2020). A Figura 1 mostra o fluxo de energia elétrica em rede tradicional, enquanto a Figura 2 mostra uma abordagem mais contemporânea com fluxo bidirecional.

Figura 1:Fluxo unidirecional em rede elétrica tradicional.



Fonte: (CANAL SOLAR , 2021)

Figura 2:Fluxo bidirecional em rede elétrica contemporânea.

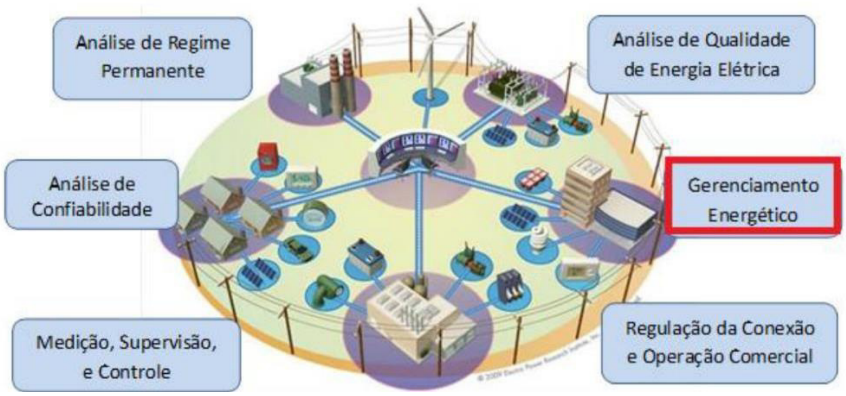


Fonte: (CANAL SOLAR, 2021)

A geração distribuída traz benefícios aos sistemas de distribuição de energia elétrica (SDEs), como redução das perdas no sistema de transmissão, uma vez que a geração está próxima aos centros de consumo, e maior diversificação da matriz energética, com a conexão de fontes eólicas e solar fotovoltaicas. No entanto, o nível de complexidade operacional dos SDEs torna-se maior devido ao grande número de participantes ativos e ao fluxo bidirecional de energia (NARUTO, 2017).

As microrredes podem promover a maior penetração de GDs, com aconsequente redução da complexidade operacional das redes de distribuição de energia. As microrredes são subsistemas compostos por GDs, dispositivos de armazenamento de energia e cargas, conectados, em média ou baixa tensão, à rede da concessionária por meio de um ponto de acoplamento comum (PAC). Uma microrrede pode operar em modo conectado ou isolado do SDE, sendo responsável pelo gerenciamento energético de seus componentes de modo a garantir qualidade da energia, confiabilidade e eficiência técnico-econômica. A Figura 3 mostra algumas ferramentas de análise e gerenciamento das microrredes.

Figura 3:Funções de uma microrrede.



Fonte: (LEGADO ENERGIA, 2020)

Para o gerenciamento energético de uma microrrede são usados métodos de despacho econômico ótimo (DEO) que visam otimizar o uso dos recursos energéticos da microrrede, atendendo às restrições técnicas e normativas.

Tradicionalmente, o gerenciamento energético é feito através de sistemas centralizados, como ilustrado a Figura 3, em que toda a estratégia de controle é feita por somente um dispositivo central que se comunica com todos os elementos da microrrede. Porém, a estrutura de controle centralizada demanda um tráfego significativo de dados entre o agente central e os elementos periféricos. Além disso,

esse tipo de controle apresenta baixa confiabilidade, uma vez que defeitos no agente central ou na rede de comunicação retiram de operação todo o sistema de gerenciamento energético (WANG, 2018).

No modelo centralizado são usados diferentes métodos para solução de DEO, como algoritmo genético (BAKIRTZIS, 1994), PSO (BARATI, 2016), (PARK, 2010) e (ABIDO, 2002), Busca Direta (ABIDO, 2002) e algoritmo diferencial de autoajuste (WANG, 2007). Todos esses métodos fazem uso de um agente Central que recebe as informações dos agentes Periféricos e, a partir dessas informações recebidas, o agente Central realiza o DEO.

Como alternativa ao controle centralizado têm sido desenvolvidas técnicas de controle distribuído que oferecem menor custo de comunicação, maior tolerância à falha e maior flexibilidade (WANG, 2018). Sistemas de controle distribuído são usados para instalações dentro de uma área confinada com fronteiras bem definidas e para processos de controle complexos, sendo, portanto, aplicável ao gerenciamento de microrredes. Técnicas de controle distribuído como algoritmo de Consenso, Difusão, Difusão Exata e Sistemas Multiagentes (SMA) têm sido largamente aplicadas em diferentes áreas da engenharia elétrica e em particular no gerenciamento de microrredes (WANG,2018),(ZHANG,2012),(HE,2019),(HAESSUM,2019) e(DEAZEVEDO,2016).

Zhang et al. (2012) e Wang et al., (2018) aplicam o algoritmo de Consenso para DEO de microrredes com somente agentes despacháveis e apresentam estudo sobre a velocidade de convergência ao ajustar parâmetros internos do algoritmo. Já Haessum et al. (2019) implementam o algoritmo de Consenso para microrrede composta por agente Despachável, agente Não Despachável, agente Bateria e agentes Cargas Inteligentes. Os autores utilizam a variável Benefício Incremental para a realizar Corte de Carga Inteligente. A variável Benefício Incremental define o nível de prioridade entre as Cargas que se deseja atender; caso o sistema não consiga atender toda a demanda, primeiramente corta-se a carga não prioritária, e, somente depois corta-se a carga prioritária. Porém (HAESSUM et al., 2019) não realiza os estudos de convergência vistos em (WANG et al., 2018). Wang et al. (2018) e Haessum et al. (2019) fazem uso somente do algoritmo de Consenso e, apesar do algoritmo conseguir realizar DEO, devido a instabilidades na própria construção do algoritmo, a convergência é lenta, exige muito processo computacional, requer uma troca de dados privados entre os agentes participantes da rede e possui um pequeno erro quando comparado a solução

ótima real. A privacidade de dados refere-se à troca de informações entre agentes que podem ser de caráter sigiloso.

Motivados por esses problemas no algoritmo de Consenso, De Azevedo et al., (2016) e He et al. (2019) implementaram o algoritmo de Difusão para DEO em uma microrrede. O algoritmo de Difusão corrige parcialmente as instabilidades do Consenso, sendo mais rápido, com demandando um menor esforço computacional, e aumentando consideravelmente a proteção de dados privados entre os Agentes, uma vez que muitos desses dados não são mais compartilhados, porém ainda mantém um pequeno erro quando comparado à solução ótima real. Além disso, (HE et al., 2019) também apresenta outra maneira de realizar o Corte de Carga Inteligente, dando a cada carga umnível mínimo e um nível máximo para mudança de seu estado de conexão. (HE et al., 2019) também aplica o algoritmo de Difusão Exata desenvolvido por (YUAN, 2018) para DEO de microrredes.

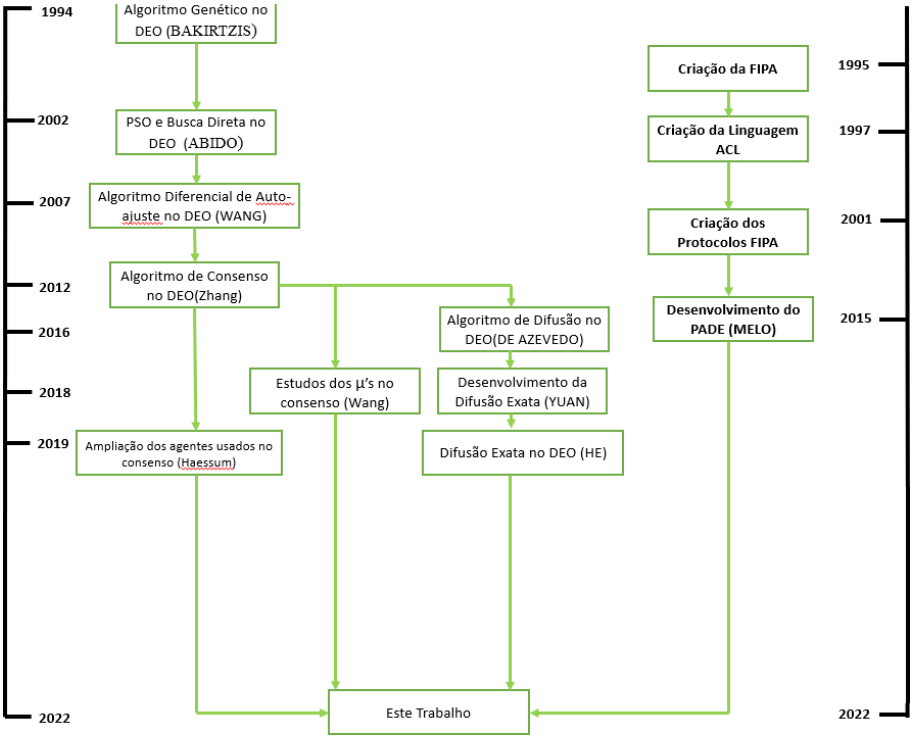
O algoritmo de Difusão Exata desenvolvido por (YUAN, 2018) corrige totalmente a instabilidade apresentada pelo Consenso, sendo mais rápido, mais eficiente, respeitando ainda mais a privacidade dos dados trocados dos agentes, uma vezque é necessário o menor número possível de dados compartilhados entre os agentesque fazem parte da microrrede, e levando ao valor exato do DEO. Outra contribuição importante de (YUAN, 2018) foi realizar um estudo de possíveis matrizes que podem ser utilizadas no algoritmo de Difusão Exata para melhorar ainda mais seu desempenho.

1.1 Motivação

Com base na revisão literária realizada para desenvolvimento de DEO para microrredes, este trabalho teve como motivação inicial agregar as contribuições dos trabalhos anteriormente apresentados, baseados em algoritmos distribuídos, utilizando o estudo de parâmetros desenvolvido por (WANG, 2018) para aumentar a velocidade do algoritmo, a estratégia de Corte de Carga desenvolvido por (HAESSUM, 2019) e o estudo de matrizes realizado por (YUAN, 2018) em um único estudo, com vistas a melhorar o desempenho do DEO para microrrede.

Os métodos supracitados representam a rede de comunicação entre os agentes por uma matriz de adjacências, não levando em consideração que, em um ambiente real de negociação entre agentes existe uma série de protocolos de comunicação a serem seguidos, bem como o problema de sincronização existente entre os agentes ao se realizar o DEO de maneira distribuída. Ciente dessa lacuna existente na literatura, tem- se como principal motivação desse trabalho implementar um Sistema Multiagente aplicado para DEO em uma microrrede utilizando a plataforma Python Agent DEvelopment (PADE) (MELO, 2019), com protocolos de comunicação padrão FIPA. AFigura 4 ilustra a revisão bibliográfica feita para este trabalho.

Figura 4:Revisão Bibliográfica



Fonte: Próprio Autor

1.2. Objetivos

O objetivo geral deste trabalho é apresentar o desenvolvimento de um sistema de despacho econômico ótimopara uma microrrede com algoritmos de otimização distribuídos em uma arquitetura de gerenciamento distribuído.

Como objetivos específicos tem-se:

- Desenvolver e comparar o desempenho de um DEO usando algoritmo de Consenso, Difusão e Difusão Exata;
- Desenvolver um Sistema Multiagente usando a plataforma PADE e o método de Consenso para despacho ótimo de uma microrrede;
- Usar a ferramenta de co-simulação mosaik para sincronização da comunicação entre agentes.

1.3. Organização do Trabalho

O trabalho está divido em 5 capítulos. O capítulo 1 apresenta uma visão dos sistemas de distribuição com geração distribuída e microrredes, a motivação, os objetivos e a estruturação do trabalho. O capítulo 2 descreve sobre métodos de controle distribuído com embasamento teórico conciso sobre Consenso, Difusão, Difusão Exatae Sistemas Multiagentes. O capítulo 3 discorre sobre teorias da economia utilizadas pararealizar o despacho econômico ótimo de maneira distribuída, apresentando as funções de custo de cada tipo de agente da otimização e as adaptações feitas nos algoritmos de otimização Consenso, Difusão, Difusão Exata bem como a modelagem dos agentes utilizados na plataforma multiagente PADE. O capítulo 4 mostra resultados de casos de estudo e a comparação de desempenho dos algoritmos matriciais e do SMA. Por fim, o capítulo 5 traz a conclusão do trabalho com sugestão para trabalhos futuros.

2. Técnicas de Controle Distribuído

Sistemas Distribuídos são um conjunto de componentes, que, através de uma rede de comunicação, operam em conjunto para a solução de um problema(TANENBAUM, ANDREW S.; STEEN, 2007). Essa comunicação deve obedecer a um conjunto de normas protocoladas internacionalmente (FIPA, 2015). Quando comparados a sistemas centralizados, ditos convencionais, os sistemas distribuídos oferecem como vantagens menor custo computacional, mais segurança e maior flexibilidade (WANG et al., 2018).

Mais recentemente, a Internet das Coisas (IoT) tem recebido atenção por permitir a conectividade de 'coisas' distribuídas para solicitar serviço ou compartilhar informação. Vê-se assim que, a tendência é que cada vez mais sistemas distribuídos sejam utilizados, já que os mesmos proporcionam as vantagens anteriormente mencionadas. (TAI-HOON ,2017)

Neste capítulo será apresentado de forma concisa o embasamento teórico para aplicação dos métodos baseados em arquitetura de controle distribuído, como Consenso, Difusão, Difusão Exata e SMA.

2.1. Grafos e Modelo Matricial

A rede de comunicação dos algoritmos de otimização distribuídos Consenso. Difusão e Difusão Exata pode ser modelada por um grafo não-dirigido, representado por $G = (V, E)$, composto por um conjunto de nós ou vértices, $V = \{1, 2, \ldots, n\}$ e um conjunto de arestas $E = (i, j) \subseteq V \times V$ (i e j se comunicam), com os nós adjacentes trocando mensagens em ambas as direções.

Seja a matriz de adjacências definida como:

$$A_{ij} = \begin{cases} 1, & para\ (i,j) \in E \\ 0, & caso\ contrário \end{cases} \tag{2.1}$$

A partir da matriz de adjacências, monta-se as matrizes de pesos P do sistema. Um elemento w_{kl} da matriz de peso P pode ser definido como sendo o grau de confiança que um componente ou agente *k* tem em uma informação que chegou do agente *l* (YUAN et al, 2017). Existem diferentes regras para a construção de matriz de pesos, porém, para os algoritmos de Consenso e Difusão usam matriz de peso do tipo *doubly-stochastic*. Nessas matrizes, tanto a soma das colunas quanto das linhas damatriz deverá ser igual a 1. No algoritmo de Difusão Exata, a matriz de peso é do tipo *left-stochastic*, ou seja, somente a soma das colunas deverá ser igual a 1.

Como toda matriz *doubly-stochastic* é também *left-stochastic*, mas nem toda matriz *left-stochastic* é *doubly-stochastic*, tem-se que o número de regras de construção da matriz de peso para o algoritmo de Difusão Exata será maior que o número de regras de construção para as matrizes de pesos dos algoritmos de Consenso e de Difusão, uma vez que todas as regras do Consenso e da Difusão podem ser utilizadas na Difusão Exata, porém nem todas as regras da Difusão Exata podem ser utilizadas nos algoritmos de Consenso e Difusão.

Além disso, tem-se que a regra utilizada para a construção da matriz de pesos influi no número de interações necessárias para o algoritmo convergir, e consequentemente no custo computacional e na velocidade do algoritmo. Além da Difusão Exata possuir um número maior de regras de construção, também se tem que matrizes *left-stochastic* normalmente necessitam de um número menor de interações para convergirem (YUAH et al, 2017).

Visando reduzir o custo computacional empregado nos algoritmos, foram estudados 5 tipos de regras de construção de matrizes de pesos diferentes, sendo duas delas *doubly-stochastics*: a matriz Metropolis (Equação 2.2) e a matriz *Mean Metropolis* (Equação 2.3); e três *left-stochastics:* a matriz de Hastings (Equação 2.4), a matriz Averaging (Equação 2.5) e a matriz *Relative-degree* (Equação 2.6).

- Matriz Metropolis:

$$W_{ij} = \begin{cases} \frac{1}{[\max(n_i, n_j)+1]}, \ j \in \Omega_i / \{i\} \\ 1 - \sum_{j \in \Omega_i} \frac{1}{[\max(n_i, n_j)+1]}, i = j \\ 0, \ caso\ contrário \end{cases} \quad (2.2)$$

Em que:

- $W_{ij:}$ Elemento da matriz de pesos
- n_i: Número de vizinhos do i-ésimo agente
- Ω_i: Conjunto dos vizinhos do i-ésimo agente

- Matriz *Mean Metropolis*:

$$W_{ij} = \begin{cases} \frac{2}{[n_i + n_j + \varepsilon]}, \ j \in \Omega_i / \{i\} \\ 1 - \sum_{j \in \Omega_i} \frac{2}{[n_i + n_j + \varepsilon]}, i = j \\ 0, \ caso\ contrário \end{cases} \quad (2.3)$$

Em que:

- $w_{ij:}$ Elemento da matriz de pesos
- n_i: Número de vizinhos do i-ésimo agente
- Ω_j: Conjunto dos vizinhos do i-ésimo agente
- ε: número que varia de 0 a 1, no trabalho ele considerado 1

- Matriz de *Hastings*:

$$W_{lk} = \begin{cases} \frac{\mu_k / q_k}{\max\{[n_k \mu_k / q_k, n_l \mu_l / q_l]\}}, \ se\ l \in \Omega_k / \{k\} \\ 1 - \sum_{j \in \Omega_k / \{k\}} w_{jk}, se\ l = k \\ 0, \ se\ l \notin \Omega_k. \end{cases} \quad (2.4)$$

Em que:

- $W_{lk:}$ Elemento da matriz de pesos
- n_k: Número de vizinhos do k-ésimo agente
- Ω_k: Conjunto dos vizinhos do k-ésimo agente
- q_k: Constante relacionada ao agente k
- μ_l : passo do agente *l*, será explicado posteriormente

- Matriz Averaging:

$$W_{lk} = \begin{cases} 1/n_k, \ se\ l \in \Omega_k \\ 0, caso\ contrário \end{cases} \quad (2.5)$$

Em que:

- $W_{lk:}$ Elemento da matriz de pesos
- n_k: Número de vizinhos do k-ésimo agente
- Ω_l: Conjunto dos vizinhos do k-ésimo agente

- Matriz Relative-degree:

$$W_{lk} = \begin{cases} \frac{n_l}{\sum_{m \in \Omega_k} n_m} \\ 0, caso\ contrário \end{cases} \quad (2.6)$$

- $W_{lk:}$ Elemento da matriz de pesos
- n_k: Número de vizinhos do k-ésimo agente

Para os algoritmos de Consenso e de Difusão,foram testados todos os modelos descritos anteriormente e foi observado que a matriz *Mean Metropolis* apresentou um melhor desempenho, já para o algoritmo de Difusão Exata, a matriz Averaging foi a que apresentou melhor desempenho.

