Revista de Informática Teórica e Aplicada - RITA - ISSN 2175-2745
Vol. 26, Num. 2 (2018) 27-44

RESEARCH ARTICLE

Stochastic Models for Optimizing Availability, Cost and Sustainability of Data Center Power Architectures through Genetic Algorithm

Modelos estocásticos para otimizar a disponibilidade, o custo e a sustentabilidade das arquiteturas de energia de data centers através de algoritmo genético

Márcio Austregésilo[1]*, Gustavo Callou[2]

Abstract: In recent years, the growth of information technology has required higher reliability, accessibility, collaboration, availability, and a reduction of costs on data centers due to factors such as social network, cloud computing, and e-commerce. These systems require redundant mechanisms on the data center infrastrucutre to achieve high availability, which may increase the electric energy consumption, impacting in both the sustainability and cost. This work proposes a multi-objective optimization approach, based on Genetic Algorithms, to optimize cost, sustainability and availability of data center power infrastructures. The main goal is to maximize availability and minimize cost and exergy consumed (adopted to estimate the environmental impacts). In order to compute such metrics, this work adopts the energy flow model (EFM), reliability block diagrams (RBD) and stochastic petri nets (SPN). Two case studies are conducted to show the applicability of the proposed strategy: (i) takes into account 5 typical data center architectures that were optimized to conduct the validation process of the proposed strategy; (ii) uses the optimization strategy in two architectures classified by ANSI / TIA-942 (TIER I and II). In both case studies, significant improvements were achieved in the results, which were very close to the optimum one that was obtained by a brute force algorithm that analyzes all the possibilities and returns the optimal solution. It is worth mentioning that the time used to obtain the results using the genetic algorithm approach was significantly lower (6,763,260 times), in comparison with the strategy which combines all the possible combinations to obtain the optimal result.
Keywords: Genetic Algorithm — Energy Flow Model — Sustainability — Stochastic Petri Net — Reliability Block Diagrams — Availability

Resumo: Nos últimos anos, o crescimento da tecnologia da informação exigiu maior confiabilidade, acessibilidade, colaboração, disponibilidade e redução de custos nos *data centers* devido a fatores como redes sociais, computação em nuvem e comércio eletrônico. Esses sistemas exigem mecanismos redundantes na infraestrutura do *data center* para alcançar uma alta disponibilidade, o que pode aumentar o consumo de energia elétrica, impactando tanto na sustentabilidade quanto no custo. Este trabalho propõe uma abordagem de otimização multi-objetivo, baseada em Algoritmos Genéticos, para otimizar custos, sustentabilidade e disponibilidade de infraestruturas de energia em *data centers*. O principal objetivo é maximizar a disponibilidade e minimizar o custo e a exergia consumida (adotada para estimar os impactos ambientais). Para computar tais métricas, este trabalho adota o modelo de fluxo de energia (EFM), diagramas de blocos de confiabilidade (RBD) e redes de petri estocásticas (SPN). Dois estudos de caso são conduzidos: (i) leva em consideração 5 arquiteturas típicas de *data centers* para mostrar a aplicabilidade e validação da estratégia proposta; (ii) utiliza a estratégia de otimização em duas arquiteturas classificadas pela norma ANSI/TIA-942 (TIER I e II). Em ambos estudos de caso, observou-se uma melhora significativa nos resultados que ficaram bem próximos ao ótimo que foi obtido por um algoritmo de força bruta que analisa todas as possibilidades e retorna a solução ótima. Vale ressaltar que o tempo utilizado para se obter as respostas utilizando a abordagem com algoritmo genético foi significativamente inferior (6.763.260 vezes) se comparado a uma estratégia que combina todas as possibilidades para obter o resultado ótimo.
Palavras-Chave: Algoritmo Genético — Modelo de Fluxo de Energia — Sustentabilidade — Rede de Petri Estocástica — Diagramas de Bloco de Confiabilidade — Disponibilidade

[1] *Department of Statistics and Informatics, Federal Rural University of Pernambuco, Brazil*
[2] *Department of Computing, Federal Rural University of Pernambuco, Brazil*
***Corresponding author**: marcio.austregesilo@ufrpe.br
DOI: http://dx.doi.org/10.22456/2175-2745.83498 • **Received:** 02/06/2018 • **Accepted:** 16/02/2019

1. Introdução

A evolução da tecnologia da informação (TI) vem transformando a sociedade atual através de, por exemplo, redes sociais, comércio eletrônico e *Internet-banking*. Essa evolução impacta desde o relacionamento pessoal até a interação profissional colaborativa entre empresas e pessoas. Devido a esses fatos, a demanda por sistemas com alta confiabilidade, acessibilidade, colaboração, disponibilidade e baixos custos, dentre outros, culminam no uso de um recente paradigma: a computação em nuvem [1].

O crescimento no número de processos pelo uso da computação em nuvem vêm demandando maiores investimentos e atenção por parte dos gestores e analistas na infraestrutura dos *data centers* que provêm suporte a tais sistemas. Com o aumento dessa infraestrutura, para prover suporte à tolerância a falhas, por exemplo, são adicionados equipamentos, o que acaba por aumentar o custo e o consumo de energia [2].

Outro fator requisitado nesses sistemas computacionais é o de alta disponibilidade, sendo assim a infraestrutura elétrica do *data center* deve ter altas taxas de disponibilidade. A redundância é um princípio de projeto amplamente adotado em tolerância a falhas para aumentar a disponibilidade de sistemas [3].

Este trabalho propõe a otimização de arquiteturas elétricas de *data centers*, visando maximizar a disponibilidade e minimizar o impacto ambiental e o custo. O impacto ambiental é parametrizado através da exergia destruída, que é a energia dissipada pelos componentes da arquitetura. Sendo assim, esse trabalho faz uso de uma técnica de otimização multi-objetiva de maneira a equilibrar essas métricas levando em consideração a relação entre elas. Um algoritmo genético multi-objetivo é proposto para realizar a otimização das arquiteturas elétricas de *data center*. Esse algoritmo desenvolvido se comunica com a ferramenta Mercury [4] para fazer a otimização dos modelos propostos. Ao ajustar os parâmetros dos modelos de disponibilidade nos Diagramas de Bloco de Confiabilidade (RBD) e redes de Petri Estocásticas (SPN), bem como nos Modelos de Fluxo de Energia (EFM) [1].

Este trabalho está organizado como segue. A Seção 2 apresenta conceitos teóricos para a fundamentação dos conteúdos deste trabalho. A Seção 3 apresenta os trabalhos relacionados. A Seção 4 explana a metodologia aplicada na utilização do Algoritmo Genético para realizar a otimização das arquiteturas de *data centers*. A Seção 5 ilustra dois estudos de caso, para validar a metodologia, e apresenta a análise dos resultados obtidos. Por fim, a Seção 6 conclui o artigo e descreve direcionamentos para trabalhos futuros.

2. Fundamentação

Nesta Seção serão explanados os conceitos e conhecimentos necessários para um melhor entendimento do trabalho proposto.

2.1 *Data Centers*

Os *data centers* são compostos por três tipos básicos de infraestrutura: infraestrutura de TI, infraestrutura de refrigeração e infraestrutura elétrica.

A infraestrutura elétrica, foco desse trabalho, é responsável pelo fornecimento ininterrupto de energia elétrica condicionada à tensão e a frequência corretas para ambas as infraestruturas de TI e de refrigeração. A energia elétrica vem das concessionárias de energia e, normalmente, passa pelos transformadores, chaves de transferência, fontes de alimentação ininterrupta (UPS - *Uninterruptible Power Supplies*), unidades de distribuição de energia (PDUs - *Power Distribution Units*) e, por fim, pelas réguas de alimentação do *rack* [2]. Devido à necessidade do aumento da tolerância a falhas, se faz necessário a redundância ou replicação dos componentes a fim de aumentar a disponibilidade, incrementando, assim, o consumo de energia. Geradores ou outras fontes de energia locais podem ser utilizados para casos de interrupções mais longas ou para proporcionar uma fração importante da demanda de potência total numa base regular [1].