Por fim, ao comparar a equação da matriz *Mean Metropolis* (2.3), utilizada no Consenso e na Difusão com a equação da matriz *Averaging* (2.5), utilizada na Difusão Exata, observa-se ainda outra vantagem da matriz *Averaging*: a mesma provê um grau de proteção à privacidade dos usuários maior que a outra, já que, para a construção da *Mean Metropolis* cada usuário necessita compartilhar o número de agentes com os quais se comunica, enquanto que, para a construção da matriz *Averaging* necessita-se apenas saber quantos usuários utilizam a rede, sendo esse um dado geral, e não específico de cada agente. Segundo Yuan et al. (2017), matrizes *Averaging* são comumente utilizadas em análises de dados envolvendo redes sociais justamente por proverem maior privacidade aos usuários dessas redes.

2.2. Algoritmo de Consenso

O algoritmo de Consenso é o mais simples dos três algoritmos de otimização distribuída dentre os estudados. Possui basicamente duas etapas: comunicar e atualizar. O algoritmo irá repetir essas duas etapas até que consiga atingir um determinado nível de convergência (HE et al., 2019).

Na etapa de comunicação, cada agente irá comunicar aos seus vizinhos o valor da sua variável de interesse como também receber esses mesmos valores de cada um dos seus vizinhos. Ao receber esses valores do agente *l,* o agente *k* irá multiplicá-lo pelo elemento da matriz de pesos w_{kl}, dessa forma, o algoritmo dá um grau de confiança à mensagem que chega do agente *l* (YUAN et al., 2017). Uma das desvantagens desse algoritmo é que irá-se comunicar o custo incremental de cada um dos agentes com os agentes vizinhos, o que diminui a privacidade de cada um dos agentes participantes do processo de otimização. A Equação 2.7 mostra o algoritmo de Consenso, a etapa de comunicação foi posta em negrito, a variável de interesse é λ_j , w_{ij} é o elemento da matriz de pesos, k é o número de interações e s é uma variável interna do consenso.

Após receber os dados os agentes avançam para a etapa de atualização. Nessa etapa cada agente irá atualizar as variáveis que ele fornece para o sistema (WANG, 2018). Após os agentes atualizarem essas variáveis o algoritmo irá verificar se o critério de parada é satisfeito por todos os agentes. Caso isso aconteça, significa que o sistema multiagente está obedecendo o princípio da igualdade marginal (HE et al, 2019), o algoritmo então irá parar de rodar e exibir essas variáveis na tela. Caso isso nãoaconteça, então o algoritmo irá voltar a executar a etapa de comunicação. A equação (2.7) também mostra a etapa de atualização (WANG, 2018).

$$\begin{cases} \lambda_i(k+1) = \sum_{j \in \Omega_i} \boldsymbol{w_{ij}\lambda_j(k)} + \mu_i(k)s_i(k) \\ s_i(k+1) = \sum_{\boldsymbol{j} \in \Omega_i} \boldsymbol{w_{ij}s_j(k)} \end{cases} \tag{2.7}$$

Observando a equação (2.7), vê-se que o custo incremental também depende de uma variável de atualização μ_i, chamada passo, que varia de agente para agente, influenciando na convergência e no número de interações necessárias para o algoritmo convergir (WANG, 2018). O valor de μ_i pode variar de 0 a 1, porém, como esse valor influencia diretamente na velocidade de convergência do algoritmo, é comum encontrar autores tais como (WANG, 2018) e (HE, 2019) que utilizam alguma meta heurística para encontrarem o conjunto de μ_i que levará o código a convergir utilizando o menor número de interações.

Uma outra maneira de escrever a equação apresentada em (2.7) é representada pela equação (2.8) (SAYED, 2014). Seu primeiro termo, marcado em negrito, é o chamado termo de cooperação. Esse termo depende da informação fornecida pelos vizinhos do agente, enquanto o segundo termo é o chamado termo de descentralização. Esse termo depende basicamente da informação já armazenada pelo agente na interação *i-1* (SAYED, 2014).

$$\lambda_i(k) = \sum_{l \in \Omega_i} \boldsymbol{w_{li}\lambda_l(k-1)} - \mu_i \nabla_\Omega J_i(\lambda_i(k-1)) \qquad (2.8)$$

A equação (2.8) evidencia melhor a assimetria da etapa de atualização do algoritmo de Consenso, enquanto que o termo de descentralização é $\lambda_{l,\ i-1}$, o termo de cooperação é diferente e envolve uma combinação onde a soma de todos os elementos $w_{l,k}$ é igual a 1 (SAYED, 2014).

Essa assimetria no algoritmo de Consenso leva a uma instabilidade do mesmo, ou seja, um número menor de μ_i que fará com que o Consenso convirja (YUAN et al, 2017). Outro problema que o algoritmo de Consenso possui é que seu resultado final apresentará um erro em relação ao custo incremental ótimo real, esse erro será funçãode μ^2. (YUAN et al, 2017).

2.3. Algoritmo de Difusão

O segundo algoritmo de otimização estudado foi a Difusão. Possui um grau de complexidade na sua implementação maior que o algoritmo de Consenso, porém resolve o problema da atualização assimétrica do algoritmo de Consenso, levando a uma

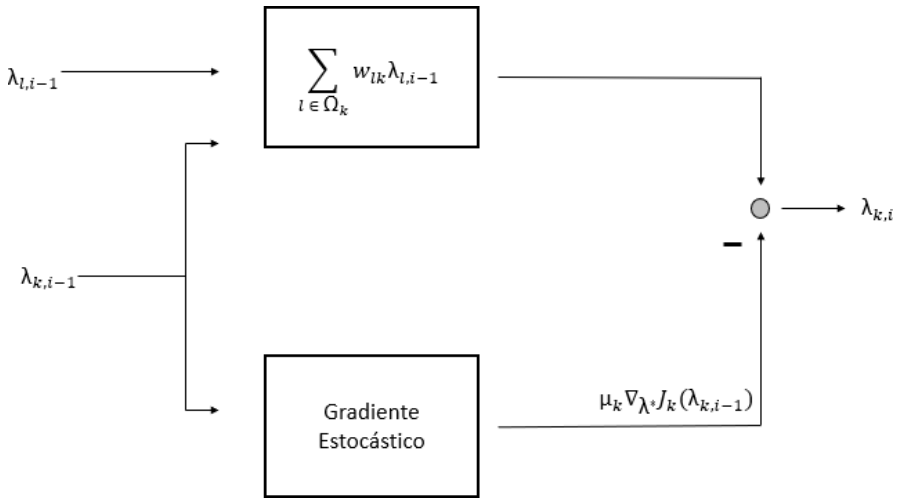
maior estabilidade e, consequentemente, a um menor número de interações necessárias para o algoritmo chegar ao princípio da igualdade marginal (YUAN et al, 2017). Mesmo resolvendo o problema da atualização assimétrica, ambos os algoritmos possuirão um erro associado, que será função de μ_i^2.

A Difusão também é composta das mesmas duas etapas que o Consenso, comunicar e atualizar, porém, corrige o problema da atualização assimétrica do algoritmo de Consenso, separando as etapas de comunicação e de atualização em duas equações diferentes e acrescentando uma variável a mais no algoritmo, a chamada variável de difusão (HE, 2019). A equação (2.9) mostra o equacionamento do algoritmo de Difusão, com suas duas etapas atualização e comunicação explícitas.

$$\begin{cases} \varphi_i(k) = \lambda_i(k-1) - \epsilon_i \nabla_x J_i\big(\lambda_i(k-1)\big), & \textit{etapa de atualização} \\ \lambda_i(k) = \sum_{j \in \Omega_i} w_{ik}\varphi_j(k), & \textit{etapa de comunicação} \end{cases} \tag{2.9}$$

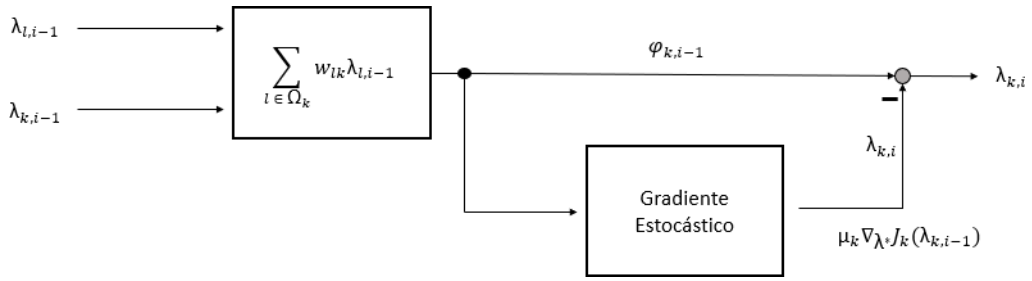
A correção do problema de atualização assimétrica pela Difusão faz com que esse algoritmo seja mais estável, quando comparado ao Consenso, ou seja, o intervalo de variáveis de *feedback* (μ) que fazem o algoritmo de Difusão convergir é maior que o intervalo de variáveis de *feedback* que faz o Consenso convergir. Como essa variável influi no número de interações necessárias para fazer o algoritmo convergir, tem-se que o algoritmo de Difusão necessita de menos interações para convergir que o algoritmo de Consenso. Como o Consenso e Difusão possuem praticamente o mesmo custo computacional por interação, tem-se que o custo computacional final da Difusão será menor que o custo computacional final do Consenso, já que é necessário menos interações para convergir. A diferença entre o Consenso e a Difusão é melhor explicitada através dos diagramas de blocos mostrados na Figuras 5 e 6.

Figura 5:Diagrama de blocos do algoritmo de Consenso.



Fonte: (SAYED,2014)

Figura 6:Diagrama de blocos do algoritmo de Difusão.



Fonte: (SAYED,2014)

Assim como no Consenso, também foi implementado um algoritmo genético para otimizar o custo computacional através da variável μ_i de cada agente. Nesse AG a função *fitness* é o número de interações final do algoritmo, e a variável utilizada para o cruzamento é o vetor μ, com um μ_i para cada agente. Dessa maneira, consegue-se melhores resultados quanto ao número de interações final do algoritmo (WANG, 2018). Além disso, foi utilizado um cruzamento por média aritmética entra dois valores do vetor μ, e a mutação escolhida foi alterar aleatoriamente um valor do vetor μ.

2.4. Algoritmo de Difusão Exata

O último algoritmo abordado é o da Difusão Exata. Apesar de possuirpraticamente o mesmo custo computacional por interação que os outros dois algoritmos, é o algoritmo com maior dificuldade para ser implementado, sendo pouco abordado em estudos que falam sobre DEO com arquitetura distribuída.

Tanto o algoritmo de Consenso como o de Difusão apresentam um erro associado ao seu resultado final em comparação com o melhor *global optima* real da microrrede, esse erro é em função de μ^2 (YUAN et al, 2017). O erro é causado por algum nível de ruído, que pode ser encontrado mesmo na Difusão (HE, 2019).

O algoritmo de Difusão Exata, tal como o nome sugere, corrige esse erro associado, levando o resultado a ser exatamente o *global optima* (HE, 2019). Esse algoritmo o faz ao adicionar uma etapa de correção, resultando então em um algoritmo de 3 etapas: atualizar,comunicar, corrigir (ACC). O equacionamento da Difusão Exata explicitando suas etapas é mostrado em (2.10).

$$\begin{cases} \varphi_i(k) = \lambda_i(k-1) - \epsilon_i \nabla_x J_i\big(\lambda_i(k-1)\big), & \textit{etapa de atualização} \\ \lambda_i(k) = \sum_{j \in \Omega_i} w_{ik} \Phi_j(k), & \textit{etapa de comunicação} \\ \Phi_i(k) = \varphi_j(k) + \lambda_i(k) - \varphi_j(k-1), & \textit{etapa de correção} \end{cases} \tag{2.10}$$

Como a etapa de correção possui um custo computacional praticamente nulo, tem-se que o custo computacional por interação da Difusão Exata será praticamente o mesmo que o custo computacional por interação da Difusão (YUAN et al., 2017). Porém as 3 etapas (ACC) da Difusão Exata dão a ela uma maior estabilidade, ou seja, um maior número de μ fará com que o algoritmo convirja, sendo assim, maiores as chances dea Difusão Exata necessitar de menos interações para convergir.

Outro fator que influencia na velocidade de convergência é a matriz de pesos utilizada. A Difusão Exata aceita matrizes left-stochastic enquanto a Difusão aceita somente matrizes *doubly-stochastic* (YUAN et al., 2017). Como mencionado anteriormente, as matrizes *doubly-stochastic* são matrizes *left-stochastic*, mas o contrário não é verdadeiro, e, portanto, o número de matrizes de pesos que podem ser usadas na Difusão Exata é maior que o número de matrizes de pesos que pode ser utilizada na Difusão. Sendo assim, chega-se à conclusão de que há uma maior possibilidade do número de interações finais da Difusão Exata ser menor que o número de interações da Difusão, tanto pelo fato da Difusão Exata permitir um número maior devariáveis de *feedback*, como também porque ela permite um número maior de matrizes de pesos.

A matriz que mostrou melhor desempenho em relação ao número final de interações na Difusão Exata foi a matriz *Averaging*. Outro benefício dessa matriz é que ela leva a um grau de privacidade maior, uma vez que, para sua construção (Eq. 2.5) os agentes não necessitam compartilhar com os vizinhos informações consideradas privadas, necessitando somente compartilhar a variável de correção Φi (que é uma variável sem nenhum sentido físico) e o *mismatch* a cada interação. Assim, o algoritmo de Difusão Exata possui um grau de privacidade alto.

2.5. Sistemas Multiagentes

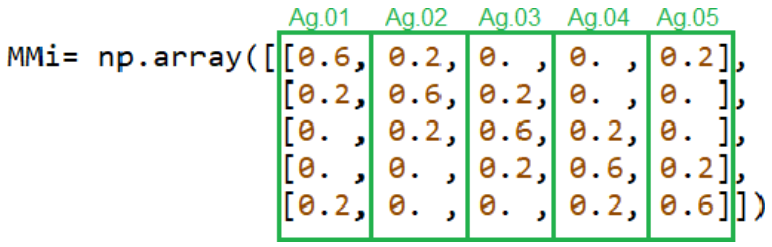
Sistema Multiagente é um sistema composto da interação entre múltiplos agentes. Um agente é uma entidade de software ou hardware que opera de forma autônoma e é capaz de interagir com seu ambiente, e, a cada interação, adaptar seu estado e comportamento com base nessa interação. Cada agente tem uma função específica visando de forma conjunta atingir um objetivo geral comum a todos, constituindo assim, o que é chamado de sistema multiagente (PASCUTTI, 2002).

Uma das características mais importantes de um agente é sua capacidade de periodicamente, executar interações para compartilhar informações, e, a partir delas, realizar tarefa. Essa interação multiagente possui três elementos chaves para ser alcançada: uma linguagem e um protocolo de comunicação que lhes sejam comuns; um formato comum para o conteúdo de comunicação é uma ontologia compartilhada

(PASCUTTI,2002). Todos esses elementos chaves são implementados na segunda parte deste trabalho seguindo protocolos do padrão FIPA (seção 2.5.1).

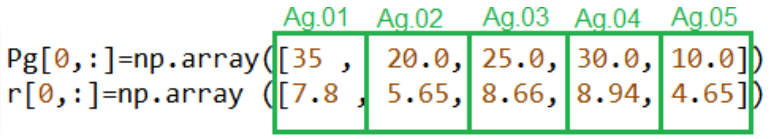
Até agora os 3 algoritmos desenvolvidos fizeram uso de um modelo matricial para modelagem da rede de comunicação. A matriz de pesos representa a rede de comunicação, e cada elemento da matriz representa a comunicação entre dois agentes. Além disso, cada agente é representado por uma coluna dessa matriz de comunicação, e as diversas características dos agentes tais como, os valores iniciais dos custos incrementais de cada agente ou a potência inicial de cada agente estão sendo representados por vetores diferentes, onde cada coluna do vetor representa um agente. As Figuras 7 e 8 ilustram a representação matricial.

Figura 7:Representação da matriz de comunicação na forma matricial.



Fonte: Próprio Autor

Figura 8:Representação das características dos agentes na forma matricial.



Fonte: Próprio Autor

Porém, desenvolver os algoritmos usando esse modelo matricial é uma etapa útil para realizar estudos que irão otimizar o número de interações necessárias para os algorimos convergirem e realizar comparações entre os algoritmos, mas não poderia ser implementado em uma situação prática, em que fosse necessário utilizar um algoritmo distribuído para realizar o DEO. Isso se dá porque alémde desenvolver o algoritmo, necessita-se também desenvolver o agente e os protocolos de comunicação entre esses agentes ao invés de usar matrizes e vetores para representá- los, o que será feito na segunda etapa, onde será desenvolvido o esquema multiagente.

Além da troca da matriz de comunicação pelos protocolos de padrão FIPA, outra grande diferença entre o modelo matricial e o modelo multiagente propriamente dito é a

troca da maneira como que se armazena as informações específicas de cada agente na maneira matricial. Como mostrado na Figura 2.3, cada tipo de informação será um vetor de tamanho igual ao número de agentes. O esquema matricial armazena as informações de custo incremental inicial e de potência que cada agente fornece para a rede inicialmente. Já no SMA cada informação é guardada dentro de cada agente dentro de uma variável específica. Tal como mostra a Figura 9.

Figura 9:Representação das características dos agentes no PADE.

```
class Equipment(Agent):

    def __init__(self, aid, ag_pub, dts):
        super(Equipment, self).__init__(aid=aid, debug=False)
        self.mosaik_sim = MosaikSim(self)
        display_message(self.aid.localname, 'Agente sendo executado!')

        # entradas
        self.tinic = dts['hour_in']
        self.tcanc = dts['hour_out']
        self.GPS = dts['GPS']
        self.ID=dts['ID']
        self.alp=dts['alpha']
        self.bet=dts['beta']
        self.epil=dts['epil']
        self.pren=dts['pren']
        self.lim=dts['lim']
        self.w=dts['w']
        self.u=dts['u']
        self.Pg=0
        #self.Pg=0
        self.n_algorit=0

        #entradas 'calculadas'

        if self.ID == 0:
            self.r=self.w
        elif self.ID == 1:
            self.r = 0
        elif self.ID == 2 or self.ID == 3:
            self.r = 2*self.alp*self.Pg +self.bet
        self.Pd=self.pren #Power Deviation = Power Mismatch
```

Fonte: Próprio Autor

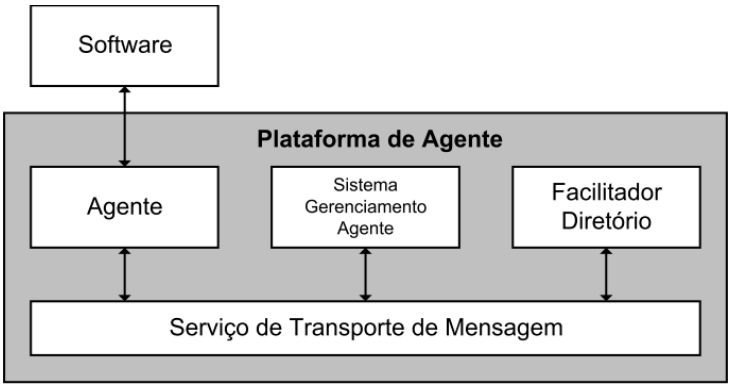
2.5.1. Padrão FIPA

A FIPA (*Foundation for Intelligent Physical Agents*) é uma organização internacional formada por universidades e empresas que atuam no desenvolvimento de aplicações de sistemas baseados em agentes, com o objetivo de criar padrão para o desenvolvimento de softwares baseados em agentes (PASCUTTI, 2002). Visando a padronização e a interoperabilidade entre agentes, a FIPA propôs especificações tais como:

- Arquitetura Abstrata FIPA, que irá promover a interoperabilidade e a reusabilidade através da identificação de elementos codificados da arquitetura.
- *Agent Communication Language* (ACL), que irá detalhar a sintaxe e a semântica de uma linguagem de comunicação de alto nível, que é baseada em atos de fala
- Gerenciamento de Agentes FIPA: que irá tratar o que é necessário para gerenciar agentes dentro de uma plataforma de agentes (PA).

A FIPA considera uma plataforma de agentes um elemento chave na arquitetura de um agente, pois, somente através de uma PA, um agente poderá interagir com outros, seja na mesma plataforma ou em plataformas diferentes, e, dessa maneira, os agentes poderão interoperar e serem gerenciados. A FIPA estabelece que uma PA consiste no mínimo de um Sistema de Gerenciamento de Agentes, um Facilitador de Diretório e umSistema de Transporte de Mensagem, como ilustrado na Figura 10 (PASCUTTI, 2002).

Figura 10:Modelo de referência FIPA para uma PA.



Fonte: (PASCUTTI, 2002)

Nesse modelo FIPA de referência de uma PA, tem-se:

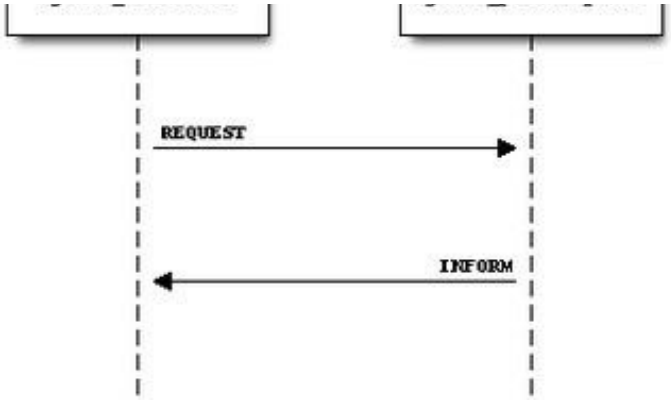
- Software: Instruções executáveis que são acessadas por agentes; agentes acessam softwares para, por exemplo, adicionar novas tarefas, adquirir novos protocolos de comunicação e acessar novos algoritmos de segurança.
- Agente: Principal componente de uma plataforma PA, inteligência artificial que combina uma ou mais tarefas em um único modelo de execução integrado.

- Sistema de Gerenciamento de Agente: Responsável por supervisionar o uso da PA e de manter um diretório de identificadores de agentes.
- Facilitador Diretório (FD): Provê serviço de páginas amarelas para outros agentes, ou seja, os agentes registram a descrição de seus serviços no FD para que outros agentes possam consultar o FD e descobrir seus serviços.
- Serviço de Transporte de Mensagem (STM): Distribui mensagens entre agentes de uma mesma PA ou de diferentes PAs. Todos os agentes possuem acesso a pelo menos uma STM e somente mensagens endereçadas para um agente poder ser enviadas para o STM.

Para se realizar o Serviço de Transporte de Mensagem, necessita-se de Protocolos de Transporte de Mensagem (PTM). A FIPA padroniza alguns desses protocolos, entre eles pode-se citar o protocolo FIPA-*Request*, o FIPA- *Contract-Net* e oFIPA-*Subscribe*.

O protocolo FIPA-*Request* é a maneira mais simples de se implementar a comunicação entre dois agentes. Basicamente, no protocolo FIPA-*Request*, um agente manda uma mensagem requerendo uma informação de outro agente, o agente querecebe essa mensagem irá mandar uma mensagem de *Inform*, com as informações requeridas para o primeiro Agente (MELO, 2018). A Figura 11 ilustra como funciona esse processo.

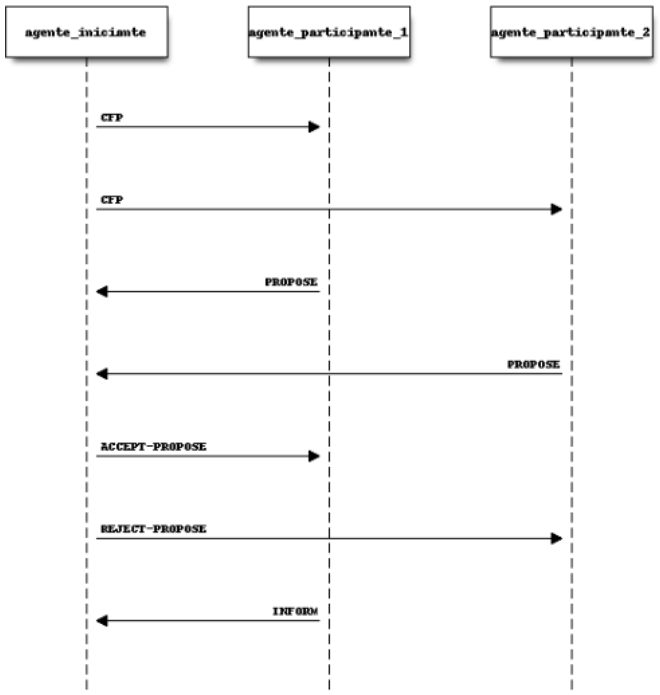
Figura 11:Ilustração do Protocolo FIPA-Request.



Fonte: (MELO,2016)

O protocolo FIPA-*Contract-Net*, como o próprio nome diz, estabelecerá uma negociação entre dois ou mais agentes, nesse caso, um agente contratante recebe as propostas dos agentes participantes, e, a partir delas, escolhe a mais viável. Após a decisão ele mandará a mensagem aceitando a proposta do agente participante escolhido, e também envia mensagens negando as propostas dos outros agentes participantes. O agente participante escolhido irá então mandar uma mensagem de *Inform* para o contratante. A Figura 12 ilustra uma negociação entre agentes através do protocolo FIPA-*Contract-Net* (MELO, 2018).

Figura 12:Protocolo FIPA-Contract-Net.



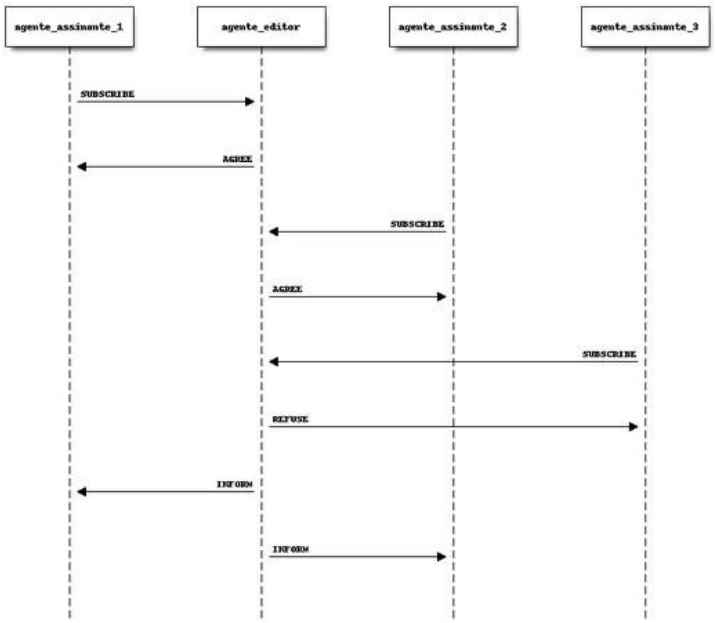
Fonte: (MELO,2016)

Já no protocolo FIPA-*Subscribe*, tem-se o agente editor e os agentes assinantes. Sempre que um agente assinante tiver interesse em uma informação transmitida pelo agente editor, ele pode mandar mensagem para esse editor (mensagem de *Subscribe*), e o editor irá mandar uma mensagem, ou concordando ou negando a assinatura (mensagem de *Agree* ou *Refuse*). Caso concorde o agente editor passa a enviar periodicamente uma informação para esse agente assinante (mensagem de *Inform*). O

agente editor pode possuir mais de um agente assinante (MELO, 2018). A Figura 13 ilustra como funciona o protocolo FIPA-*Subscribe*.

Nesse trabalho o protocolo FIPA-Request será utilizado para realizar a comunicação entre os agentes da microrredes, além disso, também terá um agente *Plug and Play*, que irá realizar as conexões entre os agentes e também ajustar a rede caso ocorra alguma falta que impossibilite um agente de se comunicar com os demais. Para o agente Plug and Play, será utilizado o protocolo FIPA-*Subscribe*.

Figura 13:Protocolo FIPA-Subscribe.



Fonte: (MELO,2016).

2.5.2. Plataforma PADE

Para realizar o desenvolvimento dos agentes foi utilizada a plataforma PADE (*Python Agent DEvelopment*), um *framework* onde é possível desenvolver o sistema multiagente utilizando a linguagem de programação Python.

O PADE foi desenvolvido tendo em vistas os requisitos para sistema de automação e oferece os seguintes recursos em sua biblioteca para o desenvolvimento de sistemas multiagentes (MELO, 2018):

- Orientação a Objetos: Irá fornecer abstração para a construção de agentes e seus comportamentos utilizando conceitos de orientação a objetos;
- Ambiente de execução: Módulo para inicialização do ambiente de execução de agentes em código Python;
- Mensagens no padrão FIPA-ACL: Módulo para a construção e tratamento de mensagens no padrão FIPA -ACL;
- Filtragem de Mensagens: Módulo para filtragem de mensagens;
- Protocolos FIPA: Módulo para a implementação dos protocolos definidos pela FIPA;
- Comportamentos Cíclicos e Temporais: Módulo para implementação de comportamentos cíclicos e temporais;
- Banco de Dados: Módulo para interação com banco de dados;
- Envio de Objetos Serializados: Possibilidade de envio de objetos serializados como conteúdo de mensagens FIPA-ACL;
- Integração com o mosaik: Integração com a API mosaik, utilizada na sincronização de Sistemas Multiagentes.

Ao implementar os algoritmos no sistema multiagente, substitui-se a matriz utilizada para representar a rede de comunicação dos vários agentes da rede pelos protocolos de padrão FIPA. Neste trabalho foram implementados dois dessesprotocolos: protocolo FIPA-*Request* e o protocolo FIPA-*Subscribe*. O primeiro protocolo é utilizado entre os agentes para demandar informações dos agentes vizinhos. Alguns exemplos de informações demandadas podem ser: custo incremental em cada interação, número de agentes com que o vizinho se comunica e variável de difusão. O segundo protocolo é utilizado por um agente chamado de agente *Plug-and-Play*. Esse agente será responsável por montar a rede de comunicação com base na informação de posição física recebida via protocolo FIPA-*Subscribe*, conectando um agente aos seus dois vizinhos mais próximos, um a direita, outro a esquerda. Além disso, o agente *Plug-and- Play* terá um comportamento temporal onde, a cada 10 segundos irá checar se algum agente entrou ou saiu da rede, sendo, portanto, responsável por reestruturar a rede caso um novo agente seja conectado a ela ou um agente saia dela. Devido a esse

comportamento, de se comunicar com todos os agentes da rede e receber desses agentes a informação da posição espacial, o agente *Plug-and-Play* é considerado uma centralização, porém, é importante destacar que ele terá participação mínima na realização do DEO em uma microrrede, uma vez que existe pouca modificação na estrutura de agentes que gerenciam a microrrede, ou seja, muito raramente um agente irá entrar ou sair da rede. Porém, em problemas em que a rede precisa de reestruturar frequentemente, como por exemplo, no carregamento dinâmico de um estacionamento de carros elétricos, esse agente teria uma participação significativa.

O tipo de rede que será montado será uma rede em anel, onde os agentes se comunicarão de tal forma que a rede de comunicação forme um círculo fechado. A rede anel utilizada está representada na Figura 14.

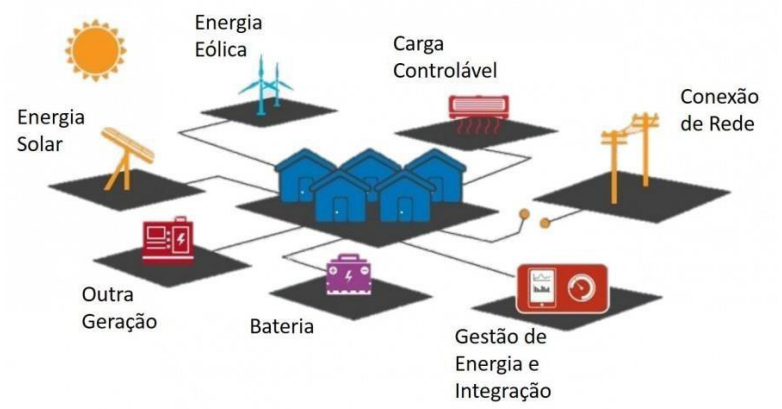
Figura 14:Rede anel utilizada no Sistema Multiagente.

Fonte: HAESUM et al, 2017.

3. Modelos de Despacho Econômico Ótimo

Em uma microrrede, normalmente tem-se uma variedade de recursos energéticos que precisam ser gerenciados, tais como: geradores despacháveis, com diferentes custos de operação; geradores não despacháveis, como aerogeradores e plantas PV; e bancos de baterias, utilizados para armazenar energia por exemplo em horários que a carga esteja mais leve ou a energia mais barata e utilizá-la em horários que a carga esteja mais pesada ou a energia mais cara, ou mesmo quando não há radiação solar. Além disso, tem-se também que a microrrede pode operar de dois modos: conectada à rede de distribuição ou isolada dela (HAESUM et al, 2019).

Figura 15:Elementos de uma microrrede.



Fonte: (LEGADO ENERGIA, 2020)

Com isso, tem-se que uma das funções de uma microrrede é gerenciar todos os seus recursos para garantir o suprimento da carga com o menor custo possível, obedecendo às restrições técnicas operativas e normativas, esse procedimento é chamado de Despacho Econômico Ótimo (WANG et al., 2018). O DEO normalmente é feito 24 horas antes, com os dados de previsão de carga e de geração das fontes não despacháveis, sendo realizado um ajuste fino durante o dia para corrigir erros nessa previsão e garantir o suprimento de 100% da carga (HAESUM et al., 2019).

O DEO pode ser realizado de duas maneiras: a primeira é quando a geração é maior do que a carga; nesse caso o sistema consegue suprir a carga sem necessitar de

um corte de carga. Sendo assim, o DEO irá se encarregar de suprir a carga com o menor custo. O outro caso é quando a carga é maior do que a geração, nesse caso o DEO terá que realizar um corte de carga.

O custo ótimo de uma microrrede será atingido quando o princípio da igualdade marginal for atendido, ou seja, quando a equação (3.1) for atendida por todos os agentes que fazem parte de seu despacho econômico ótimo (PYNDICK, 2006).

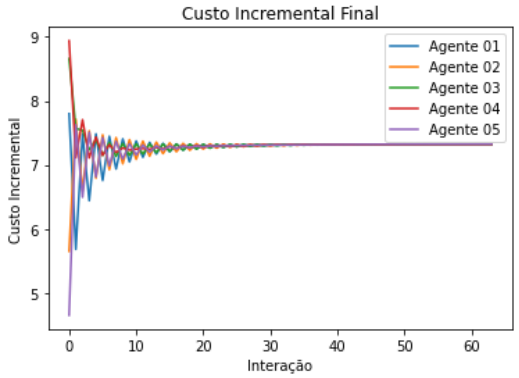
$$\lambda_{ger} = \lambda_{bat} = \lambda_{ren} = B_{carg} \tag{3.1}$$

Em que:

- λ_i:Custo incremental do agente i
- B_{carg}: Benefício incremental do agente carga

Os algoritmos usados neste trabalho possuem a função de buscar essa convergência entre os agentes, consequentemente, atingindo o DEO. A Figura 16 mostra esse processo de convergência em uma microrrede composta por 5 geradores despacháveis. Quando o custo incremental atinge o valor de 7,34 [$/kW], tem-se o valor ótimo de operação da microrrede.