A principal referência de classificação de *data centers* é a norma ANSI/TIA-942 [5], proposta pela *Telecommunications Industry Association*, que classifica os *data centers* quanto a sua redundância e disponibilidade. Essa norma estabelece algumas definições de redundância para auxiliar a classificação, são elas:

- **N**: o sistema atende aos requisitos básicos e não possui redundância;
- **N+1**: A redundância fornece uma unidade, módulo, caminho ou sistema adicional, além do mínimo necessário para satisfazer o requisito de base. A falha ou manutenção de qualquer unidade, módulo ou caminho não irá interromper as operações. (ex.: energia elétrica da concessionária + *Nobreak*);
- **N+2**: A redundância fornece duas unidades, módulos, caminhos ou sistemas adicionais, além do mínimo necessário para satisfazer o requisito de base. A falha ou a manutenção de duas únicas unidades, módulos ou caminhos não interromperão as operações. (ex.: energia elétrica da concessionária + *Nobreak* + Gerador);
- **2N**: fornece duas unidades completas, módulos, caminhos ou sistemas para todos os componentes requeridos para um sistema base. Falha ou manutenção de uma unidade inteira, módulo, caminho ou sistema não irá interromper as operações (ex.: energia elétrica concessionária A + energia elétrica da concessionária B);
- **2(N+1)**: fornece duas unidades completas (N+1), módulos, caminhos ou sistemas. Mesmo em caso de falha ou manutenção de uma unidade, módulo, caminho ou sistema, as operações não são interrompidas. (ex.: energia elétrica da concessionária A + energia elétrica da concessionária B; *Nobreak* A+ *Nobreak* B; Gerador A + Gerador B).

Por fim a norma ANSI/TIA-942 [5] estabelece quatro níveis de classificação dos *data centers* de acordo com a sua infraestrutura e confiabilidade [6]:

- **TIER I**: está sujeito às interrupções em suas atividades, tanto as planejadas quanto as não. Possui redundância N. Este *data center* pode possuir quadro de distribuição de energia e refrigeração, *nobreak*, gerador (motor). Se o *data center* desse nível possuir um *nobreak* ou gerador, estes serão normalmente sistemas de módulo único, o que ocasionará pontos de falhas. Nesses casos, para se realizar a manutenção preventiva ou reparação dos equipamentos, os sistemas são desligados;
- **TIER II**: é um pouco mais suscetível às interrupções planejadas ou não. Possui *nobreak* e geradores na estrutura de redundância N+1, porém em um único segmento. É preciso haver o desligamento dos sistemas de energia quando se faz necessária a manutenção;
- **TIER III**: possui equipamentos de refrigeração e alimentação de energia redundantes, porém com apenas um equipamento de cada segmento ligado, possibilitando atividades planejadas sem interromper a operação. O TIER III ainda está sujeito às falhas de operação e de componentes;
- **TIER IV**: possui equipamentos de refrigeração e alimentação de energia redundantes e ativos proporcionando tolerância às falhas. No *data center* TIER IV, equipamentos que não são construídos com múltiplas entradas de energia devem utilizar uma chave de transferência automática para que não haja nenhum tipo de interrupção. O TIER IV suporta atividades planejadas de manutenção sem interrupção, pois possui redundância 2(N+1).

2.2 Dependabilidade

A dependabilidade [7, 8] de um sistema pode ser entendida como a capacidade de fornecer um conjunto de serviços de forma confiável. Sendo assim, a dependabilidade pode indicar a qualidade de um serviço fornecido por um sistema e a confiança depositada nesse serviço [9].

Alguns dos atributos principais ligados à dependabilidade [10] são: confiabilidade, disponibilidade, segurança de funcionamento, segurança, mantenabilidade. Para este trabalho o atributo foco da dependabilidade, utilizado na otimização, será a disponibilidade que será avaliada através de modelos de redes de petri estocásticas (SPN) e diagramas de bloco de confiabilidade (RBD).

A disponibilidade é a probabilidade do sistema se manter em funcionamento levando em consideração a ocorrência de falhas e reparo de dispositivos que compõe tal sistema [11]. A definição de disponibilidade pode ser expressada pela relação entre o tempo ativo e a soma do tempo ativo (*uptime*) com o tempo inativo (*downtime*). *Uptime* é o período de tempo em que o sistema está operacional, *downtime* é o período de tempo em que o sistema não está operacional devido a ocorrência de um defeito ou atividade de reparo, e o *uptime* + *downtime* é o período de tempo de observação do sistema [12].

Sempre que não for possível obter o tempo ativo e o tempo inativo, a disponibilidade pode ser avaliada em função dos tempos médios para falhar e recuperar o sistema. O MTTF (Tempo Médio Para Falha) é medido pelo tempo médio entre a ocorrência de falhas, e o MTTR (Tempo Médio Para Reparo) é o tempo médio gasto para reparar uma falha [13]. Portanto, a disponibilidade pode ser computada pela Equação 1.

$$Disp = \frac{MTTF}{MTTF + MTTR} \tag{1}$$

2.3 Exergia e Sustentabilidade

A Segunda Lei da Termodinâmica [14] afirma que há sempre uma perda ao se converter um tipo de energia em outro. Ligada a este conceito, a exergia quantifica a energia que pode ser convertida em trabalho útil [14, 15]. Sendo assim, exergia pode ser usada para estimar a eficiência da conversão de energia de um sistema. A exergia é calculada como o produto de energia por um fator de qualidade de acordo com a Equação 2.

$$Exergia = Energia \times F \tag{2}$$

Em geral, a análise de exergia, no contexto da sustentabilidade, procura maximizar a eficiência da utilização de energia ou avaliar a degradação dos recursos naturais [16]. Este trabalho mede a exergia destruída. Quanto menor o valor desta métrica, menor o seu impacto em termos de sustentabilidade [17].

2.4 Redes de Petri

As redes de Petri (PN) [18] são uma família de formalismos adequados para modelar vários tipos de sistemas, uma vez que a simultaneidade, a sincronização, os mecanismos de comunicação, bem como os atrasos determinísticos e probabilísticos são naturalmente representados.

Este trabalho adota uma extensão particular das redes de Petri, as redes de Petri Estocásticas (SPN) [19], que permitem a associação de tempo às transições, e o respectivo espaço de estados pode ser convertido em uma cadeia de Markov de tempo contínuo (CTMC) [20]. Os tempos associados às transições temporizadas são distribuídos exponencialmente, enquanto que as transições imediatas não possuem tempo [12]. Além disso, SPN permite a adoção de técnicas de simulação para a obtenção de métricas de confiabilidade como uma alternativa para a geração da cadeia de Markov [21].

Os elementos de uma SPN (ver Figura 1) são os mesmos de uma PN: arcos, lugares, transições e *tokens* com a adição de mais dois elementos: transição temporizada (ver Figura 1-e) e o arco inibidor (ver Figura 1-f). O arco inibidor é um tipo de arco especial que possui um pequeno círculo branco

em uma das extremidades em vez de uma seta, e, geralmente, é usado para desativar transições se houver *tokens* (ou ativar quando não houver) presentes em um lugar [22]. Níveis diferentes de prioridade podem ser atribuídos às transições imediatas. Além disso, as transições imediatas tem maior prioridade que as transições temporizadas[12]. As prioridades podem solucionar ocorrências de confusão entre transições [23]. Além disso, regras de habilitação associadas às transições imediatas podem solucionar situações de conflito entre transições [24, 23].

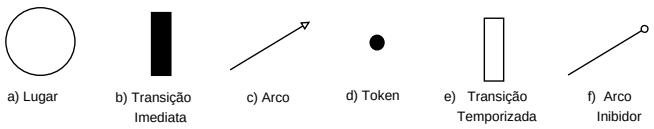

Figura 1. Elementos de uma Rede de Petri Estocástica.

Um exemplo de modelo em SPN para a obtenção da disponibilidade, é o modelo *Cold Standby*, que consiste num sistema redundante de espera composto por um módulo de reposição não ativo que é para ser ativado quando o módulo ativo principal falhar. Para que o módulo reserva entre em funcionamento, é necessário um período entre a ativação. Por esse motivo, a redundância *cold standby* é denominada passiva [21]. A Figura 2 representa o modelo SPN deste sistema modelado a partir de um conjunto composto por um módulo principal (PRI) e um módulo de *backup* (BKP), que inclui quatro lugares: *PRI_ON*, *PRI_OFF*, *BKP_ON*, *BKP_OFF*. Esses lugares representam os estados operacionais e os estados de falhas dos principais módulos respectivamente [25]. O módulo de backup é inicialmente desativado, e somente após a falha do módulo principal é que o módulo de backup é acionado. Quando o módulo principal falha, a transição *TACT* é acionada para ativar o módulo de backup [25].

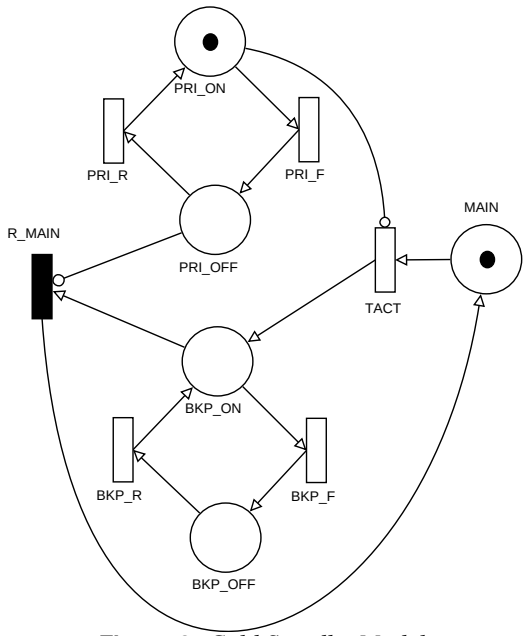

Figura 2. *Cold Standby Model*.

O lugar *PRI_ON* representa o estado de funcionamento do módulo principal. Quando este módulo falha, a transição *PRI_F* é disparada. Uma vez disparada, um *token* é removido do lugar *PRI_ON* e colocado no *PRI_OFF*. Na ausência de *tokens* em *PRI_ON* a transição *TACT* é habilitada. Com o disparo desta transição o módulo reserva (BKP) é ativado removendo um *token* do lugar *MAIN* e o colocando em *BKP_ON*. A disponibilidade desse sistema pode ser computada através da probabilidade de termos o módulo principal ou o de backup em funcionamento. Isso é representado em SPN pela expressão A = P{(#PRI_ON = 1) OR (#BKP_ON = 1)}

2.5 Diagrama de Blocos de Confiabilidade (RBD)

O diagrama de bloco de confiabilidade (RBD) [26] é um modelo combinatório inicialmente proposto como uma técnica para o cálculo da confiabilidade de um sistema usando diagramas de blocos. A estrutura do RBD estabelece a interação lógica entre os componentes que definem quais combinações de elementos falhos e ativos podem sustentar a operação do sistema. A Figura 3 ilustra dois exemplos, em que os blocos independentes são dispostos em série e em paralelo.

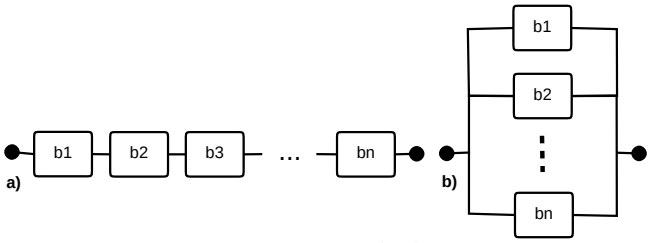

Figura 3. Um exemplo de RBD.

Na disposição em série (ver Figura 3-a), o sistema somente funcionará se todos os seus componentes estiverem ativos. Para um arranjo com n componentes, a confiabilidade (P_s) é obtida através da Equação 3.

$$P_s = \prod_{i=1}^{n} P_i \tag{3}$$

onde P_i representa a confiabilidade $R_i(t)$ (disponibilidade instantânea $(A_i(t))$ ou disponibilidade de estado estacionário (A_i)) do bloco *bi*.

Na composição em paralelo (ver Figura 3-b), o sistema funcionará se um de seus componentes estiver funcionando. Para uma arranjo com n componentes, a confiabilidade (P_p) é obtida através da Equação 4.

$$P_p = 1 - \prod_{i=1}^{n} (1 - P_i) \tag{4}$$

onde P_i representa a confiabilidade $R_i(t)$ (disponibilidade instantânea $(A_i(t))$ ou disponibilidade de estado estacionário (A_i)) de *bi*.

2.6 Modelo de Fluxo de Energia (EFM)

O Modelo de Fluxo de Energia (EFM) tem o objetivo de representar o fluxo de energia elétrica entre os componentes do sistema, considerando a eficiência e a capacidade máxima que cada componente pode fornecer ou extrair [1].

A Figura 4-a mostra um exemplo de uma infraestrutura elétrica. Supondo que a potência exigida pelo sistema de TI do *data center* é de 30 kW (valor associado ao *target T*) e a capacidade de potência máxima (Figura 4-b) dos nós internos (componentes), *UPSs* e *power strips*, é de 20 kW para cada. A Figura 4-c representa um possível fluxo de energia, no qual cada UPS fornece potência de 15 kW, que é transferida para os *Subpanels*(15 kW). No exemplo da Figura 4-d, apenas um *Subpanel* é considerado. Este sistema suporta apenas 20 kW de potência exigida (*target* T). Portanto, o sistema não é capaz de lidar com a potência exigida [2].

A Figura 4-e representa um sistema com dois *Subpanels*. O *Subpanel1* é capaz de fornecer duas vezes mais energia que o *Subpanel2* devido ao peso associado a aresta do grafo. Além disso, os algoritmos são propostos para calcular o impacto de custo e sustentabilidade através do EFM [2].

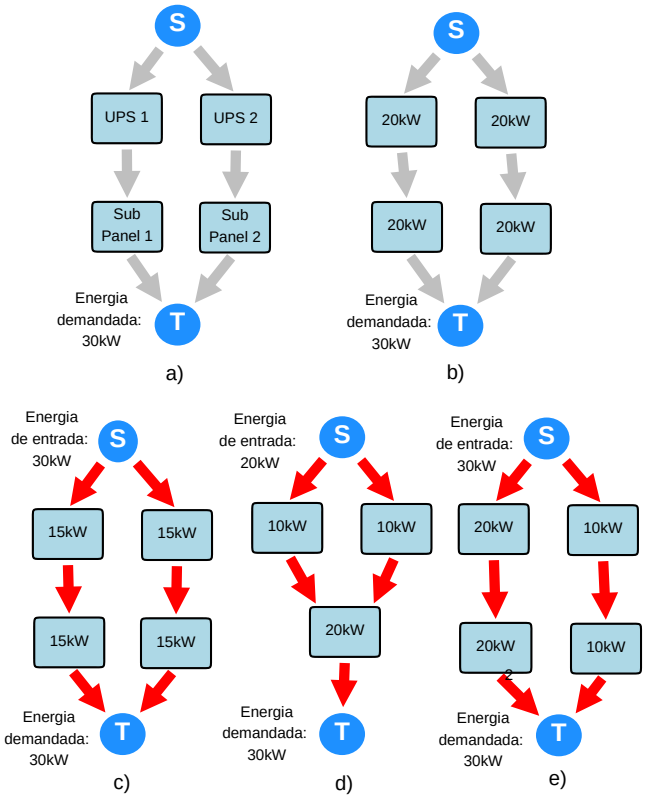

Figura 4. (a) Exemplo de infraestrutura de carga; (b) Capacidade máxima de energia; (c) Fluxo de energia realizado com sucesso; (d) Fluxo de energia falho; e (e) exemplo de peso nas arestas.

2.7 Calculando a exergia operacional

O consumo de exergia operacional pode ser entendido como a energia dissipada por cada item de equipamento que não pode ser convertido em trabalho útil. A seguinte equação representa a exergia operacional do sistema [27].

$$Ex_{op} = \sum_{i=1}^{n} Ex_{op_i} \times T \times (Disp + \alpha(1 - Disp)) \qquad (5)$$

onde Ex_{op_i} é a exergia operacional de cada dispositivo, T é o período de análise, $Disp$ é a disponibilidade do sistema e α é o fator que representa a quantidade de energia que continua a ser consumida depois que um componente falha.

2.8 Calculando o custo

O custo é a soma do custo de aquisição (CA) com o custo operacional (CO). O custo de aquisição é calculado pelo somatório dos preços de varejo (P_a) dos equipamentos que compõem a arquitetura. Já o custo operacional, para este trabalho, considera apenas o custo referente ao consumo de energia do sistema de acordo com a Equação 6 [2]. Outros fatores, no entanto, também podem ser incluídos.

$$CO = E_{cons} \times T \times P_{energia} \times (Disp + \alpha(1 - Disp)), \qquad (6)$$

onde E_{cons} é a energia elétrica consumida pelo sistema, T é o período assumido; $P_{energia}$ corresponde ao preço da energia, $Disp$ corresponde a disponibilidade do sistema e α é o fator que representa a quantidade de energia que continua a ser consumida depois que um componente falha.

2.9 Algoritmos Genéticos

A lógica dos algoritmos genéticos funciona de forma semelhante ao processo de cruzamento biológico de cromossomos que compartilham informação genética para criar um novo indivíduo. No final, chegam a obter um indivíduo de alta adaptação, que no jargão de matemática não significa uma solução ótima, mas sim a "melhor solução" encontrada [28].

Os algoritmos genéticos se diferenciam dos algoritmos tradicionais de otimização por serem [29]: estocásticos; de busca múltipla (possível obter várias soluções); de convergência pouco sensível à população inicial, não existem restrições no espaço de busca; algoritmos intrinsecamente paralelos capazes de operar simultaneamente com várias soluções, o que possibilita geralmente um funcionamento mais rápido nas execuções; algoritmos que resultam poucas soluções falsas.

Para se otimizar ao máximo um determinado problema, todo o processo, ilustrado na Figura 5, deve ser repetido por várias gerações (ciclos) com o objetivo de se chegar a uma solução ótima. O fluxograma do algoritmo genético pode ser representado pelas etapas descritas na Figura 5.