Figura 16:Convergência do custo incremental através do algoritmo de consenso.



Fonte: O Próprio Autor

3.1. Modelagem dos Agentes da Microrrede

Para cada recurso energético da microrrede mostrada na Figura 2.10, definiu-se uma função custo correspondente, conforme especificado a seguir.

- Fontes despacháveis (Gerador a Diesel, Gerador a Gás): Tem sua função custo representada por uma função quadrática em função da potência gerada. Além disso,

existem limites mínimos e máximos de potência gerada que devem ser respeitados nesse tipo de fonte. A Equação 3.2 mostra um exemplo de função custo para fontes despacháveis e a Equação 3.3 mostra seus limites (HAESUM et al, 2019).

$$\mathrm{F}(\mathrm{P}_{ger}) = \alpha P_{ger}^2 + \beta P_{ger} + \gamma \qquad (3.2)$$

$$P_{ger_{min}} \leq P_{ger} \leq P_{ger_{max}} \qquad (3.3)$$

Em que:

- P_{ger}*:* Potência usada pelo gerador despachável.
- α, β *e* γ*:* Parâmetros específicos de custo de cada gerador.
- $P_{ger_{min}}$: Potência mínima a ser gerada pelo gerador.
- $P_{ger_{max}}$*:* Máxima potência que o gerador pode gerar.

- Fontes não despacháveis (PV; Aerogerador): Possuem custo 0 de geração embora existem custo atrelados à manutenção e perdas de equipamentos, porém os mesmo são desconsiderados. A energia gerada pelas fontes renováveis não despacháveis é tratada como se fosse uma carga negativa. A equação (3.4) mostra um exemplo de função custo para fontes não-despacháveis (HAESUM et al., 2019).

$$F(P_{ren}) = 0 \qquad (3.4)$$

Em que:

- P_{ren}*:* Potência usada pelo gerador não-despachável.

- Bateria: Possui um custo de degradação atrelado a sua utilização, que é considerado quando a bateria está sendo carregada, bem como descarregada. Esse custo também é representado por uma função quadrática em função da potência. Além disso, a bateria também possui limites de potência para a carga e para a descarga (HAESUM et al., 2019).

$$\beta(P_{bat} + 3\ P_{bat_{max}}(1 - \mathrm{SOC})) + \alpha\,(P_{bat} + 3\,P_{bat}\ \ (1 - SOC))^2 + \gamma \qquad (3.5)$$

$$P_{bat_{min}} \leq P_{bat} \leq P_{bat_{max}} \qquad (3.6)$$

Em que:

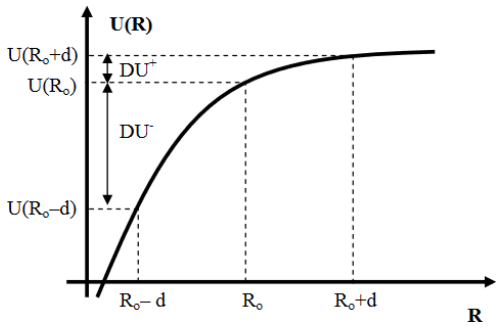
- P_{bat}: Potência entregue ou consumida pela bateria
- $P_{bat_{max}}$: Máxima potência que pode ser entregue pela bateria.
- $P_{bat_{min}}$: Mínima potência que pode ser entregue pela bateria.
- α, β e γ: Parâmetros específicos de custo de cada bateria.
- SOC: estado de carregamento da bateria, que varia de 0 (totalmente descarregada) até 1 (totalmente carregada).

- Cargas: O gerenciamento da microrrede pode ser obtido com o corte inteligente de carga. Uma das maneiras de realizá-lo é estabelecendo uma relação de prioridade entre as cargas. Primeiramente elimina-se as cargas com menor prioridade, e somente se o sistema não conseguir suprir a carga remanescente é que se elimina as cargas mais prioritárias. Tal processo é chamado de corte de carga inteligente.

Para estabelecer essa relação de prioridade das cargas, utiliza-se uma função de utilidade que modela o grau de satisfação de um consumidor em consumir determinada quantidade de um produto. A função utilidade deve ser definida de tal forma que, as cargas que possuem maior prioridade sempre possuam um valor de utilidade maior que cargas ditas não prioritárias. Dessa maneira, o algoritmo do corte de carga inteligente, que terá como função objetivo maximizar a função utilidade, sempre irá optar por primeiramente atender as cargas prioritárias (HAESUM et al., 2019).

A função utilidade é modelada de tal forma que, até determinado ponto ela é uma função quadrática, após determinado nível, pode-se dizer que o cliente já está satisfeito com seu consumo, e consumi-lo mais não é de interesse do cliente (PYNDICK, 2006), sendo assim, depois desse ponto, a função se torna paralela ao eixo x, tal como mostra a Figura 17. Isso ocorre porque a partir de uma certa quantidade disponível de um produto, não existe mais benefício em aumentar sua disponibilidade. Tal quantidade disponível é calculada como sendo seu coeficiente “a”, da função quadrática, dividido pelo seu coeficiente “b”.

Figura 17:Exemplo de Função Utilidade



Fonte: (AGUIAR,2008)

A equação (3.7) define a função custo para a carga prioritária e não-prioritária.

$$F(P_{Car}) = \begin{cases} w.(P_{Carg} - \left(\frac{v}{2}\right) P_{carg}^2, se\ 0 \leq\ P_{Carg} \leq {}^{w}/_{v} \\ {w^2}/_{v}\ \ , se\ P_C \geq {}^{w}/_{v} \end{cases} \quad (3.7)$$

Em que:

- P_{carg}: Potência consumida pela carga
- w e v: Parâmetros de utilidade pré-estabelecidos da carga, é através do valor deles que se configura a Carga Prioritária e a Não-Prioritária.

No DEO será assumido que todos os geradores despacháveis e a bateria fornecem o máximo nível de potência possível para a rede.

Para encontrar o custo ótimo em uma microrrede utilizando as equações previamente descritas, deve-se utilizar do operador de Lagrange em cada uma delas,com isso, tem-se (HAESUM et al., 2019):

- Geração Despachável:

$$\frac{\nabla F(P_{ger})}{\nabla P_{ger}} = \frac{\nabla\ (\alpha\ (P_{ger})^2\ +\ \beta\ P_{ger} + \gamma)}{\nabla\ P_{ger}}$$

$$\lambda_{ger} = \frac{\nabla F(P_{ger})}{\nabla P_{ger}} = 2 * \alpha^* P_{ger} + \beta \quad (3.8)$$

- Geração Não-Despachável:

$$\frac{\nabla F(P_{ren})}{\nabla P_{ren}} = \frac{\nabla (0)}{\nabla P_{ren}}$$

$$\lambda_{ren} = \frac{\nabla F(P_{ren})}{\nabla P_{ren}} = 0 \quad (3.9)$$

- Bateria:

$$\frac{\nabla F(P_{bat})}{\nabla P_{bat}} = \frac{\nabla \left(\beta(P_{bat} + 3\ P_{bat_{max}}(1 - \mathrm{SOC}) + \gamma(P_{bat} + 3\,P_{bat_{max}}(1 - SOC)^2) + \alpha\right)}{\nabla P_{bat}}$$

$$\lambda_{bat} = \frac{\nabla F(P_{bat})}{\nabla P_{bat}} = \beta + \alpha(2P_{bat} - 6P_{bat_{max}}(\mathrm{SOC} - 1)) \quad (3.10)$$

- Carga Prioritária e Não Prioritária:

$$\nabla \frac{F(P_{carg})}{\nabla P_{carg}} = \frac{\nabla (w.(P_{carg}) - \left(\frac{v}{2}\right).P_{carg}^2)}{\nabla P_{carg}}$$

$$B_{carg} = \frac{\nabla F(P_{carg})}{\nabla P_{carg}} = w - v.P_{carg} \quad (3.11)$$

As variáveis, λ_{gen}, λ_{ren} e λ_{bat}, obtidas através de um processo de derivação das funções de custo, são chamadas de custo incremental, e corresponde ao custo atrelado em se consumir mais uma unidade de um determinado produto. Como a função de custo total é uma função quadrática, o custo incremental é uma função linear. Já no caso das Cargas, tem-se a função Benefício incremental, que é a derivada da função utilidade e representa o grau de satisfação do cliente em consumir mais uma determinada unidade de determinado produtor, no caso, a energia. O benefício incremental irá aumentar linearmente até determinado ponto, que será o valor de w/v, e

então irá ser igual a 0, indicando assim que não existe mais benefício em acrescentar a demanda de determinado produto em mais uma unidade (HAESUM et al., 2019).

3.2. Modelagem do DEO usando o Consenso

Para que o Consenso realize o DEO em uma microrrede, ele deverá passar por algumas adaptações, sendo reescrito como mostra a equação (3.12). Comparando essa equação com a (2.7), tem-se que a variável de interesse a ser otimizada é o custo incremental global da microrrede, a variável interna s_i é o *power mismatch* (diferença dos valores de potência obtidos na interação passada e na interação atual do agente e de seus vizinhos) de cada agente na interação i e p_i será a potência de cada agente na interação i. Além disso, as equações de (3.8) até (3.11) foram utilizadas para inicializarem o custo incremental de seus respectivos agentes, a potência inicial e o *power mismatch* de cada agente é 0. A etapa de comunicação foi posta em negrito na eq.(3.12), e o que não está em negrito refere-se à etapa de atualização.

$$\begin{cases} \lambda_i(k+1) = \sum_{\boldsymbol{j} \in \Omega_i} \boldsymbol{w_{ij}} \boldsymbol{\lambda_j}(\boldsymbol{k}) + \mu_i(k) s_i(k) \\ p_i(k+1) = \frac{\lambda_i(k+1) - \beta_i}{2\alpha_i} \\ s_i(k+1) = \sum_{\boldsymbol{j} \in \Omega_i} \boldsymbol{w_{ij}} \boldsymbol{s_j}(\boldsymbol{k}) - [p_i(k+1) - p_i(k)] \end{cases} \tag{3.12}$$

Outro fator importante é que na microrrede cada agente possui limites mínimos e máximos de potência que devem ser obedecidos para que operem apropriadamente, esses limites são representados em (3.3), (3.6) e a segunda equação de 3.7. Uma vez queo algoritmo converge, ou seja, a equação 3.1 é obedecida, o código irá checar se os limites de potência em cada agente são obedecidos. Caso sejam obedecidos, o DEOpode ser dito como finalizado, caso não sejam obedecidos, o algoritmo irá ajustar o agente que ultrapassou seu limite de potência para o seu valor máximo ou mínimo e rodar novamente, deixando fixa a potência desse agente em todas as suas interações (WANG, 2018). No código, essa etapa é feita por meio de *flags* de limite de potência. Caso alguma *flag* seja acionada, o algoritmo irá rodar novamente, esse processo continua até que nenhuma *flag* seja acionada. A equação (3.13) mostra como se

comportam as potências de cada agente nesse caso em que o limite de potência de algum agente é ultrapassado, o custo incremental (λ_i) e o desvio de potência (s_i) se comportam da mesma maneira que na equação (3.12).

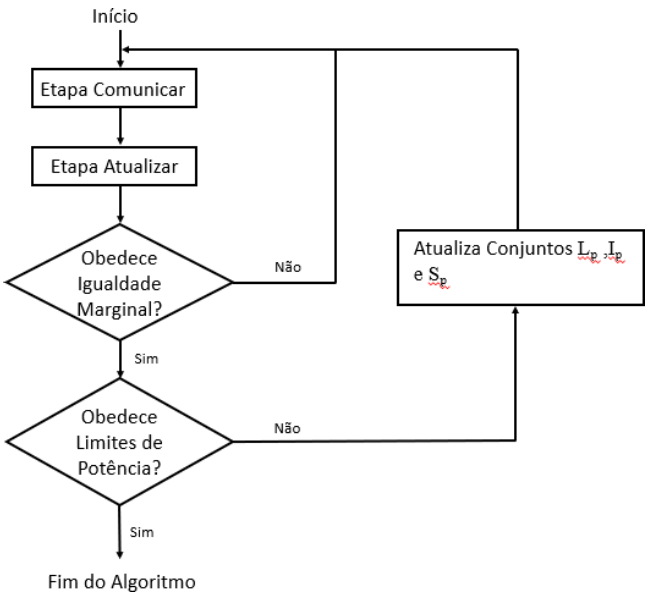
$$p_i^* = \begin{cases} \frac{\lambda_i(k+1) - \beta_i}{2\alpha_i}, se\ i\ \in L_p \\ p_{min_i}, se\ i\ \in I_p \\ p_{max_i}, se\ i\ \in S_p \end{cases} \quad (3.13)$$

Em que:

- L_p: Conjunto dos agentes que obedeceram aos limites de potência;
- I_p: Conjunto dos agentes que ultrapassaram o limite de potência inferior;
- S_p: Conjunto dos agentes que ultrapassaram o limite de potência superior;
- p_{min_i}: Limite de potência inferior do agente i;
- p_{max_i}: Limite de potência superior do agente i.

O algoritmo de Consenso é resumido no Fluxograma da Figura 18.

Figura 18:Fluxograma do algoritmo de Consenso.



Fonte: O Próprio Autor

Uma das desvantagens do Consenso quando aplicado para DEO de microrredes é que ele irá comunicar o custo incremental e o valor do *mismatch* de cada um dos agentes com os agentes vizinhos, o que diminui a privacidade dos participantes do processo de otimização.

3.3. Modelagem do DEO usando a Difusão

Para o DEO em uma microrrede, o algoritmo de Difusão é adaptado tal como mostra a equação (3.14). Como esse algoritmo apresenta melhor distinção entre quais equações fazem parte da etapa de comunicação e quais equações fazem parte da etapa de atualização, as mesmas foram descriminadas ao lado da equação. Esse algoritmo também irá rodar várias vezes até que todos os limites de potência de todos os agentes sejam obedecidos, logo tanto o fluxograma da Figura 3.4 como o equacionamento de limites de potência mostrado na Eq. (3.13) também são válidos para esse algoritmo.

$$\begin{cases} \varphi_i(k) = \lambda_i(k-1) - \epsilon_i \nabla_x J_i\big(\lambda_i(k-1)\big), \ etapa\ de\ atualização \\ \lambda_i(k) = \sum_{j \in \Omega_i} w_{ik}\varphi_j(k), \ etapa\ de\ comunicação \\ p_i(k+1) = \frac{\lambda_i(k+1) - \beta_i}{2\alpha_i}, etapa\ de\ atualização \end{cases} \tag{3.14}$$

Ao observar a Eq. (3.14) observa-se que as variáveis a serem compartilhadas na rede de comunicação são a variável de difusão e o *mismatch*, e que cada agente possui um equacionamento próprio para converter a variável de difusão ($\varphi_i(k)$) no custo incremental. Isso irá prover uma maior privacidade aos agentes da microrrede quando comparado ao Consenso, uma vez que os agentes não necessitam mais compartilhar o custo incremental,e sim uma variável sem nenhum significado físico propriamente dito. Porém, vale ressaltar que a matriz de Pesos utilizada na difusão ainda é a Matriz *Mean Metropolis*, ou seja, é necessário cada agente compartilhar o número de agentes com que ele se comunica. Portanto, vê-se que a Difusão possui um grau de privacidade médio.

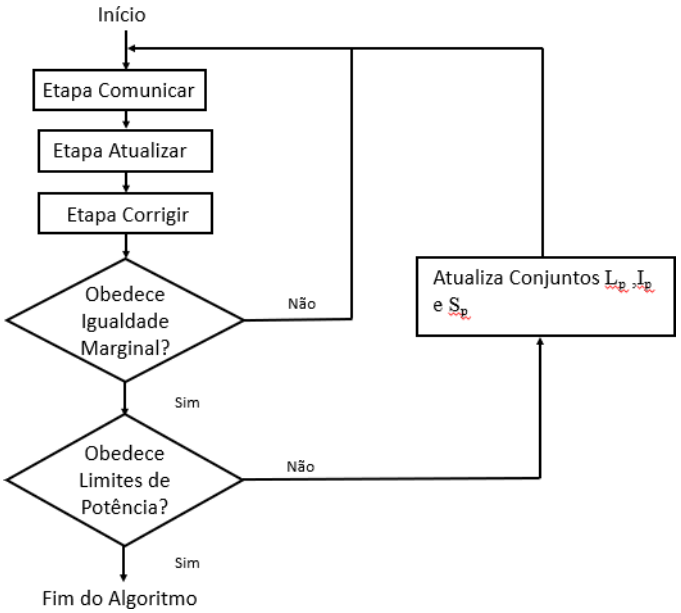
3.4. Modelagem do DEO usando a Difusão Exata

O modelo da Difusão Exata para DEO é modificado em relação ao da equação (2.10) tal como mostra a equação (3.15).

$$\begin{cases} \varphi_i(k) = \lambda_i(k-1) - \epsilon_i \nabla_x J_i\big(\lambda_i(k-1)\big), \ \ etapa\ de\ atualização \\ \lambda_i(k) = \sum_{j \in \Omega_i} w_{ik} \Phi_j(k), \ \ etapa\ de\ comunicação \\ \Phi_j(k) = \varphi_j(\mathrm{k}) + \lambda_i(k) - \varphi_j(\mathrm{k}-1), \ etapa\ de\ correção \\ p_i(k+1) = \ \frac{\lambda_i(k+1) - \beta_i}{2\alpha_i} \end{cases} \tag{3.15}$$

O fluxograma da Difusão Exata mostrado na Figura 19 é diferente do fluxograma dos outros algoritmos pois possui adicionalmente a etapa de correção já explicada na sessão 2.4 deste trabalho. A Difusão Exata compartilha somente a variável de difusão ($\Phi_j(k)$), que por ser uma variável sem sentido físico, não exige nenhum grau de privacidade. Como a Difusão Exta usa a matriz de pesos *Averaging* (Eq. 2.5), não existe necessidade de um agente ter conhecimento de com quantos outros agentes seu vizinho se comunica.