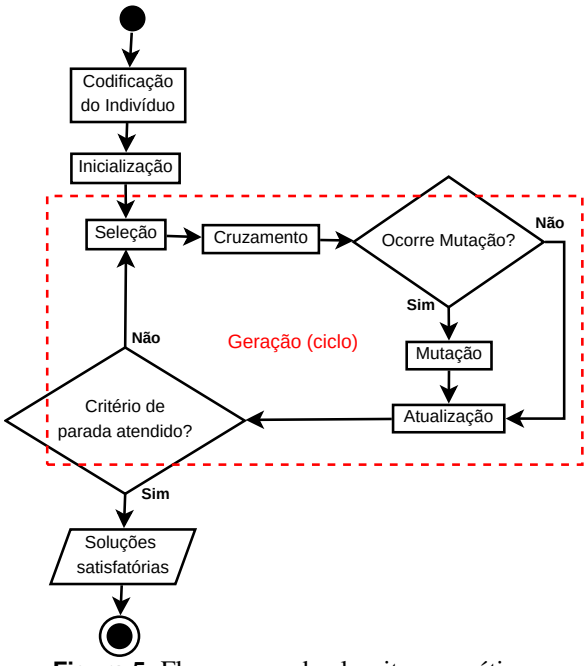

Figura 5. Fluxograma do algoritmo genético.

Um algoritmo genético passa pelas seguintes etapas a seguir [30]:

- **codificação do indivíduo**: Inicialmente, cria-se um indivíduo para representar a solução do problema [31]. Codificando o indivíduo como uma estrutura binária é obtido o cromossomo ilustrado na Figura 6. Neste exemplo é apresentado um indivíduo, para $x = 76$, transformado em binário e ajustado ao genótipo do cromossomo.

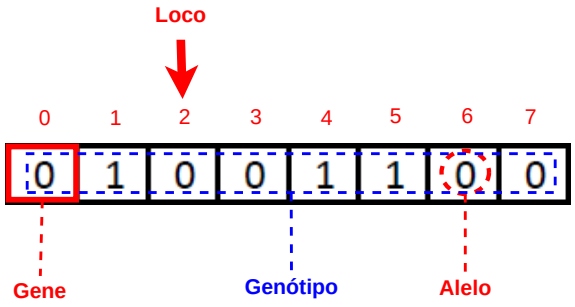

Figura 6. Exemplo de Cromossomo.

O gene é a característica, e para esse exemplo possui um espaço de busca de 0 ou 1. Os 0*s* ou 1*s*, propriamente ditos, são os valores dessa característica denominados alelos. O loco é a posição no vetor cromossômico em que se encontra um determinado gene.

- **inicialização**: basicamente produz uma população de soluções aleatórias (indivíduos) para o problema a ser otimizado.

- **avaliação**: avalia-se a aptidão dos indivíduos através de uma função *fitness*, ou função objetivo, para se estabelecer a qualidade de cada solução gerada resultando numa pontuação para cada indivíduo.

- **seleção**: indivíduos são selecionados para a reprodução a partir de sua aptidão.

- **cruzamento**: características das soluções são recombinadas, gerando novos indivíduos. Um ponto de corte é determinado aleatoriamente para, a partir dele, haver a troca dos genes entre os indivíduos. O exemplo ilustrado na Figura 7 representa o processo que inicia com dois cromossomos x e y com comprimento L, escolhe-se, aleatoriamente, um número P (ponto de corte) tal que $0 < P < L$. O filho s_0 receberá os genes de 0 até o índice (P) do cromossomo x, e os genes de $P+1$ até L do cromossomo y. O filho s_1 herdará os genes de $P+1$ até L do cromossomo x e os genes de 0 até P do cromossomo y.

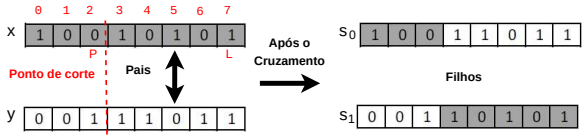

Figura 7. Cruzamento.

- **mutação**: existe uma probabilidade de que características dos novos indivíduos resultantes do processo de cruzamento sejam alteradas, acrescentando assim variedade à população. Normalmente são selecionados, aleatoriamente, dois genes que são permutados entre si.

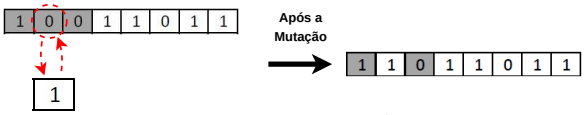

Figura 8. Mutação *flip*.

O objetivo da mutação é introduzir novas informações nas populações, impedindo que elas fiquem saturadas de cromossomos similares, ao passo que visa elevar a variedade populacional possibilitando uma maior exploração do espaço de busca [32].

- **atualização ou elitismo**: os indivíduos criados nesta geração são inseridos na população. O elitismo é uma estratégia com o propósito de preservar as melhores soluções encontradas na geração atual. Em sua lógica mais trivial, o elitismo conserva os n_{elit} melhores indivíduos da geração atual, mantendo-os na população para a próxima geração sem nenhuma mudança. Dessa forma, os melhores indivíduos não são só passados de uma geração para outra, como também colaboram na geração dos novos indivíduos da nova geração [31].

- **finalização**: verifica se as condições de encerramento da evolução foram atingidas, retornando para a etapa de avaliação em caso negativo e encerrando a execução em caso positivo.

3. Trabalhos Relacionados

Vários trabalhos vêm sendo desenvolvidos com o objetivo de reduzir o consumo de energia e aperfeiçoar técnicas e modelos sustentáveis em *data centers*. Além disso, um outro aspecto que é levado em consideração é a necessidade de se ter o sistema sempre disponível, e assim, pesquisas relacionadas a esse tema também vêm atraindo atenção da comunidade acadêmica e das empresas. Técnicas de otimização vêm sendo aplicadas para otimizar tanto a disponibilidade, como o consumo de energia e a sustentabilidade [27].

Callou et. al [2] usam uma abordagem de otimização multi-objetivo através do "procedimento de busca gulosa adaptativa randomizada" (*GRASP*) para otimização de arquiteturas de *data center*. Modelos em SPN, RBD e EFM foram criados através do ambiente Mercury e a partir de uma lista de equipamentos com diferentes parâmetros (MTTF, custo, eficiência energética), foi utilizado o algoritmo GRASP de maneira a maximizar a disponibilidade, minimizando o custo e o impacto ambiental. Diferente desse trabalho, esse trabalho faz uso de algoritmos genéticos como heurística de otimização que resulta sempre numa solução próxima da ótima, enquanto o *GRASP*, que é baseado em uma busca randômica, pode retornar uma solução não satisfatória para a otimização.

Ferreira et. al [17] propõem um algoritmo de distribuição de carga de energia (*PLDA*), baseado no algoritmo de *Ford-Fulkerson*, que minimiza o consumo de energia em um modelo proposto de fluxo de energia. A aplicabilidade do PLDA proposto foi verificada por um estudo de caso que analisa seis arquiteturas elétricas de nuvem privada observando resultados significativos, como a redução no consumo de energia de 10,7% e uma redução no impacto ambiental. Porém, Ferreira et. al não focam na otimização da disponibilidade e do custo.

Kar et. al [33] propõem a otimização, através de algoritmos genéticos, do consumo de energia, poluição ambiental, e do tempo de resposta das tarefas na alocação de recursos de escalonamento em *data centers*. Os autores não sugerem a otimização da disponibilidade e do custo, diferentemente deste trabalho.

Para otimizar o consumo de energia, o desempenho do sistema e a confiabilidade, Luo et. al [34] sugerem um algoritmo de agendamento de recursos. Esse algoritmo se fundamenta numa abordagem de modelagem correlacionada empregando modelos de *Semi-Markov*, transformação de *Laplace-Stieltjes* e uma abordagem bayesiana para analisar as relações entre confiabilidade e desempenho, e as relações entre confiabilidade e consumo de energia. Os autores focam na otimização da confiabilidade, consumo e custo, aliados ao desempenho da infraestrutura de TI. Não foi o foco do trabalho a otimização da infraestrutura elétrica.

Bosse et. al [35] propõem a otimização de um projeto de serviço de TI tolerante a falhas para minimizar os custos e as emissões de gases do efeito estufa (GEE) sujeitos a um certo nível de disponibilidade em um *data center*. Para otimizar, um problema de alocação de redundância (RAP) multi-objetivo é modelado a partir do projeto, e otimizado através de algoritmos genéticos. Dados de um estudo de caso real são considerados para conduzir experimentos demonstrando que as emissões de GEE podem ser reduzidas consideravelmente. Os autores, porém, não otimizam a disponibilidade, apenas atribuem um certo grau de disponibilidade para considerar que o projeto a ser otimizado é tolerante a falhas.

Ziafat e Babamir [36] apresentam uma abordagem utilizando o algoritmo multi-objetivo NSGA-II e o agrupamento *K-means* para minimizar o custo e maximizar a disponibilidade na escolha de um *data center*. A abordagem proposta foi empregada em alguns *data centers* distribuídos geograficamente. Os resultados foram comparados com otimização através de busca gulosa e de algoritmos comuns aleatórios para mostrar que a otimização sugerida supera as outras duas heurísticas. Os autores não propõem a sustentabilidade como parâmetro para a otimização da escolha do *data center*.

Pode-se concluir que poucos trabalhos fazem esse estudo integrado de custo, disponibilidade e sustentabilidade. Como observado em [2], [34] e [35], estes trabalhos utilizam técnicas de otimização para esta integração dos parâmetros estudados. Este trabalho se difere porque propõe um algoritmo genético como heurística de otimização destes parâmetros.

4. Metodolgia

O objetivo desta Seção é apresentar as etapas adotadas para se realizar a otimização das arquiteturas de *data center* com o algoritmo genético poposto. Vale ressaltar que o *Algoritmo Genético* se comunica com a ferramenta Mercury [1] (e os modelos feitos nesta ferramenta) para poder realizar a otimização dos resultados. Mercury é uma ferramenta utilizada para modelar sistemas através de SPN, CTMC, EFM e RBD [4]. Esta metodologia se destina a projetistas de *data center*.

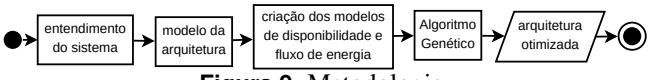

Figura 9. Metodologia.

A Figura 9 ilustra as etapas da metodologia adotada para a otimização integrada do custo, disponibilidade e sustentabilidade das infraestruturas elétricas de *data center*. A primeira etapa consiste no entendimento do sistema para a criação de um modelo geral, a partir da arquitetura de *data center* que se deseja otimizar, que servirá como base para auxiliar na criação dos modelos RBD, EFM e SPN, e na codificação do indivíduo (modelagem do cromossomo).

4.1 Criação dos Modelos

Os modelos gerais (segunda etapa) são uma abstração mais simples das arquiteturas para facilitar o entendimento do fluxo

e as disposições em série e paralelo dos componentes, facilitando assim a criação dos modelos de disponibilidade e fluxo de energia como ilustra o exemplo da Figura 10.

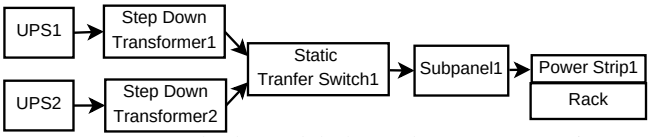

Figura 10. Exemplo de modelo baseado em uma arquitetura elétrica de *data center* [2].

Na terceira etapa, a partir dos modelos das arquiteturas, são criados, no Mercury, modelos em SPN ou RBD, para obter a disponibilidade, e EFM para obter a exergia operacional e o custo.

Os modelos em SPN são utilizados para obter a disponibilidade quando a arquitetura possui redundância com dependência entre seus componentes (ver Figura 11). A estratégia de modelagem hierárquica consiste em obter a disponibilidade de partes da arquitetura, através dos modelos em SPN, e atribuir a um bloco do modelo em RBD para obter a disponibilidade de toda a arquitetura. Essa disponibilidade é utilizada como entrada no modelo EFM para calcular a exergia e o custo.

O exemplo da Figura 11-a apresenta um modelo criado para representar uma arquitetura elétrica básica. Essa arquitetura possui um UPS e um gerador (GEN), na qual o UPS está ligado e o gerador permanece desligado, sendo acionado em caso de falha do UPS. A disponibilidade obtida através desse modelo SPN, Figura 11-b, é obtida pela expressão P(#UPS_ON = 1) OR (#GEN_ON = 1). Essa disponibilidade é utilizada para representar um bloco do modelo RBD (ver Figura 11-c) e, assim, se computar a disponibilidade da arquitetura.

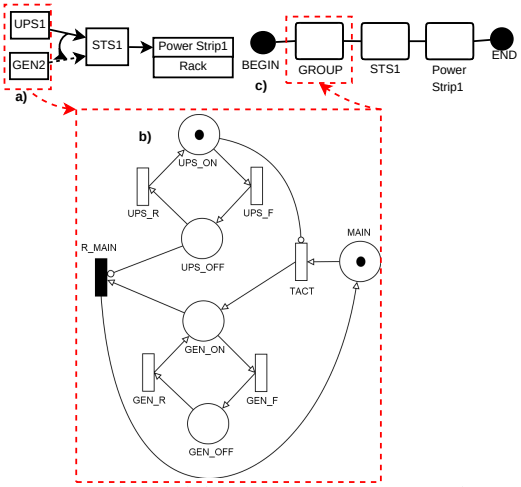

Figura 11. Uso do modelo SPN e RBD para se obter a disponibilidade representando redundância de componentes.

Esse processo de cálculo prévio da disponibilidade de integração entre os três modelos é possível através da "Linguagem de *Script* do Mercury" [37]. O exemplo da Figura 12 ilustra como funciona a integração dos modelos através do *script* do Mercury. Na Figura 12-a, a disponibilidade é calculada no modelo em SPN e em seguida passada como atributo do bloco UPS do modelo em RBD na Figura 12-b. O modelo em RBD calcula, juntamente com os MTTFs e MTTRs dos outros blocos, a disponibilidade de todo sistema. Essa disponibilidade é passada como parâmetro para o modelo EFM ilustrado na Figura 12-c que de posse dos valores de custo da energia elétrica (por KWh) e da eficiência energética dos equipamentos, avalia o custo e a exergia.

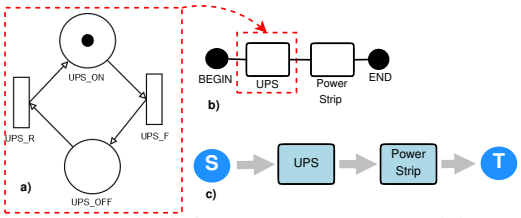

Figura 12. Exemplo de integração entre os modelos em SPN, RBD e EFM.

Para representar o exemplo da Figura 12 na linguagem de *script* do Mercury, é necessário, primeiro declarar e montar os modelos a partir da sintaxe do *script*. A Figura 13 mostra a representação do modelo em SPN da Figura 12-a.

A primeira linha declara o tipo de modelo (que no caso é SPN) e o nome do mesmo. Nas linhas 3 e 4, são declarados e denominados os lugares e a quantidade de *tokens*. Da linha 6 até a linha 15, são definidas as transições e seus lugares de entrada (*inputs*), de saída (*outputs*), e o *delay* que para a transição de falha (UPS_F) representa o MTTF, e para transição de reparo (UPS_R) representa o MTTR. A linha 16 denomina a disponibilidade como a métrica A1, e define o tipo de análise e a expressão de cálculo da disponibilidade.

```
1  SPN SPNModel{
2
3      place UPS_OFF;
4      place UPS_ON( tokens= 1 );
5
6      timedTransition UPS_F(
7          inputs = [UPS_ON],
8          outputs = [UPS_OFF],
9          delay = tempoFalha
10     );
11     timedTransition UPS_R(
12         inputs = [UPS_OFF],
13         outputs = [UPS_ON],
14         delay = tempoReparo
15     );
16     metric A1 = stationaryAnalysis( expression = "P{#UPS_ON=1}" );
17 }
```

Figura 13. Representação da Figura 12-a na linguagem de *script* do Mercury.

A Figura 14 representa o modelo em RBD referente a Figura 12-b na linguagem de *script* do Mercury. Pode-se observar que na linha 19 é definido o tipo e o nome do modelo. Na linha 21, o primeiro bloco (UPS) é definido como *hierarchy* pelo fato de receber, como parâmetro, a disponibilidade, que, no caso, é previamente calculada pelo modelo em SPN descrito no *script* (ver Figura 13) através do método *solve*. O

bloco *PowerStrip* é definido na linha 22 e recebe como atributos o MTTF e o MTTR. O método *series* (linha 23) determina que os blocos, descritos nele, estão em série. O método *top*, na linha 25, define a série principal do modelo. Da linha 27 à 30, são definidas as métricas que poderão ser calculadas no modelo. A métrica *av*, por exemplo, calcula a disponibilidade do modelo.

```
19 RBD RBDModel{
20
21     hierarchy UPS( availability = solve (SPNModel, A1));
22     block PowerStrip( MTTF = mttf1, MTTR = mttr1);
23     series s0(UPS, PowerStrip );
24
25     top s0;
26
27     metric av = availability;
28     metric rel = reliability( time = 8760 );
29     metric mttf = mttf;
30     metric mttr = mttr;
31 }
```

Figura 14. Representação da Figura 12-b na linguagem de *script* do Mercury.