Figura 19:Fluxograma do algoritmo da Difusão Exata.



Fonte: Próprio Autor

3.5. Modelagem do DEO usando SMA

Para a modelagem dos agentes no SMA foram utilizadas as equações descritas na seção 3.1, além disso também foram utilizados os protocolos descritos na seção 2.5 deste trabalho. Porém, ao se implementar o algoritmo no PADE percebe-se que, como cada agente é uma máquina independente que irá se comunicar com as outras através de uma rede de comunicação, cada um deles irá demorar um tempo diferente para processar uma interação do algoritmo. Sendo assim, ao rodar o SMA é comum se deparar com os agentes do SMA rodando o código em uma interação diferente, o que afeta a comunicação entre eles e a convergência para o valor ótimo.

3.5.1. Sincronização da Comunicação dos Agentes

A primeira tentativa de resolver esse problema foi através da função *call_later()* do PADE. Essa função retarda a execução de um determinado trecho do código durante um determinado tempo especificado (MELO, 2019). Para utilizar essa função deve-se especificar o tempo de espera que o agente deve dar para executar o próximo método e qual é o método que deve ser executado logo em seguida após o tempo pré- determinado. A Figura 20 mostra um exemplo de como a função *call_later()* é utilizada. Nesse código o Agente deve esperar 10 segundos antes de executar a função que imprime na tela a mensagem “Hello, I'm an agent!”.

No caso do SMA aplicado ao DEO, utiliza-se a função *call_later()* para pausar a execução do código sempre que uma mensagem é enviada por um agente e sempre que um agente acaba sua etapa de atualização. A primeira pausa garante que todos os agentes recebam a mensagem via protocolo do FIPA-*Request* e iniciem a implementar a etapa de atualização ao mesmo tempo, enquanto a segunda pausa irá frear agentes que executarem a etapa de atualização mais rapidamente, dando tempo aos agentes mais lentos de terminarem de implementar sua etapa de atualização.

Figura 20:Código que utiliza o call_later().

```
from pade.misc.utility import display_message, start_loop, call_later
from pade.core.agent import Agent
from pade.acl.messages import ACLMessage
from pade.acl.aid import AID
from sys import argv

class HelloAgent(Agent):
    def __init__(self, aid):
        super(HelloAgent, self).__init__(aid=aid, debug=False)

    def on_start(self):
        super().on_start()
        self.call_later(10.0, self.say_hello)

    def say_hello(self):
        display_message(self.aid.localname, "Hello, I\'m an agent!")


if __name__ == '__main__':

    agents = list()

    hello_agent = HelloAgent(AID(name='hello_agent'))
    agents.append(hello_agent)

    start_loop(agents)
```

Fonte: (MELO,2018)

O tempo utilizado por cada uma das funções *call_later()* foi descoberto por meio de tentativa e erro, sendo utilizado na primeira função 3 segundos e na segunda 2 segundos. Porém, o tempo utilizado por cada uma dessas funções depende da máquina em que o PADE é executado, sendo assim esses valores só são válidos para a máquinaem que foi testado. Portanto, a função *call_later()* não é eficiente se for desejável ter uma solução geral, que sirva para todas as máquinas, independente do tempo de processamento de cada uma.

3.5.2. Plataforma de Co-simulação Mosaik

Para obter uma simulação mais genérica, que seja independente do tipo de máquina utilizada, foi utilizada a plataforma mosaik, a qual foi integrada com o PADE para servir de base temporal para todos os agentes do SMA. A integração mosaik + PADE dividirá o tempo dos agentes em passos, e somente irá permitir que todos os agentes avancem para o próximo passo quando todos os agentes terminarem de executaro passo anterior.

O mosaik é uma plataforma que pode utilizar simuladores, em uma situação comum para performar uma simulação coordenada de um determinado cenário (Mosaik, 2012). Para realizar uma simulação sincronizada, o mosaik pode: a) Utilizar uma API para os simuladores se comunicarem com o mosaik; b) Implementar manipuladores para diferentes tipos de simuladores; c) Permitir a modelagem de cenários de simulação envolvendo diferentes simuladores; d) Programar a simulação passo a passo de diferentes simuladores, gerenciando a troca de dados. (Mosaik, 2012).

No caso do SMA, utiliza-se uma integração mosaik + PADE para programar a simulação passo a passo entre os agentes. O mosaik será o agente que guarda a informação sobre qual passo a simulação está, e, ao passar para os agentes do SMA, os mesmos executam as ações correspondentes de cada passo, e comunicam ao mosaik, que por sua vez, somente atualizará o passo quando todos os agentes tiverem rodado o passo anterior. A Figura 21 mostra o código da implementação do mosaik no SMA.

Figura 21:Código da Implementação do Mosaik no SMA.

```
import mosaik
from time import sleep
# python mosaik_start.py
if __name__ == '__main__':
    num_agentes=5
    START = 0
    END = 10000
    port=20000
    k=1
    sim_config = {'Interface':{'python':'SimuladorInterface:Interfacex'}}
    sim_config.update({'Agent_posto':{'connect':'localhost:{}'.format(port)}})
    for i in range(num_agentes):
        sim_config.update({'Agente_{}'.format(i+1): {'connect':'localhost:{}'.format(port + (i+1)*k)}})
    world = mosaik.World(sim_config, debug=True)
    # inicializar e instanciar agentes e interfaces logo em seguida
    agen = list()
    for i in range(num_agentes):
        agen.append(world.start('Agente_{}'.format(i+1),eid_prefix='AgEquip',start = START,step_size = 1))
        agen[i] = agen[i].InterfaceAgente()
    agen.append(world.start('Agent_posto',eid_prefix='AgPosto',start = START,step_size = 1))
    agen[i+1] = agen[i+1].InterfaceAgente()
    intf = world.start('Interface', eid_prefix='Interface_', start = START,step_size = 1)
    intf = intf.Interface.create(num_agentes+1)
    for mod0, int0 in zip(agen, intf):
        # world.connect(a, b, 'val_in')
        # world.connect(b, a, 'val_in', time_shifted=True, initial_data={'val_in': 0})
        world.connect(mod0, int0, 'val_in', async_requests=True)
    world.run(until=END)
```

Fonte: Próprio Autor

Um ponto importante a ressaltar é que essa integração permite também a utilização de uma função conhecida como *step_done()*. No código, ao enviar uma

mensagem via protocolo FIPA-*Request*, a execução do algoritmo fica presa em um *return* vazio, ou seja, sem avançar o passo da simulação. Esse passo só é avançado quando o outro agente recebe a mensagem do protocolo, através dessa função *step_done()*. A Figura 22 mostra todos os locais onde o *return* vazio é usado, sendo nas duas primeiras vezes para esperar a chegada da mensagem enviada pelo agente *Plug- and-Play* via protocolo FIPA-*Subscribe* para montar a rede de comunicação, enquanto a terceira utilização dessa função é para esperar a chegada das informações enviadas entreos agentes via protocolo FIPA-*Request*. A terceira utilização da função irá ser usada em *loop* até o algoritmo convergir.

Figura 22:Função return vazio.

```
def step(self, time, inputs):

    if time == self.agent.tinic and time != 0:
        self.agent.launch_subscriber_protocol()
        return

    if  self.agent.flag_1req == 0 and len(self.agent.vz) > 0 :
        self.agent.protocol.pedir_numerovz()
        return

    #Aqui começa o consenso, é a partir daqui que eu vou adaptar para o consenso da
    #microrrede
    if self.agent.comece_consenso==1:
       self.agent.modifique=1

       # Ainda tenho que alterar a matriz de pesos
       #processo =0 irá calcular a matriz de pesos
       if self.agent.processo ==0 and self.agent.modifique==1:
            self.agent.calculate_weightMX()
            if self.agent.vez==2:
                self.agent.processo=1 #Será o próximo passo do consenso
                self.agent.modifique=0# só rodará no proximo tempo do mosaik


       #processo=1 irá mandar a mensagem de consenso
       if self.agent.processo ==1 and self.agent.modifique==1:

            self.agent.answer_request.consensus_message()
            self.agent.modifique=0
            return
```

Fonte: Próprio Autor

Já as Figuras 23 e 24 mostram onde no código a função *step_done()* é implementada. A Figura 23 mostra sua implementação dentro do protocolo FIPA-*Subscribe*. O algoritmo pausa ao pedir solicitação do protocolo FIPA-*Subscribe* (isso é feito pelo *return* vazio). Quando o agente *Plug-and-Play* aceita essa solicitação, ele

implementa a função *step_done()* para dar continuidade. Já o *step_done()* mostrado na Figura 24 é para dar continuidade ao código quando ele é pausado por aguardar a aceitação do protocolo FIPA-*Request*.

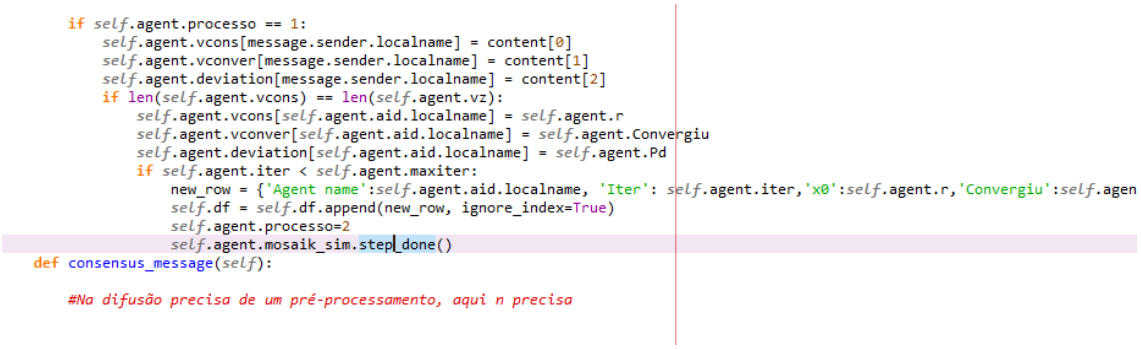
Pode-se afirmar que a plataforma mosaik é uma espécie de centralização, uma vez que ela guarda a informação do passo atual do SMA e se comunica constantemente com todos os agentes do SMA para atualizá-los sobre essa informação de passo. Porémé importante lembrar que o mosaik só irá existir em um ambiente de simulação de uma microrrede, onde todos os agentes rodam via PADE na mesma máquina física, e que o objetivo da utilização dessa plataforma é somente deixar o código mais genérico possível, ou seja, que possa ser executado em qualquer máquina independente de suas características. Em uma aplicação de SMA real, a sincronização poderia ser feita via *call_later()* ou via GPS, com o tempo de espera sendo descoberto via tentativa e erro.

Figura 23:Função step_done() sendo utilizada pelo protocolo FIPA-Subscribe.

```
class SubscriberProtocol(FipaSubscribeProtocol):

    def __init__(self, agent, message):
        super(SubscriberProtocol, self).__init__(agent,
                                                 message,
                                                 is_initiator=True)

    def on_start(self):
        super(SubscriberProtocol, self).on_start()

    def handle_agree(self, message):
        self.agent.mosaik_sim.step_done()

    def handle_inform(self, message):

        if message.sender.localname == 'Agent_posto_20000':
            content = json.loads(message.content)
            ag_ativos = content
            nAG_ativos = len(content)
            num_maxVZ = 2

            # responsável por verificar a quantidade de agentes podem ser meus vizinhos
            if nAG_ativos <= num_maxVZ:
                nVz = nAG_ativos - 1  #N: Creio que é por causa do for mais a frente
                                      #N: Ex: for i in range(0,2): 0,1,2 [seriam 3]
            else:
                nVz = num_maxVZ

            myindex = ag_ativos.index(self.agent.aid.localname)
            ag_ativos.pop(myindex) # a partir daqui só tenho possíveis vizinhos
```

Fonte: Próprio Autor

Figura 24:Função step_done() sendo utilizada pelo protocolo FIPA-Request.



```
        if self.agent.processo == 1:
            self.agent.vcons[message.sender.localname] = content[0]
            self.agent.vconver[message.sender.localname] = content[1]
            self.agent.deviation[message.sender.localname] = content[2]
            if len(self.agent.vcons) == len(self.agent.vz):
                self.agent.vcons[self.agent.aid.localname] = self.agent.r
                self.agent.vconver[self.agent.aid.localname] = self.agent.Convergiu
                self.agent.deviation[self.agent.aid.localname] = self.agent.Pd
                if self.agent.iter < self.agent.maxiter:
                    new_row = {'Agent name':self.agent.aid.localname, 'Iter': self.agent.iter,'x0':self.agent.r,'Convergiu':self.agen
                    self.df = self.df.append(new_row, ignore_index=True)
                    self.agent.processo=2
                    self.agent.mosaik_sim.step_done()
    def consensus_message(self):

        #Na difusão precisa de um pré-processamento, aqui n precisa
```

Fonte: Próprio Autor

Por fim, uma vez implementado o framework mosaik, tem-se um SMA voltado a obter o Despacho Econômico Ótimo de uma Microrrede. O DEO se dará de maneira síncrona, independentemente do tipo de máquina que se utiliza para rodar a simulação.

4. Estudos de Caso

Nesse capítulo mostra-se alguns cenários de operação de uma microrrede com o intuito de testar os algoritmos desenvolvidos, tanto no caso matricial quanto no caso SMA. Primeiramente, são testados 3 casos diferentes para o caso matricial: com carga leve, com carga média e com carga pesada. Nesse último caso o sistema será levadoa fazer um corte de carga inteligente. Esses casos foram testados em todos os três algoritmos desenvolvidos: Consenso, Difusão e Difusão Exata.

Como informado no capítulo anterior, a microrrede teste possui 5 agentes: um agente Geração Despachável, um agente Geração Não-Despachável, um Agente Bateria e dois agentes Carga, tendo uma Carga maior benefício Marginal que a outra, logo maior prioridade em ser atendida. Os parâmetros gerais dos agentes, que são utilizados em todos os casos estão mostrados na Tabela 4.1. Foi adotado que valores de potência menores que 0 significam que o agente está fornecendo energia para o sistema e valores maiores que 0 significam que o agente está demandando energia do sistema. Os parâmetros dos agentes Carga variam segundo o cenário de avaliação.

Tabela 4.1 – Parâmetros da microrrede utilizada para fazer os testes.

Tipo de Agente	Variável	Valor
Despachável	α	0,18
	β	97
	p_{min_i}	0
	$\lvert p_{max_i} \rvert$	50
Bateria	α	2
	β	40
	SOC_{min}	0,1

Fonte: Adaptado pelo Autor

Neste trabalho foi utilizado um algoritmo genético (AG) para encontrar o melhor conjunto de μ_i. Foi utilizado um AG em que a função *fitness* é o número de interações final do algoritmo, e a variável utilizada para o cruzamento é o vetor μ, com um μ_i para cada agente.

Com essa estratégia, obteve-se melhores resultados quanto ao número de interações final do algoritmo (WANG, 2018). Além disso, foi utilizado um cruzamento por média aritmética entre dois valores do vetor de μ, e a mutação escolhida foi alterada aleatoriamente, um valor do vetor μ.

Vale ressaltar que, por ser uma meta heurística, não existe nenhuma comprovação que o AG encontre necessariamente o conjunto de μ_i que corresponde ao *global optima*, porém como esse algoritmo foi rodado mais de uma vez, e em todas as vezes obteve-se valores próximos um do outro, existe um forte indício que o algoritmo esteja chegando em um *global optima*.

Logo após a avaliação dos cenários para o caso matricial, escolhe-se o caso de carga pesada para ser testado no Sistema Multiagente executado no PADE. Esse caso de carga pesada foi escolhido por ser o mais complexo, uma vez que envolve corte de carga

inteligente. Os resultados obtidos via PADE são comparados com os resultados obtidos via matricial para comprovar que a implementação do SMA está correta.

4.1 Caso de Carga Leve

Os parâmetros específicos para o cenário de Carga Leve estão mostrados na Tabela 4.2. Observa-se que a bateria possui um SOC que permite tanto que ela seja carregada como descarregada, e que o algoritmo utilizado para realizar o DEO deve apontar se é o melhor momento para carregá-la ou descarregá-la.

Tabela 4.2 – Parâmetros específicos para o caso de Carga Leve.

Tipo de Agente	Variável	Valor
Carga Não-Prioritária	u	-1,9532
	w	-36,33
	$\lvert p_{max_i} \rvert$	18,6
Bateria	SOC	0,1
	p_{min_i}	-8
	p_{max_i},	32
Geração Não Despachável	P_{ger}	-10
Carga Prioritária	u	-1,482
	w	-44,46
	p_{max_i},	30

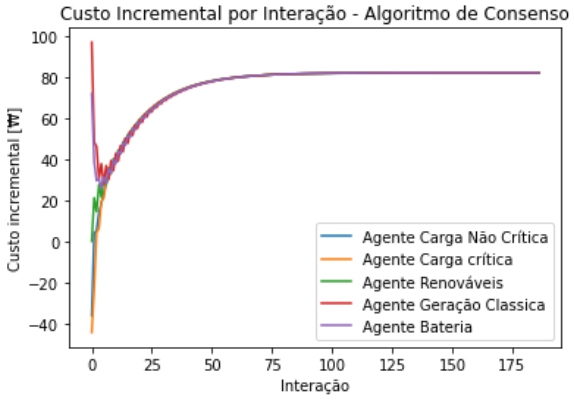
Fonte: Adaptado pelo Autor

4.1.1 Consenso

A Figura 25 mostra a evolução do custo incremental em cada interação do algoritmo de consenso. Essa variável converge na interação de número 187 para o valor de custo incremental de potência de 82,19 ₩. Já a Figura 26 mostra a evolução da potência dos agentes em cada interação. Os valores finais de potência de cada agente são mostrados na Tabela 4.3. Observa-se que para o caso de carga leve o algoritmo opta por carregara bateria, uma vez que o seu valor de potência é maior que 0. Isso condiz com que é esperado do comportamento de uma microrrede, uma vez que, em horários de carga

leve, o custo de carregamento da bateria será menor que em horários de carga mais pesada. Além disso, percebe-se que nesse caso o sistema conseguiu atender toda a demanda. Por fim, o tempo necessário para atingir a convergência foi de 11 ms, o que corresponde aproximadamente a 0,059 ms/interação. No entanto, vale ressaltar que o tempo por interação obtido na prática varia bastante mesmo se utilizando a mesma máquina, pois a máquina provavelmente possuirá condições diferentes quando roda cada um dos algoritmos.