Obtida a disponibilidade através dos modelos SPN e RBD, a mesma é passada como parâmetro para o modelo EFM ilustrado na Figura 12-c através do mesmo *script* do Mercury como mostra a Figura 15.

```
33 EFM EFMModel{
34     component SourcePoint1(
35         type = "SourcePoint",
36         parameters = (
37             efficiency = 100.0,
38             retailPrice = 0.0
39         )
40     );
41     component UPS_5kVA1(
42         type = "UPS_5kVA",
43         parameters = (
44             maxPower = 5.0,
45             efficiency = e1,
46             retailPrice = r1,
47             embeddedEnergy = 3.1392
48         )
49     );
50     component PowerStrip1(
51         type = "PowerStrip",
52         parameters = (
53             maxPower = 5.0,
54             efficiency = e2,
55             retailPrice = r2,
56             embeddedEnergy = 0.35568
57         )
58     );
59     component TargetPoint1(
60         type = "TargetPoint",
61         parameters = (
62             efficiency = 100.0,
63             retailPrice = 0.0
64         )
65     );
66     arc SourcePoint1 -> UPS_5kVA1;
67     arc UPS_5kVA1 -> PowerStrip1;
68     arc PowerStrip1 -> TargetPoint1;
69
70     metric ic = initialCost( eletricityCost = 0.11);
71     metric oc = operationalCost( eletricityCost = 0,11,
72     availability = solve (RBDModel, av), time = 8760 );
73     metric ee = embeddedExergy;
74     metric oe = operationalExergy( time = 8760,
75     availability = solve (RBDModel, av));
76     metric tc(ic + oc);
77     metric te(ee + oe);
78 }
```

Figura 15. Representação da Figura 12-c na linguagem de *script* do Mercury.

A definição do modelo EFM inicia na linha 33. Da linha 34 à 40 e da linha 59 à 64 são definidos o *SourcePoint* e o *TargetPoint*, respectivamente. Da linha 41 à 58 são definidos os componentes, os tipos (*type*) de cada e os atributos (*parameters*) que já possuem um valor por padrão no Mercury. Neste trabalho, os parâmetros que serão utilizados no modelo EFM são a eficiência (*efficiency*) e o preço de aquisição (*retailPrice*), e que no exemplo são atribuídos pelas variáveis $e1$, $e2$, $r1$ e $r2$. Da linha 66 à 68, são definidos os arcos que ligam os componentes. As métricas são estabelecidas da linha 70 à 77. Pode-se observar que nas linhas 71 e 72 é utilizada a disponibilidade antes calculada no modelo RBD como parâmetro para obter o custo operacional. A disponibilidade também é utilizada para o cálculo da exergia operacional nas linhas 74 e 75.

A linguagem de *scrpit* do Mercury possui uma função principal (*main*) que invoca os demais modelos e métodos declarados e imprime os resultados obtidos (ver Figura 16).

```
80 main {
81
82     A1 = solve(SPNModel, A1);
83     println(A1);
84
85     av = solve(RBDModel, av);
86     rel = solve(RBDModel, rel);
87     mttf = solve(RBDModel, mttf);
88     mttr = solve(RBDModel, mttr);
89
90     println("Availability: " .. av );
91     println("Reliability: " .. rel );
92     println("Mean time to failure: " .. mttf );
93     println("Mean time to repair: " .. mttr );
94
95     ic = solve(EFMModel, ic);
96     println("Acquisition Cost: " .. ic);
97     oc = solve(EFMModel, oc);
98     println("Operational Cost: " .. oc);
99     tc = solve(EFMModel, tc);
100    println("Total Cost: " .. tc);
101
102    println("");
103
104    oe = solve(EFMModel, oe);
105    println("Operational Exergy: " .. oe);
106
107 }
```

Figura 16. Função principal do *script*.

4.2 Algoritmo Genético (AG)

A quarta etapa consiste na aplicação do algoritmo genético. O objetivo é utilizar o AG integrado com o Mercury para avaliar os modelos das arquiteturas e obter os parâmetros desejados para a otimização. A Figura 17 ilustra a integração entre o algoritmo genético desenvolvido, a linguagem de *script* e o Mercury.

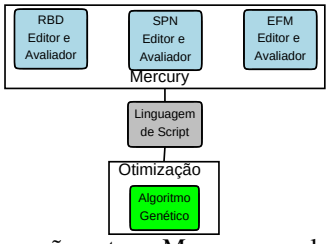

Figura 17. Integração entre o Mercury e o algoritmo genético.

No fluxograma ilustrado na Figura 18, pode-se observar as etapas do AG proposto, sendo a integração com o Mercury efetivada nas etapas "Geração da população inicial", "Cruzamento" e "Mutação", nas quais os cromossomos são gerados e alterados. Tanto a integração com o Mercury como as demais etapas são melhor detalhadas ao decorrer desta seção.

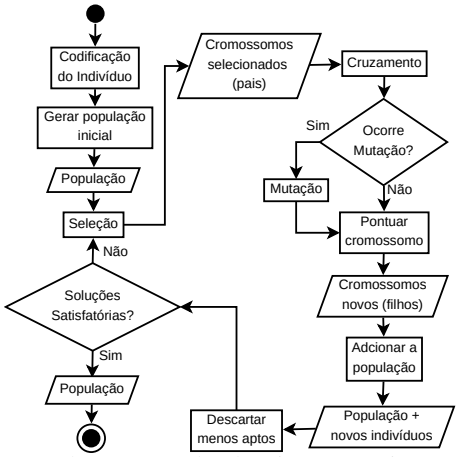

Figura 18. Fluxograma do algoritmo genético proposto.

Cada etapa da Figura 18 é definida pelo Algoritmo 1 que apresenta o algoritmo genético proposto neste trabalho. Cada parte deste algoritmo será detalhada no decorrer desta seção.

Algoritmo 1 Algoritmo Genético

```
1:  populacao[ ] ← gerarPopulacao(listaEquipamentos)        ▷ Geração da população inicial
2:  para i ← 0 até qGen faça                                ▷ qGen = quantidade de gerações
3:      selecionados[ ] ← selecaoTorneio(populacao)
4:      filhos[ ] ← cruzamento(selecionados)
5:      para j ← 0 até tamanho(filhos[ ]) faça
6:          filhos[j] ← mutacao(filhos[j])                  ▷ a mutação pode ou não ocorrer
7:          filhos[j] ← pontuarCromossomo(filhos[j])
8:      fim para
9:      populacao[ ] ← populacao[ ] + filhos[ ]
10:     populacao[ ] ← descartarPiores(populacao[ ])
11: fim para
12: retorna populacao
```

No algoritmo genético proposto, o indivíduo é codificado com um único cromossomo, sendo assim, pode-se considerar que o indivíduo é o cromossomo. O cromossomo é estruturado de acordo com o modelo de maneira a agrupar todos seus componentes em um vetor único de genes. Cada gene é alocado em uma posição do vetor, independentemente de estarem em série ou paralelo, conforme pode ser visto na Figura 19. Desta forma, torna-se mais viável o cruzamento dos cromossomos. Como no exemplo da Figura 19, as Séries 1 e 2 são agrupadas no vetor de genes, uma após a outra, e em seguida, concatenadas com a Série 3 para formar o cromossomo.

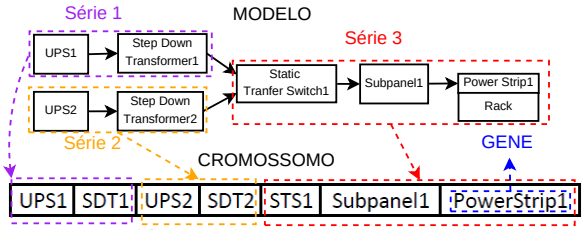

Figura 19. Codificação do cromossomo a partir do exemplo da Figura 10.

Após a codificação do cromossomo, inicia-se a geração da população inicial na qual cada cromossomo possui uma solução aleatória. Para esta fase do algoritmo genético, é utilizado um arquivo de entrada contendo uma lista de equipamentos correspondentes aos componentes presentes na arquitetura para a geração dos indivíduos. Este arquivo possui os dados de identificação dos equipamentos bem como os parâmetros utilizados por este algoritmo genético. A geração da população inicial é representada pelo fluxograma da Figura 20.

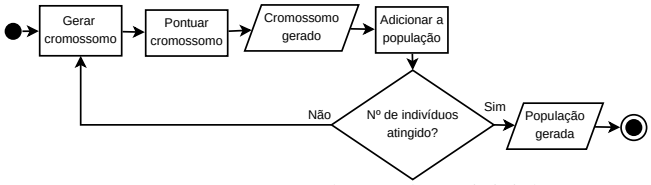

Figura 20. Geração da população inicial.

O Algoritmo 2 apresenta o passo-a-passo da geração da população inicial. Pode-se observar, na linha 2, que o tamanho da população (quantidade de cromossomos), é definido de acordo com o estudo de caso a ser aplicado. O retorno desta função é um vetor de cromossomos, cada um contendo uma solução aleatória.

O cromossomo é um objeto formado, basicamente, por um vetor de genes e a pontuação que representa o valor de sua aptidão. Para gerar a população inicial é necessário gerar cada cromossomo, o que implica em sortear os equipamentos para preencher cada gene e, em seguida, pontuar o cromossomo a partir da avaliação dos modelos de disponibilidade e fluxo no Mercury.

Algoritmo 2 Geração da população inicial

```
1: função GERARPOPULACAO(listaEquipamentos)
2:     para i ← 0 até tamPop faça        ▷ tamPop = tamanho desejado da população
3:         cromossomo ← gerarCromossomo(listaEquipamentos)
4:         cromossomo ← pontuarCromossomo(cromossomo)
5:         populacao[ ].adicionar(cromossomo)
6:     fim para
7:     retorna populacao[ ]
8: fim função
```

O Algoritmo 3 apresenta as etapas da geração do cromossomo. A entrada da função é um vetor com equipamentos gerado a partir de um arquivo. O retorno é um cromossomo ainda não pontuado e com o vetor de genes preenchido com o componente da arquitetura correspondente a cada posição do vetor.

Algoritmo 3 Geração do cromossomo

```
1: função GERARCROMOSSOMO(listaEquipamentos[ ])
2:     para i ← 0 até tamanho(cromossomo.genes[ ]) faça
3:         cromossomo.genes[i] ← sortearEquipamento(listaEquipamentos[ ])
4:     fim para
5:     retorna cromossomo
6: fim função
```

A Figura 21 ilustra como cada equipamento é sorteado e alocado em um gene. Para cada equipamento presente no modelo, esse arquivo de entrada fornece equipamentos com diferentes MTTFs, custo de aquisição e eficiência energética. A partir da leitura deste arquivo, o algoritmo genético instancia cada cromossomo da população de acordo com modelo da arquitetura proposta (ver Figura 19). Em cada posição do vetor cromossômico, é alocado aleatoriamente, a partir da lista de equipamentos, um componente do mesmo tipo descrito no gene.

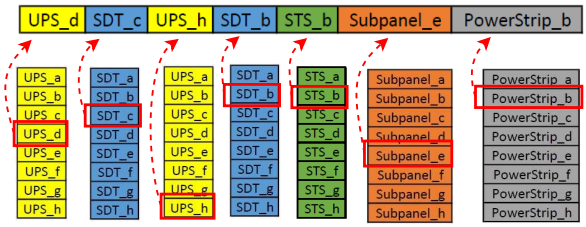

Figura 21. Geração de cromossomo aleatório.

Assim que gerado o cromossomo, ele precisa ser pontuado para mensurar sua aptidão. Os equipamentos presentes no cromossomo são atribuídos aos modelos, criados na etapa 3 da metodologia, RBD, SPN e EFM no Mercury. Em seguida, os modelos RBD e SPN são avaliados para se obter a disponibilidade da arquitetura em análise. Esse resultado é utilizado como parâmetro de entrada no modelo EFM, empregado para se quantificar a exergia operacional e o custo.

De posse destas métricas: disponibilidade, exergia operacional e custo, estas são utilizadas para o cálculo da função *fitness* multi-objetiva afim de pontuar o cromossomo com um valor que possa determinar a qualidade da solução obtida. Esse processo de pontuação pode ser observado no fluxograma ilustrado na Figura 22.

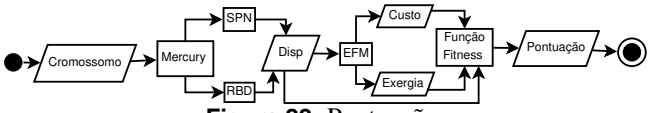

Figura 22. Pontuação.

A função *pontuarCromossomo* é descrita pelo Algoritmo 4. Na linha 2, a integração do algoritmo genético com o Mercury fica evidenciada pela função *scriptMercury*. Este método utiliza, como parâmetro de entrada, um arquivo contendo um *script* do Mercury que descreve os modelos em SPN, RBD e EFM (etapa 3, da metodologia), já integrados, da arquitetura a ser otimizada. Essa função transmite ao *script* os atributos de cada componente da arquitetura presente no cromossomo. Pode-se observar este processo nas linhas 5 à 8 com o MTTF, MTTR, custo de aquisição e a eficiência sendo passados para o *script*. Em seguida, o *script* calcula, a partir da avaliação dos modelos em SPN, RBD e EFM no Mercury, disponibilidade, exergia e custo (linhas 10 à 13). Por fim, esses parâmetros são passados para a função *Fitness* (linha 14) que retorna uma pontuação (valor de aptidão) para o cromossomo.

Algoritmo 4 Pontuar cromossomo

```
1: função PONTUARCROMOSSOMO(cromossomo)
2:     script = scriptMercury("script.mry")        ▷ Integração do AG com o Mercury
3:                                  ▷ script.mry é o arquivo onde o script foi escrito
4:     para i ← 0 até tamanho(cromossomo.genes[ ]) faça
5:         script ← script(cromossomo.genes[i].mttf)
6:         script ← script(cromossomo.genes[i].mttr)
7:         script ← script(cromossomo.genes[i].custoAquisicao)
8:         script ← script(cromossomo.genes[i].eficiencia)
9:     fim para
10:    disp ← script.executar(RBD, disponibilidade)
11:            ▷ RBD e disponibilidade: tipo de modelo e métrica respectivamente
12:    exergiaOp ← script.executar(EFM, exergiaOperacional)
13:    custoT ← script.executar(EFM, custoTotal)
14:    cromossomo.pontuacao ← funcaoFitness(disp, esergiaOp, custoT)
15:    retorna cromossomo
16: fim função
```

Para cada estudo de caso, uma função *Fitness* deve ser criada devido as diferenças nos graus de complexidade de cada caso. A função *Fitness* multi-objetiva equilibra as três métricas de maneira que, quanto maior a disponibilidade, menor a exergia e menor o custo, maior será a pontuação atribuída para a solução contida no cromossomo.

Após gerados e pontuados todos os cromossomos da população inicial, o próximo passo do algoritmo genético é a seleção. São sorteados quatro indivíduos da população, dos quais os dois melhores (com pontuação maior) são selecionados para realizar o cruzamento. Este método de seleção é conhecido como Torneio (ver Algoritmo 5).

Depois de selecionados os cromossomos (pais), começa o processo de cruzamento descrito no Algoritmo 6. É estabelecido, aleatoriamente, um ponto de corte (linha 4) para, a partir dele, haver o cruzamento dos genes dos pais para formar novos cromossomos (filhos). Esse ponto de corte é um valor inteiro que será atribuído ao índice onde o vetor de genes é cortado para a troca. Essa troca de genes é definida nas linhas 6 à 14. O *filho1* recebe os genes da posição 0 até a posição $pontoCorte - 1$ do *pai1* e os genes da posição *pontoCorte* até o fim do vetor de genes do *pai2*. Já o *filho2* recebe os genes da posição 0 até a posição $pontoCorte - 1$ do *pai2* e os genes da posição *pontoCorte* até o fim do vetor de genes do *pai1*. O retorno desta função é um vetor com os dois novos cromossomos gerados no cruzamento.

Algoritmo 5 Seleção Torneio

```
1: função SELECAOTORNEIO(populacao[ ])
2:     torneio[ ] ← sortear(populacao[ ], 4)    ▷ sorteia 4 cromossomos da população
3:     selecionados[ ] ← selecionarMelhores(torneio[ ], 2)    ▷ seleciona os dois melhores
4:     retorna selecionados[ ]
5: fim função
```

Algoritmo 6 Cruzamento

```
1: função CRUZAMENTO(selecionados[ ])
2:     pai1 ← selecionados[0]
3:     pai2 ← selecionados[1]
4:     pontoCorte ← sortearInteiro(tamanho(pai1.genes[ ]) - 1)
5:                                    ▷ sorteia um ponto de corte no cromossomo
6:     para i ← 0 até pontoCorte faça
7:         filho1.genes[ ].adcionar(pai1.genes[i])
8:     fim para
9:     para i ← pontoCorte até tamanho(pai2.genes[ ]) faça
10:        filho1.genes[ ].adcionar(pai2.genes[i])
11:    fim para
12:    para i ← 0 até pontoCorte faça
13:        filho2.genes[ ].adcionar(pai2.genes[i])
14:    fim para
15:    para i ← pontoCorte até tamanho(pai1.genes[ ]) faça
16:        filho2.genes[ ].adicionar(pai1.genes[i])
17:    fim para
18:    filhos[ ].adicionar(filho1, filho2)
19:    retorna filhos[ ]
20: fim função
```

Troca de genes estabelecida pelo cruzamento pode ser ilustrada pelo exemplo ilustrado na Figura 23.