Figura 25:Custo incremental em cada interação ao utilizar o algoritmo de Consenso.



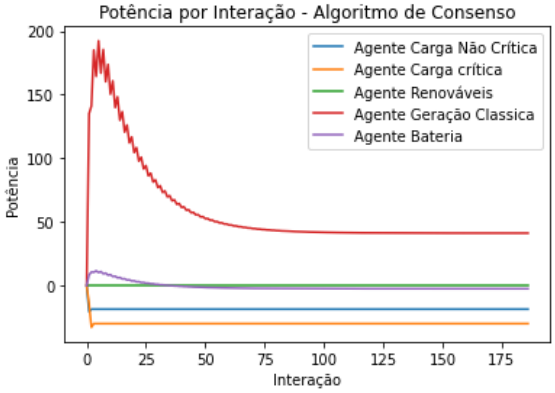
Fonte: Próprio Autor

Tabela 4.3 – Valores finais de potência em cada agente.

Agente	Potência Final [kW]
Carga Não Prioritária	18,6
Carga Prioritária	30
Bateria	2,5463
Geração Despachável	-41,1511
Geração Não Despachável	-10

Fonte: Próprio Autor

Figura 26:Potência dos agentes em cada interação com o algoritmo de Consenso.

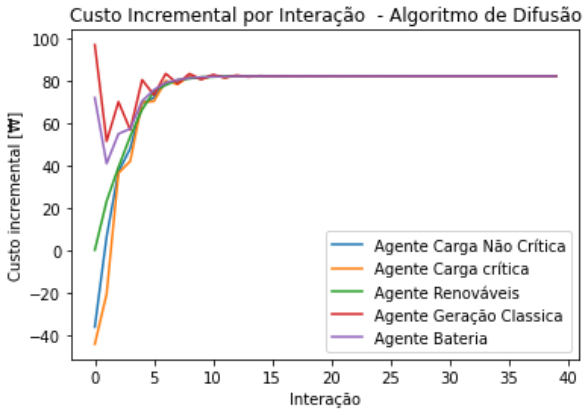


Fonte: Próprio Autor

4.1.2 Difusão

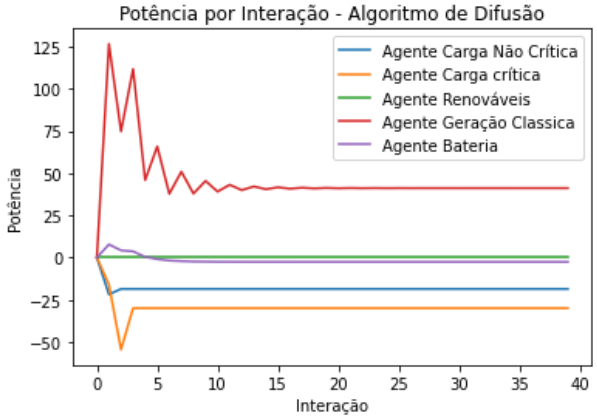
A Figura 27 mostra a evolução do custo incremental em cada interação da Difusão Exata. Para fins de comparação, o eixo da abcissa da figura foi modificado paraser o mesmo do Consenso. Utilizando o algoritmo de Difusão, a convergência se dá na interação de número 40 para o valor de custo incremental de potência de 82,19 ₩. Os valores finais de potência encontrados pela Difusão são os mesmos do Consenso. O gráfico mostrando como se comportam os valores de potência para cada agente na Difusão é mostrado na Figura 28. O tempo para o algoritmo de Difusão atingir a convergência foi de 1,9 ms, o que corresponde a 0,0475 ms por interação. Pode-se perceber que o número de interações que leva a convergência na Difusão é consideravelmente menor que no Consenso e que o tempo por interação da Difusão é menor que o do Consenso. A mesma observação feita no algoritmo de Consenso quanto ao tempo por interação ser dependente da máquina onde é executado o algoritmo é válido para a Difusão.

Figura 27:Custo incremental em cada interação ao utilizar o algoritmo de Difusão.



Fonte: Próprio Autor

Figura 28:Potência em cada interação ao utilizar o algoritmo de Difusão Exata.



Fonte: Próprio Autor

4.1.3) Difusão Exata

Por fim, implementa-se o algoritmo da Difusão Exata, o único que irá utilizar a matriz de pesos *Averaging rule*; os outros dois algoritmos utilizaram a matriz *Mean Metropolis*.

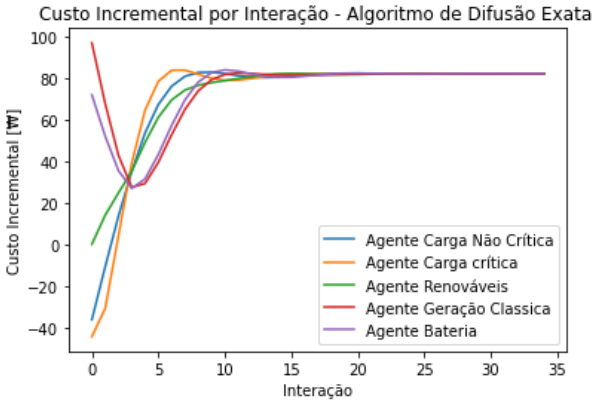
A convergência do custo incremental é mostrada na Figura 29. Assim como no algoritmo de Difusão, o eixo da abcissa do gráfico da Difusão Exata foi ajustado para o

mesmo número de iterações do Consenso para permitir melhor visualização comparativa de performance entre os métodos.

A Difusão Exata convergiu para o valor ótimo de custo incremental de potência de ₩ 82,19 em 35 interações. Foram necessários 2 ms para rodar as 35 interações, ou seja 0,057 ms por interação, porém, como já dito anteriormente, o tempo para atingir a convergência pode variar segundo as condições da máquina.

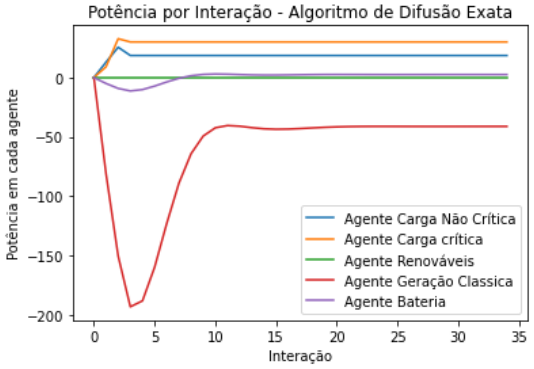
A Figura 30 mostra o comportamento por interação dos valores de potência para cada agente e os valores finais de potência obtidos na Difusão Exata são os mesmos mostrados na Tabela 4.3.

Figura 29:Custo incremental por interação ao utilizar o algoritmo de Difusão Exata.



Fonte: Próprio Autor

Figura 30:Potência por interação ao utilizar o algoritmo de Difusão Exata.



Fonte: Próprio Autor

4.2 Caso de Carga Média

Os parâmetros específicos para o cenário de Carga Média estão mostrados na Tabela 4.4. Pode-se observar que para esse caso a bateria possui um SOC de 0,4, o que permite que ela seja tanto carregada como descarregada. O algoritmo utilizado para realizar o DEO deverá escolher se o momento é de carga ou de descarga da bateria.

Tabela 4.4 – Parâmetros específicos para o caso de Carga Média.

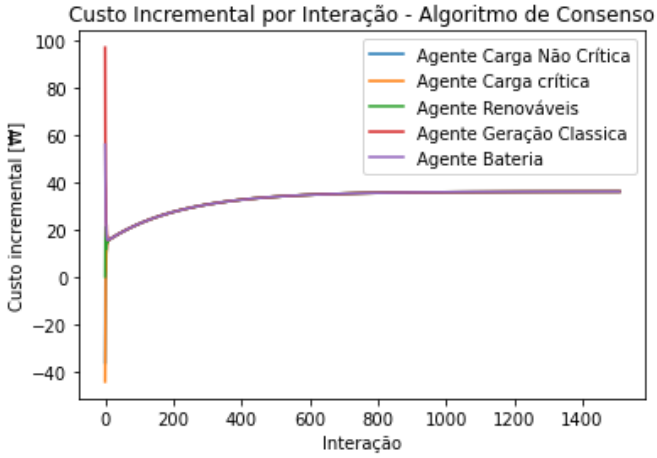
Tipo de Agente	Variável	Valor
Carga Não Prioritária	u	-1,1592
	w	-36,33
	$\|p_{max_i}\|$	31,34
Bateria	SOC	0,4
	p_{min_i}	-16
	p_{max_i},	24
Geração Não Despachável	P_{ger}	-8,5
Carga Prioritária	U	-1,3829
	w	-44,46
	p_{max_i},	32,15

Fonte: Adaptado pelo Autor

4.2.1 Consenso

A Figura 31 mostra como se comporta o custo incremental do Consenso em cada interação, para o caso de carga no nível médio. O custo incremental converge em 1509 interações para o valor de 36,04 ₩. A Figura 32 mostra como os valores de potência se comportam em cada interação enquanto a Tabela 4.5 mostra os valores finais obtidos pelo algoritmo de Consenso. Observa-se que o algoritmo optou pordescarregar a bateria, uma vez que o gerador despachável estava trabalhando no seu limite, e caso a bateria não fosse utilizada no modo de descarregamento, o sistema teria que sofrerum corte de carga. Por fim, percebe-se que o tempo necessário para atingir a convergência é de 188,97 ms, o que corresponde a aproximadamente 0,125 ms por interação.

Figura 31:Comportamento do Custo Incremental por interação.



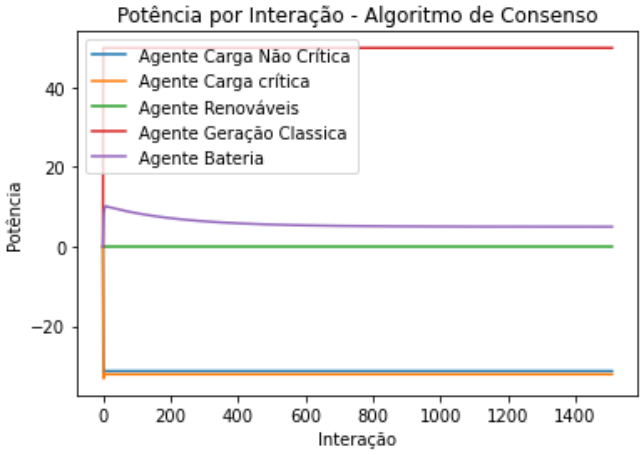
Fonte: Próprio Autor

Tabela 4.5 – Valores finais de potência em cada agente.

Agente	Potência Final [kW]
Carga Não Prioritária	31,34
Carga Prioritária	32,15
Bateria	-4,9954
Geração Despachável	-50
Geração Não Despachável	-8,5

Fonte: Próprio Autor

Figura 32:Potência dos agentes em cada interação.

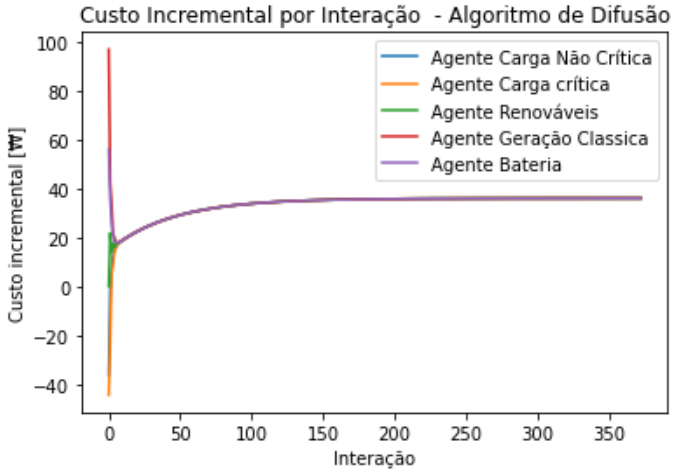


Fonte: Próprio Autor

4.2.2 Difusão

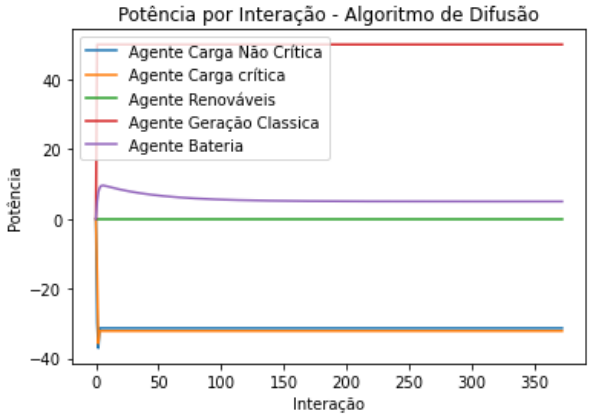
A Figura 33 mostra o comportamento do custo incremental por interação ao se implementar o algoritmo de Difusão. Observando que o eixo da abcissa da figura foi ajustado para permitir comparação nas mesmas bases. O algoritmo de Difusão consegue ser mais eficiente em termos de esforço computacional que o algoritmo de Consenso. Por fim, para rodar o algoritmo foram necessários 29 ms, o que resulta em 0,077 ms por interação. O custo incremental convergiu para o valor de 36,04 ₩, enquanto os valores finais de potência foram os mesmos encontrados na Tabela 7.5.

Figura 33:Comportamento do custo incremental por interação.



Fonte: Próprio Autor

Figura 34: Potência dos agentes em cada interação.



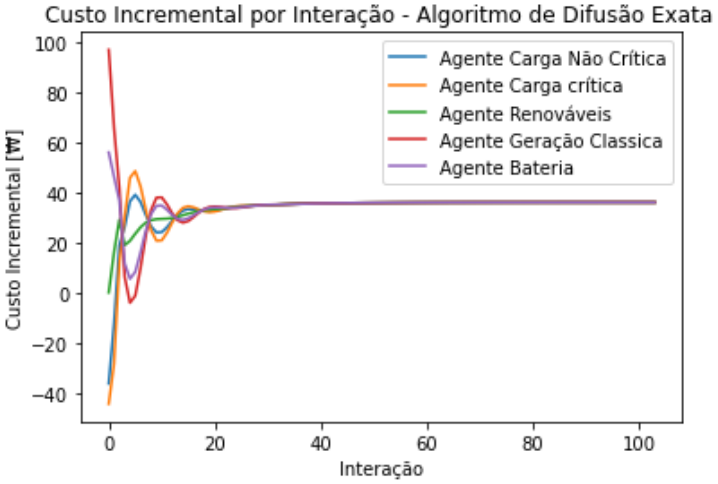
Fonte: Próprio Autor

4.2.3 Difusão Exata

A Figura 35 mostra o comportamento do custo incremental com o algoritmo de Difusão Exata. O algoritmo convergiu em 104 interações, que demoraram 6 milissegundos, o que resulta em 0,0577 ms/interação. O algoritmo convergiu para custo incremental de potência de 36,04 ₩.

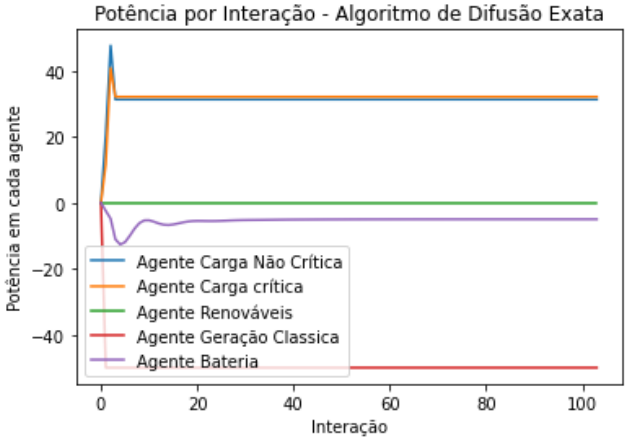
O comportamento da potência por interação é mostrado na Figura 36, novamente, os valores finais de cada agente são mostrados na Tabela 4.5. Percebe-se mais uma vez que o algoritmo de Difusão Exata apresenta performance superior ao Consenso e a Difusão, além é claro, da vantagem desse algoritmo respeitar melhor a privacidade entre os agentes.

Figura 35:Custo Incremental em cada interação.



Fonte: Próprio Autor

Figura 36:Potência por interação.



Fonte: Próprio Autor

4.3 Caso de Carga Pesada

Os parâmetros gerais para o cenário com Carga Pesada são mostrados na Tabela 4.6. Nesse caso, como o objetivo é testar o corte de carga, coloca-se o SOC da bateria bastante baixo de 0,1, limitando sua potência de descarregamento a somente -4 kW. Espera-se que o algoritmo primeiramente corte toda sua carga não prioritária antes de cortar sua carga prioritária, uma vez que a prioritária possui um benefício marginal maior que a Não Prioritária.

Tabela 4.6 – Parâmetros específicos para o caso de Carga Pesada.

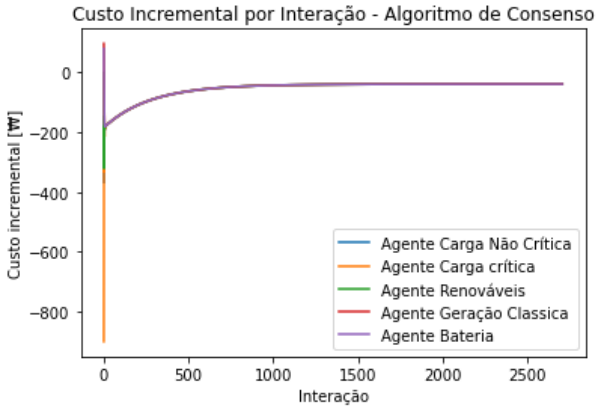
Tipo de Agente	Variável	Valor
Carga Não Prioritária	u	-6,6750
	w	-200,25
	$\lvert p_{max_i} \rvert$	30
Bateria	SOC	0,1
	p_{min_i}	-4
	p_{max_i},	36
Geração Não Despachável	P_{ger}	-13,15
Carga Prioritária	u	-20,05
	w	-900,44
	p_{max_i},	44,9

Fonte: Adaptado pelo Autor

4.3.1 Consenso

A Figura 37 mostra como se comportou o custo incremental em cada interação utilizando o consenso para o caso de carga pesada. O custo incremental convergiu para o valor final de -38,42 ₩. O valor final do custo incremental ser negativo é um sinal que o algoritmo realizou um corte de carga, esse corte irá maximizar o benefício social das cargas. O algoritmo converge em 2709 interações, e para isso ele leva 239,99 milissegundos, o que resulta em 0,089 segundos/interação.

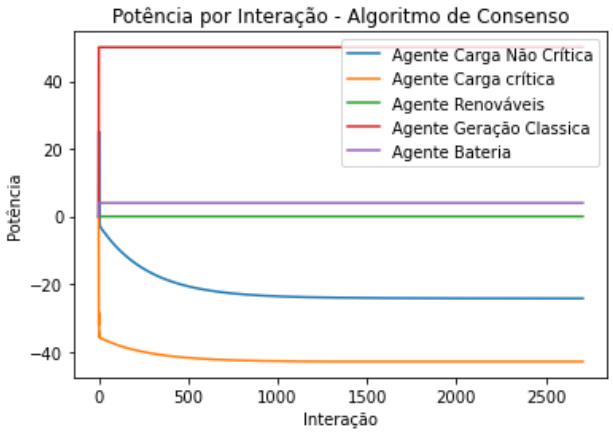
Figura 37:Custo incremental por interação.



Fonte: Próprio Autor

A Figura 38 expressa o valor das potências em cada interação do código e a Tabela 4.7 mostra os valores finais da otimização. Percebe-se que o algoritmo corta aproximadamente 5,75 kW de carga não prioritária e que não há corte da carga prioritária. Isso está de acordo com o que era esperado, uma vez que a carga prioritária foi modelada para sempre possuir um benefício maior que a não prioritária, e por isso, teria preferência em ser atendida pela microrrede.

Figura 38:Potência por interação.



Fonte: Próprio Autor

Tabela 4.7– Valores finais de potência em cada agente.

Agente	Potência Final [kW]
Carga Não Prioritária	24,2449
Carga Prioritária	42,9
Bateria	-4
Geração Despachável	-50
Geração Não Despachável	-13,15

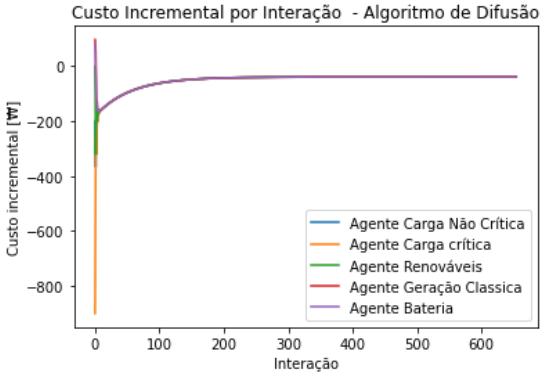
Fonte: Próprio Autor

4.3.2 Difusão

A Figura 39 mostra o comportamento do custo incremental para cada interação. A difusão converge para custo incremental igual a -38,39 ₩ com 655 interações. O tempo para rodar essas 655 interações foi de 46,87 milissegundos, o que resulta em 0,072 segundos/interação.

Vale ressaltar a diferença existente entre o custo incremental final do Consenso e da Difusão, isso se deve devido ao erro que os dois algoritmos apresentam em relação ao melhor resultado, sendo esse erro em função de μ^2, como esperado. No entanto, cabe ressaltar que o erro é pequeno.

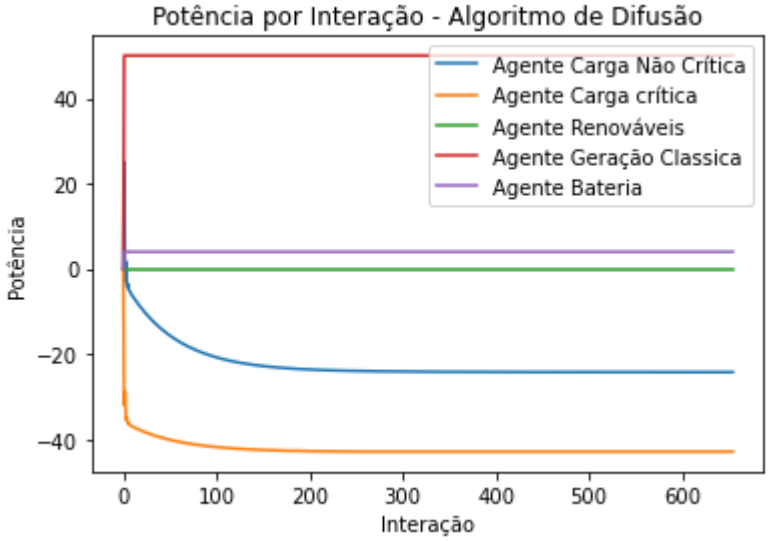
Figura 39:Custo incremental por interação



Fonte: Próprio Autor

A Figura 40 mostra como se comporta a potência em cada interação do algoritmo, os resultados obtidos pela Difusão são praticamente os mesmos do Consenso,que estão mostrados na Tabela 4.16, salientando a diferença de que na Difusão a carga não prioritária tem um valor final de 24,2489 kW.

Figura 40: Potência por interação

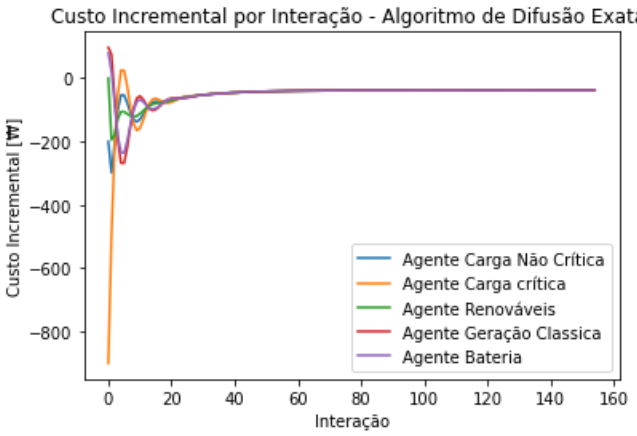


Fonte: Próprio Autor

4.3.3 Difusão Exata

A Figura 41 mostra como o custo incremental se comporta ao logo de todas as interações. A Difusão Exata converge na interação de número 155 para o valor de custo incremental de -38,38 ₩, um valor ainda melhor que o obtido pelo Consenso e a Difusão. Para alcançar esse valor o algoritmo necessitou de 29,18 milissegundos, o que equivale a 0,188 milissegundos por interação.

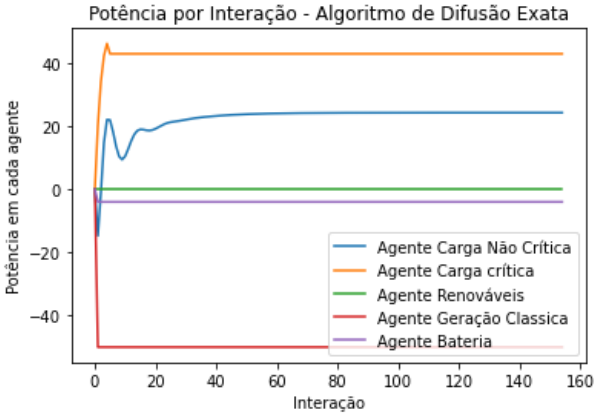
Figura 41: Custo Incremental por Interação



Fonte: Próprio Autor

Já a Figura 42 mostra o comportamento dos valores de potência em cada agente. Os valores finais ao se utilizar a Difusão Exata são praticamente os mesmos que os mostrados na Tabela 4.7, a única diferença é que no agente carga Não Prioritária teve seu valor final igual a 24,2498 kW. Ao se fazer as somas das potências percebe-se que o algoritmo de Difusão Exata é o que chega mais próximo de 0, ou seja, o que melhor atende o despacho econômico ótimo.

Figura 42: Potência por interação



Fonte: Próprio Autor

4.4 Comparação do Desempenho dos Algoritmos para DEO

A Tabela 4.8 resume as principais características apontadas nos três algoritmos nas sessões 4.1, 4.2 e 4.3. Observa-se que o algoritmo de Difusão Exata, apesar de ter uma maior complexidade de implementação, consegue ter um maior desempenho tanto no número de interações como no grau de privacidade de informações que o utilizará para realizar o DEO. Além disso, apesar do algoritmo de Difusão Exata ter um custo computacional por interação levemente maior que os outros algoritmos, ele irá se sobressair aos outros devido necessitar de um número menor de interações para convergir, e também a precisão com que ele alcança esse resultado, com um erro 0, em comparação ao resultado ótimo real.

Tabela 4.8 – Comparativo entre Consenso, Difusão e Difusão Exata.

Característica	Consenso	Difusão	Difusão Exata
Complexidade na implementação	Baixa	Média	Alta
Estabilidade	Baixa	Média	Alta
N° de Interações para Convergir	Alto	Médio	Baixo
Custo Computacional por Interação	Baixo	Quase o mesmo, porém maior	Quase o mesmo, porém maior que a Difusão
Privacidade	Baixa	Média	Alta
Erro no Resultado Final	$f(\mu_i^2)$	$f(\mu_i^2)$	0

Fonte: Próprio Autor

4.5 Caso SMA

Por fim, foi desenvolvido e implementado no PADE um sistema multiagente para DEO de uma microrrede. A plataforma PADE foi integrada à plataforma mosaik para sincronizar a comunicação entre os agentes. Para otimizar o despacho foi utilizadoo algoritmo de Consenso. Em cada agente da microrrede foi embarcado o algoritmo de

Consenso com sua função custo e restrições associadas para que a partir da troca de informações fosse otimizado o despacho dos recursos energéticos da microrrede.

A seguir será descrito os procedimentos para implementação do SMA com Consenso e a rede de comunicação.

Para executar o SMA é necessário abrir duas abas do terminal de comando do computador, uma para rodar o PADE, outra para rodar o mosaik. Além disso, deve-se esperar um tempo entre a execução do PADE e a do mosaik para que o PADE consiga atualizar suas listas. A Tabela 4.9 mostra os parâmetros inseridos no PADE, a Figura 43 mostra a tela inicial do PADE e a Figura 44 mostra os resultados obtidos no final do processo.

Pela Tabela 4.9 percebe-se que as variáveis que serviram de entrada para o PADE foram:

- ID: Identificação do Tipo de Agente. ID = 0 significa um agente do tipo Carga, ID=1 significa agente do tipo Geração Não-Despachável, ID=2 identifica oagente de Geração Despachável e ID=3 é o agente Bateria.
- Alpha e Beta: Parâmetros da função custo, tanto do agente Geração Despachável quanto do agente Bateria. Como esses parâmetros não são utilizados pelos outros tipos de agentes, foi-se definido como 0 nos demais tipos.
- Epil: Variável de *feedback* de cada um dos agentes, utilizada para chegar na convergência com o menor número de interações possível.
- P_{ren}: Quantidade de potência renovável dividida pelo número de agentes. Dessa maneira cada agente irá levar em conta esse tipo de geração ao ser rodado o algoritmo de Consenso.
- lim: Limite dos agentes do tipo Bateria e Geração Despachável. Os agentes do tipo Carga também possuem limites, porém foi dado um valor fictício muito alto como parâmetro de entrada, pois, os valores limites reais são tratados de maneira intrínseca ao código.
- We U: Parâmetros da função Benefício utilizados pelo agente Carga. Como esses parâmetros não são utilizados pelos outros tipos de agentes, foi-se definido como 0 nos demais tipos.
- GPS: Posição Geográfica de cada um dos agentes. Essa variável será utilizada pelo agente *Plug-and-Play* para montar a rede de comunicação.

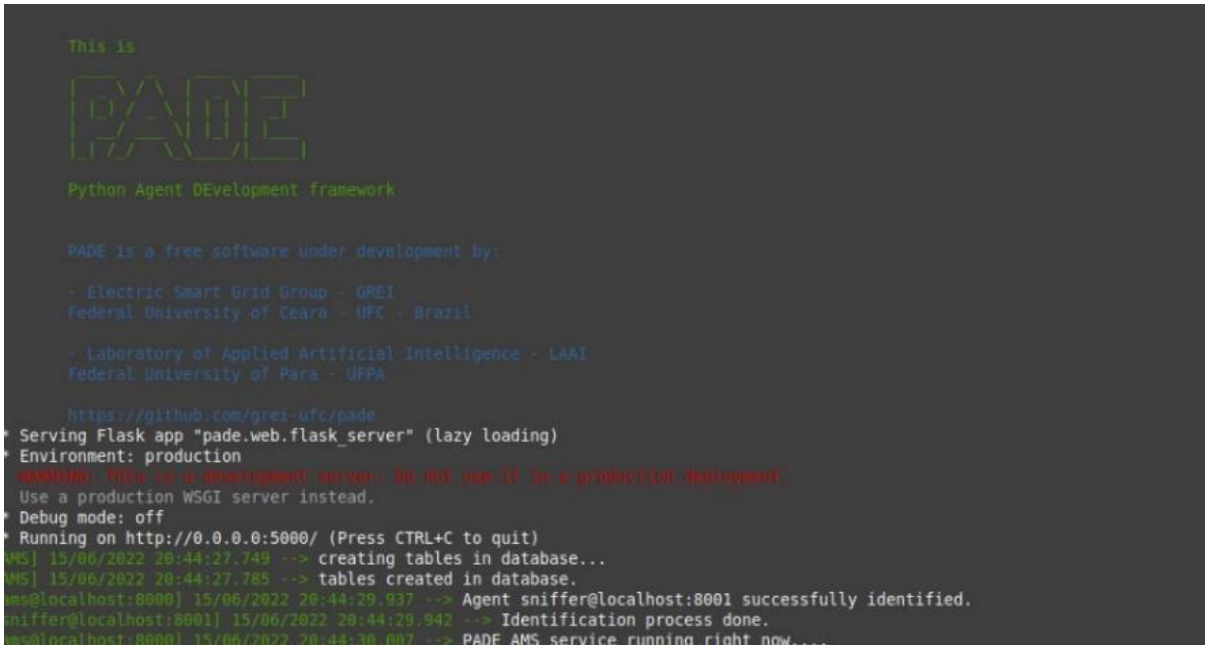
- *Hour_in*: Horário de entrada dos agentes na rede; todos os agentes entram no instante 01.
- *Hour_Out*: Horário de saída dos agentes da rede. Como não é comum a entrada e saída de agentes de uma microrrede, esse horário foi atribuído como sendo um tempo muito maior do que o tempo necessário para o SMA atingir o DEO.

Tabela 4.9 – Parâmetros de entrada do SMA.

Agente	ID	alpha	beta	epil	P_{ren}	lim	w	u	GPS	hour_in	hour_out
Carga NC	0	0	0	0.0915	2.63	10000	-200.250	-6.675	aa	1	1E+10
Carga C	0	0	0	0.0979	2.63	10000	-900.440	-20.054	ab	1	1E+10
Renováveis	1	0	0	0.0821	2.63	0	0	0	ac	1	1E+10
Despachavel	2	0.18	97	0.0834	2.63	-50	0	0	ad	1	1E+10
Bateria	3	2	80	0.0984	2.63	-4	0	0	ae	1	1E+10

Fonte: Próprio Autor

Figura 43: Tela inicial do PADE

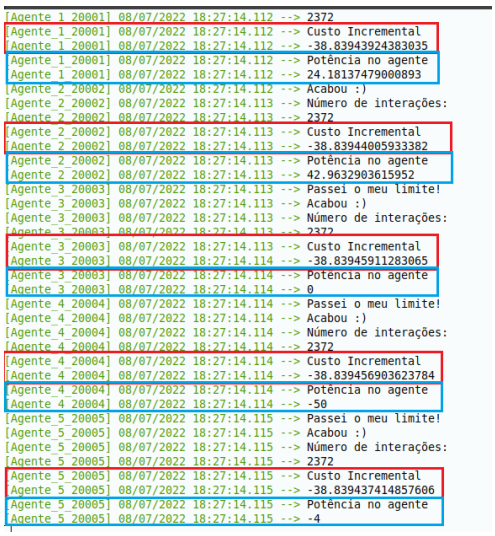


Fonte: Próprio Autor

Na Figura 44 estão descritos os resultados obtidos pelo SMA, o custoincremental de cada agente está destacado pelas caixas vermelhas enquanto a potência

dos agentes está destacada pelas caixas azuis. Ao compará-la com os resultados obtidos na seção 4.3.1, descritos na Tabela 4.7, percebe-se que todas as variáveis possuem o mesmo valor final, comprovando-se assim, os resultados obtidos pelo SMA.

Figura 44: Resultados do PADE.



Fonte: Próprio Autor.

4.6 Caso SMA + Plug and Play

Para o caso do Sistema Multiagente sincronizado via Mosaik , também foi desenvolvido o agente *Plug and Play* . Esse agente irá receber as informações sobre a geoposição dos outros agentes e a partir dessa informação irá montar a rede de comunicação onde os demais agentes irão se comunicar para realizarem o DEO.

O agente *Plug and Play* também será responsável por recompor a rede de comunicação entre os agentes em casos de falha de um agente, onde ele irá excluir o componente defeituoso da rede e remontá-la somente com os componentes não defeituoso. E nos casos onde é desejável incluir um agente extra na rede, onde o *plug and play* irá remontar a rede de comunicação, inserindo o novo participante nessa rede.

O agente *Plug and Play* irá verificar, a cada 10 interações do Mosaik, se algum agente solicitou entrada ou saída do SMA, caso algum agente tenha feito qualquer uma dessas solicitações, ele irá interromper o consenso, reconfigurar a rede e depois reiniciar o algoritmo.

No caso de estudo, tem-se uma microrrede com os mesmos parâmetros que no caso de Carga Pesada, mostrados na Tabela 4.6, sendo assim o SMA irá realizar um corte de carga, caso não haja mudanças na configuração da rede, porém no instante 45 do Mosaik , um novo agente geração irá solicitar para o *plug and play* para entrar na microrrede. Os parâmetros desse novo agente geração são mostrados na Tabela 4.10.

Tabela 4.10: Parâmetros do Novo Agente Plug and Play

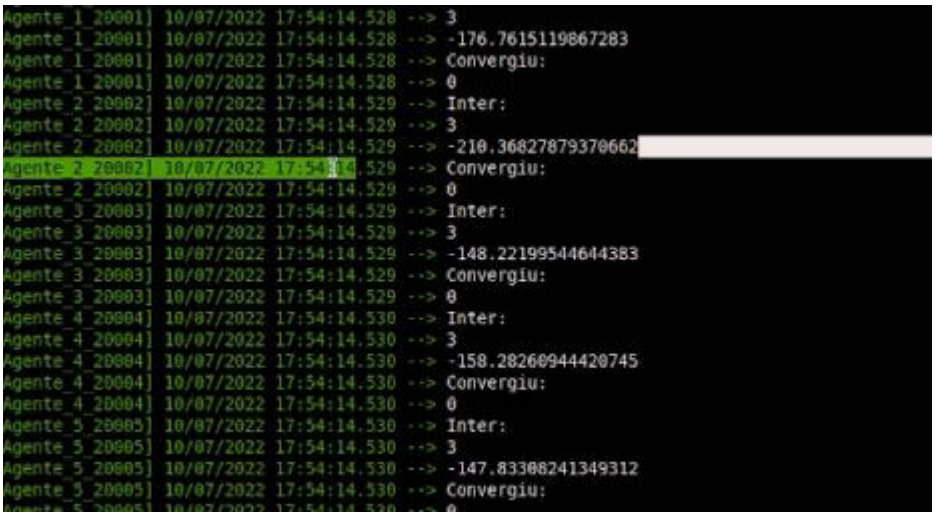
ID	2
Alpha	0.20
Beta	97
Epil	0.0279293
Lim	-30
W	0
U	0

GPS	af
Hour_in	45
Hour_out	Muito Alto

Fonte: Próprio Autor

A Figura 45 mostra como o SMA estava se comportando antes da entrada do novo gerador , por ela percebe-se que o custo incremental do sistema estava convergindo para um valor negativo.

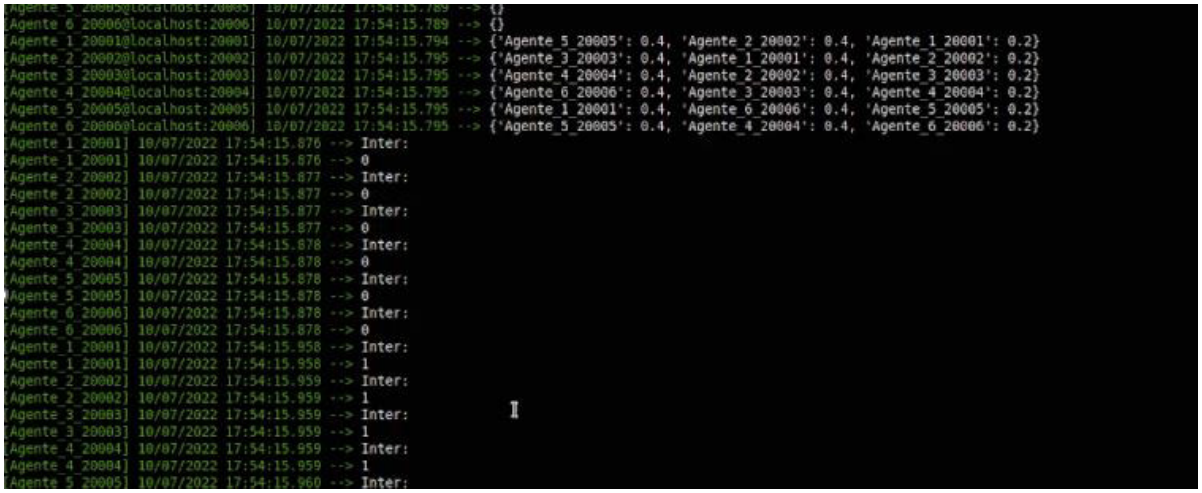
Figura 45: Comportamento do SMA antes do Novo Agente



Fonte: Próprio Autor

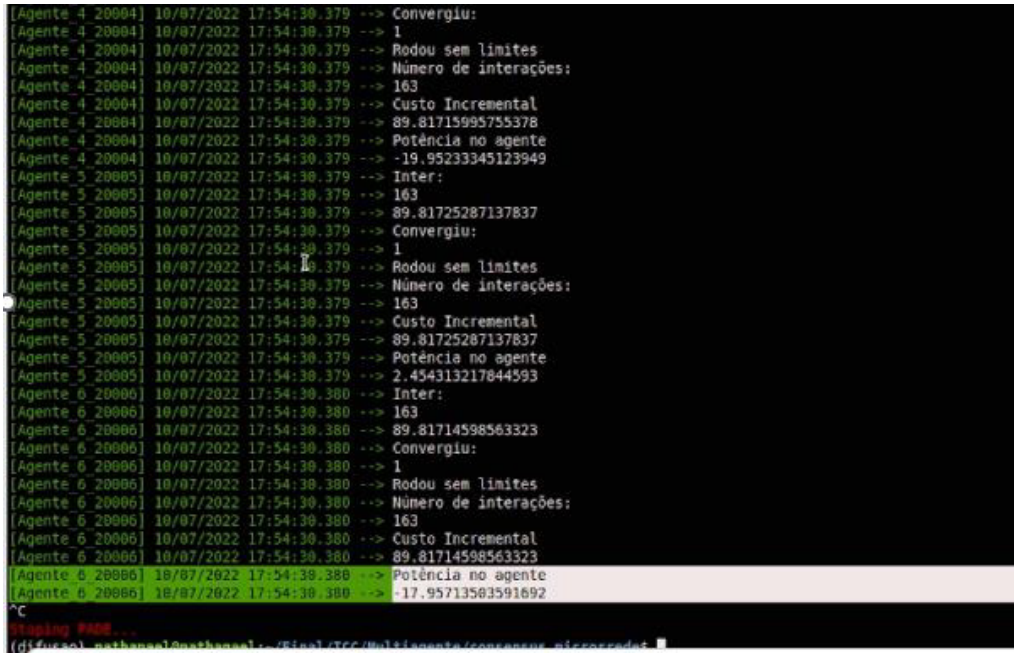
Já a Figura 46 mostra o novo agente entrando em cena, e a rede se reconfigurando, enquanto que a Figura 47 mostra o resultado final do DEO, já com os 6 Agentes participando do SMA. Os resultados mostrados na Figura 47 mostram que o SMA convergiu para um valor positivo de custo incremental, ou seja, não houve corte de carga, devido a inserção desse novo gerador no SMA.

Figura 46: Rede se recompondo após a entrada do Novo Agente



Fonte: Próprio Autor

Figura 47: Valores Finais do SMA com o Plug and Play



Fonte: Próprio Autor

5 Conclusão

Neste Trabalho foi apresentado o desenvolvimento, teste e validação de um sistema de despacho econômico ótimo para microrredes objetivos propostos para mircrorredes com algoritmos de otimização em uma arquitetura de gerenciamento distribuído, tendo sido implementados três algoritmos distribuídos, Consenso, Difusão, Difusão Exata e um deles foi levado ao Sistemas Multiagente via PADE para obter o despacho econômico ótimo de uma microrrede.

Para melhorar a performance dos algoritmos distribuídos com rede de comunicação modelada por matriz, foi aplicada a metaheurística algoritmo genético para obter o melhor conjunto de variáveis de *feedback* de cada agente, a fim de melhoraro número de interações para convergência dos métodos. Além disso, o estudo das matrizes de pesos contribuiu para trazer maior eficiência quanto ao número de interações.

O Sistema Multiagente foi desenvolvido e implementado na plataforma PADE integrada ao mosaik num ambiente de co-simulação. Esse sistema construído pode ser utilizado em uma implementação prática, diferentemente do modelo matricial normalmente encontrado nos demais trabalhos que tratam sobre o tema. Para obter o despacho econômico ótimo foi implementado no SMA o algoritmo de Consenso.

Dentre os algoritmos distribuídos matriciais o que mostrou melhor performance quanto ao erro e privacidade de informação trocada foi o algoritmo de Difusão Exata. Toda potência disponível na microrrede foi usada e a informação trocada entre é o parâmetro de difusão. O DEO usando SMA com Consenso apresentou resultado semelhante ao do Consenso matricial.

5.1 Trabalhos Futuros

Como trabalho futuro propõe-se o uso dos métodos de Difusão e Difusão Exata no SMA a fim de avaliar o desempenho dos modelos. Ainda, propõe-se a consideração de veículos elétricos cuja característica técnica se assemelha a de uma bateria, no entanto apresentando maior aleatoriedade no carregamento e na conexão à microrrede.Como uma rede que possui veículos elétricos é bem mais dinâmica que a microrrede atual, outro trabalho futuro proposto é tentar encontrar alguma relação entre os agentes utilizados e o

valor ótimo da variável de feedback nos algoritmos.

Quanto ao sistema multiagente, algumas melhoras podem consideradas como: a reconfiguração do Agente Plug and Play, para tornar o sistema totalmente independente de um agente central e a elaboração de um novo critério de corte de carga, uma vez que esse critério utilizado aumenta consideravelmente o número de interação necessárias para a convegência.

REFERÊNCIAS

ABIDO, M.A.**; Optimal Design of Power–System Stabilizers Using Particle Swarm Optimization**. IEEE Transactions on Energy Conversion.Vol.17, September 2002, Pages 406-413.

ALI, H. ; HUSSAIN, A. ; BUI, V. ; KIM, H. **Consensus Algorithm-Based Distributed Operation of Microgrids during Grid-Connected and Islanded Modes**. IEEE Acess. Vol.8, April 2020, Pages 78151-78165.

AGUIAR, A.S. **Equivalente Certo e Medidas de Risco em Decisões de Comercialização de Energia** . 2008. Tese (Doutorado em Engenharia Elétrica) – Departamento de Engenharia Elétrica, Pontíficia Universidade Católica, Rio de Janeiro, 2008.

BAKIRTZIS, A.; PETRIDIS, B.; KAZARLIS, S. **Genetic Algorithm Solution to the Economic Dispatch Problem**. IEE Proc.-Gener. Transm. Distrib., Vol. 141, No. 4, July 1994.

BARATI, H.; SADEGHI, M. **An Efficient Hybrid MPSO-GA Algorithm for Solving Non-Smooth/Non-Convex Economic Dispatch Problem with pratical constraints**. Ain Shams Engineering Journal, Vol. 9, December 2018. Pages 1279-1287.

CANAL SOLAR. **O que é geração distribuída de energia elétrica?** Canal Solar. 2021. Disponível em: https://canalsolar.com.br/o-que-e-geracao-distribuida-de-energia-eletrica/. Acesso em: 12 maio. 2022.

DE AZEVEDO, R. ; CINTUGLU, M.H.; MA, T. ; MOHAMMED, O.A. **Multiagent-Based Optimal Microgrid Control Using Fully Distributed Diffusion Strategy.** IEEE Transactions on Smart Grid, Vol.8 , July 2017, Pages 1997-2008.

DONÁRIO, A.A.; SANTOS, R.B. **Teoria do Consumidor**.2015. Disponível em: https://repositorio.ual.pt/bitstream/11144/3191/3/TEORIA%20DO%20CONSUMIDOR.pdf. Acesso em: 25 maio. 2022.

HE, Y.; WANG, W.; WU, X. **Multi-Agent Based Fully Distributed Economic Dispatch in Microgrid Using Exact Diffusion Strategy**. IEEE Acess. Vol. 8, December 2019, Pages. 7020-7031.

FIPA. **The Foundation for Intelligent Physical Agents standards**. Disponível em: http://www.fipa.org. Acesso em: 10 maio. 2022.

FIPA. **History of FIPA.** Disponível em: http://www.fipa.org/subgroups/ROFS-SG-docs/History-of-FIPA.htm. Acesso: 15 abril.2022.

KIM, T.H.; RAMOS, C.; MOHAMMED, S. **Smart City and IoT. Future Generation Computer Systems**, Volume 76, November 2017, Pages 159-162.

MARQUES, G. **Você Sabe o que é uma microrrede?** Legado Energia. 2020. Disponível em:legadoenergias.com/publicacao/voce-sabe-o-que-e-uma-microrrede. Acesso em: 24 de abr. 2022.

MELO, L. **Python Agent DEvelopment framework**, c2016. Protocolos, Disponível em: pade.readthedocs.io/pt_BR/latest/. Acesso em: 24 abr. 2022.

MOSAIK. **What's Mosaik Supposed to Do**. 2012. Disponível em: mosaik.readthedocs.io. Acesso em: 26 abr. 2022.

NARUTO, D.T. **Vantagens e Desvantagens da Geração Distribuída e Estudo de Caso de um Sistema Sola Fotovoltaico Conectado à Rede Elétrica**. 2017. Monografia (Bacharelado em Engenharia Elétrica) – Departamento de Engenharia Elétrica, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2017.

PARK, J.P.; JEONG, Y.W.; SHIN, J.R.; LEE, K.Y. **An Improved Particle Swarm Optimization for Nonconvex Economic Dispatch Problems**. IEEE Transaction on Power Systems, Vol. 25, Pages 156-166.

PASCUTTI, M.C.D. **Uma Proposta de Arquitetura de um Ambiente de Desenvolvimento de Software Distribuído Baseada em Agentes**. 2002. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2002.

SAYED, A.H. **Multi-Agent Distributed Strategies**. *In*: SAYED, A.H. Adaptation, Learning and Optimization over Networks. Foundations and Trends in Machine Learning, vol. 7, Pages 448 -470.

PINDYCK, R.; RUBINFELD, D. **Comportamento do Consumidor**. *In*: PINDYCK, R. ; RUBINFELD, D. Microeconomia. Pearson Education, vol. 7, Pages 65-101.

SAYED, A.H. **Multi-Agent Distributed Strategies**. *In*: SAYED, A.H. Adaptation, Learning and Optimization over Networks. Foundations and Trends in Machine Learning, vol. 8, Pages 448 -470.

SAYED, A.H. **Stability of Multi-Agent Networks**. *In*: SAYED, A.H. Adaptation, Learning and Optimization over Networks. Foundations and Trends in Machine Learning, vol. 7, Pages 507 -552.

TANENBAUM, ANDREW S.; STEEN, M. V. Distributed Systems: Principles and Paradigms. 2. ed. ed. Amsterdam: Pearson Prentice Hall, 2007.

TAVEIRA, I.M.; LEÃO, R.P.S.L.; CAVALCANTE, T.M. **A Influência da Geração Fotovoltaíca no Fator de Unidades Prossumidoras: Uma Avaliação Técnica e Financeira**. In: Congresso Brasileiro de Energia Solar, 8., 2020. Fortaleza. Anais [...]. Fortaleza: Hotel Grand Marquise Fortaleza, 2020.

WANG, R.; LI, Q.; ZHANG, B. ; WANG, L. **Distributed Consensus Based Algorithm for Economic Dispatch in a Microgrid**. IEEE Transactions on Smart Grid. Vol. 10, July 2019, Pages 3630 – 3640.

WANG, S.K. ; CHIOU, J.P. ; LIU, C.W. **Non-Smooth/Non-Convex Economic Dispatch by a Novel Hybrid Differential Evolution Algorithm**. IET Generation, Transmission and Distribution, Vol. 01, October 2007, Pages 793-803.

YUAN, K. ; YING, B.; ZHAO, X.; SAYED, A.H. **Exact Diffusion for Distributed Optimization and Learning – Part I : Algorithm Development**. IEEE Transactions on Signal Processing.Vol.27, October 2018, Pages 708-723.

ZHANG, Z.; CHOW, M. Y. **Cost Consensus Algorithm Under Different Communication Network Topologies in a Smar Grid**. IEEE Transactions on Power Systems. Vol. 27, No. 4 , November 2012, Pages. 1761-1768.

APÊNDICE A – Códigos Desenvolvidos

Código 01 - Algoritmo de Consenso: Modo Matricial

O código desenvolvido no github se encontra no repositório do Grupo de Redes Elétricas Inteligentes e pode ser acessado através do link:

https://github.com/grei-ufc/TCC-2022-NathanaelDuque/tree/main/Matricial/Consenso

Código 02 - Algoritmo de Difusão: Modo Matricial

O código desenvolvido no github se encontra no repositório do Grupo de Redes Elétricas Inteligentes e pode ser acessado através do link:

https://github.com/grei-ufc/TCC-2022-NathanaelDuque/tree/main/Matricial/Difus%C3%A3o

Código 03 - Algoritmo de Difusão Exata: Modo Matricial

O código desenvolvido no github se encontra no repositório do Grupo de Redes Elétricas Inteligentes e pode ser acessado através do link:

https://github.com/grei-ufc/TCC-2022-NathanaelDuque/tree/main/Matricial/Difus%C3%A3o%20Exata

Código 04 - Algoritmo Genético para otimização do μ

O código desenvolvido no github se encontra no repositório do Grupo de Redes Elétricas Inteligentes e pode ser acessado através do link:

https://github.com/grei-ufc/TCC-2022-NathanaelDuque/tree/main/Matricial/Algoritmo%20Gen%C3%A9tico

Código 05 – Estudos de Velocidade pelas matrizes

O código desenvolvido no github se encontra no repositório do Grupo de Redes Elétricas Inteligentes e pode ser acessado através do link:

https://github.com/grei-ufc/TCC-2022-NathanaelDuque/tree/main/Matricial/Estudos%20de%20Velocidade

Código 06 – Sistema Multiagentes

O código desenvolvido no github se encontra no repositório do Grupo de Redes Elétricas Inteligentes e pode ser acessado através do link:

https://github.com/grei-ufc/TCC-2022-NathanaelDuque/tree/main/PADE