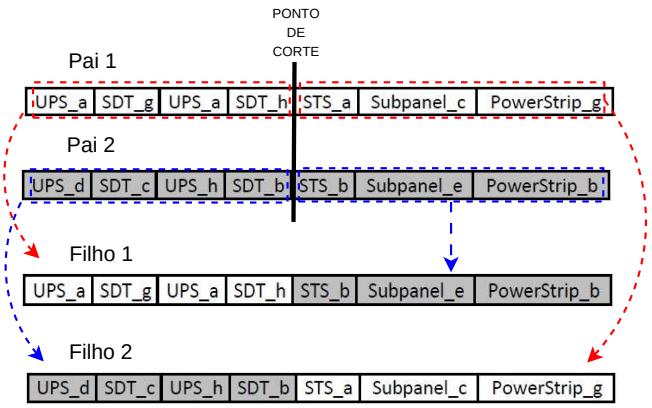

Figura 23. Exemplo de cruzamento.

Nestes dois novos cromossomos gerados existe uma pequena probabilidade de haver mutação (ex., 5%), ou seja, de um gene aleatório ser escolhido para ter seu valor trocado. O Algoritmo 7 demonstra que, caso haja a mutação, um gene do cromossomo é sorteado para que o equipamento nele contido seja substituído por outro equipamento escolhido aleatoriamente na lista gerada pelo arquivo de entrada (linhas 4 e 5). A Figura 24 expõe um exemplo do processo de mutação descrito no Algoritmo 7.

Algoritmo 7 Mutação

```
1: função MUTACAO(cromossomo)
2:     probabilidade ← sortearInteiro(100)
3:     se probabilidade < 5 então    ▷ probabilidade de 5% de ocorrer mutação
4:         i ← sortearInteiro(tamanho(cromossomo.genes[ ])-1)
5:         cromossomo.genes[i] ← sortearEquipamento(listaEquipamentos)
6:     fim se
7:     retorna cromossomo
8: fim função
```

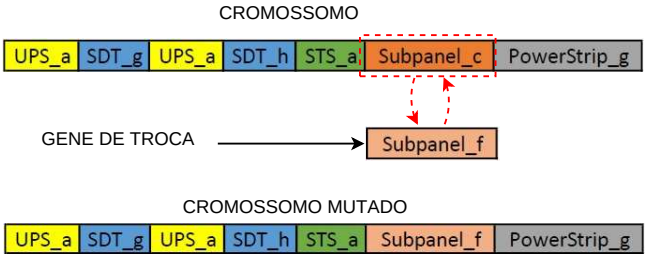

Figura 24. Exemplo de mutação.

Após o cruzamento e mutação (caso haja), os dois novos indivíduos passam pelo processo de pontuação descrito pela Figura 22 e pelo Algoritmo 4 já explicado nesta seção. Estes novos cromossomos, depois de pontuados, são inseridos à população. Por fim, após esta inserção, os dois piores indivíduos (com menor pontuação) de toda a população são descartados (Elitismo).

O algoritmo genético é repetido a partir da seleção até que se atinja o critério de parada e o algoritmo retorne uma população de soluções satisfatórias. Para finalizar, o algoritmo genético observa se a população de cromossomos apresenta soluções satisfatórias através através do grau de convergência da população. Verifica-se a média da pontuação da população em cada geração. Quando essas médias tendem a ser iguais por algumas gerações é sinal de que não haverá mais melhoras significativas na população, mesmo rodando mais gerações.

5. Estudo de Caso

Este capítulo apresenta dois estudos de caso, o primeiro foca em validar a estratégia proposta, e o segundo analisa e aplica otimização em uma infraestrutura de um *data center* típico. Em ambos os estudos de caso, todos os parâmetros de entrada dos modelos de fluxo e disponibilidade foram extraídos de um arquivo .txt que contem a lista de equipamentos gerada a partir dos componentes e seus respectivos parâmetros presentes na Tabela 1. Esses dados foram extraídos de [2] [38] [39]. Cada equipamento possui, como parâmetros de entrada, o MTTF a eficiência energética (Ef) e o custo de aquisição. Para o MTTR, em todos os componentes, foi adotado o período de 8h.

5.1 Estudo de caso I

O objetivo desse estudo de caso é comparar os resultados obtidos através da estratégia de otimização com os resultados obtidos com o algoritmo de força bruta. Este algoritmo analisa todas as combinações possíveis de equipamentos da

Tabela 1. Parâmetros de entrada.

Componente	MTTF (h)	Ef (%)	Custo (USD)
UPS_250KVA	[32787; 72996]	[85; 99]	[51215; 65513]
UPS_5KVA	[32787; 72996]	[85; 99]	[3015; 4302]
SDT	[148598; 357689]	[84; 98]	[437; 647]
Subpanel	[152642; 451949]	[84; 98]	[156; 224]
STS	[36732; 58284]	[84; 97]	[667; 879]
Power Strip	[100386; 274097]	[87; 97]	[153; 210]
Generator	[49654; 64355]	[21; 35]	[49200; 65433]

arquitetura que se deseja otimizar. Além disso, este estudo também compara o tempo de execução das duas estratégias.

5.1.1 Modelos

Cinco arquiteturas (A1, A2, A3, A4 e A5) [2] de graus de complexidade diferentes foram adotadas com o intuito de mostrar a eficácia do algoritmo genético para otimizar a disponibilidade, exergia operacional e o custo total em um curto espaço de tempo em relação ao algoritmo de força bruta. A Figura 25 mostra as cinco arquiteturas elétricas de *data center* adotadas nesse estudo.

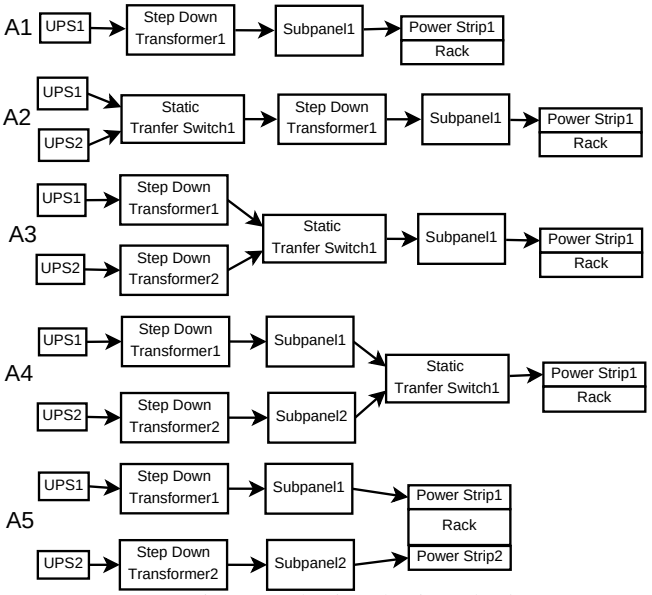

Figura 25. Arquiteturas elétricas básicas de *data center*.

A partir das arquiteutras do *data center* são criados os modelos RBD e EFM. As Figuras 26 e 27 apresentam os modelos RBD e EFM para a arquitetura A2. A execução do algoritmo genético vai gerar o cromossomo que deve ser avaliado. Os equipamentos que compõem o cromossomo vão ser utilizados no modelo RBD para se obter a disponibilidade. Em seguida, repete-se o processo para atualizar o modelo EFM com os equipamentos utilizados no cromossomo. De posse dos resultados da avaliação do RBD e EFM, o algoritmo genético vai calcular a pontuação do cromossomo com a função *fitness*.

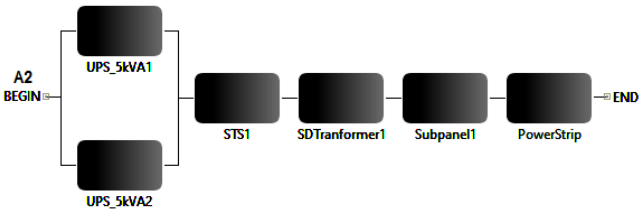

Figura 26. Modelo em RBD da arquitetura A2.

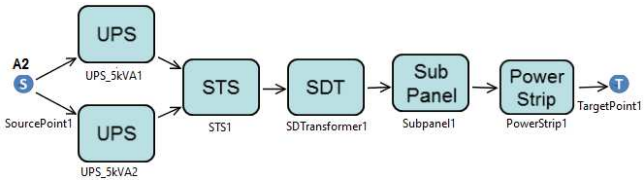

Figura 27. Modelo em EFM da arquitetura A2.

Cada cromossomo é modelado conforme as cinco arquiteturas adotadas, concatenando toda a arquitetura num vetor único como ilustra a Figura 28. Vários cromossomos são gerados até atingir uma população inicial de 30 indivíduos. Posteriormente, selecionam-se dois indivíduos para o cruzamento, com 5% de chance de haver mutação, gerando dois novos indivíduos que são inseridos na população. Em seguida toda a população é avaliada afim de se descartar os dois piores cromossomos (de menor pontuação de acordo com a função *fitness*). Cada ciclo deste, a partir da população inicial, é uma geração.

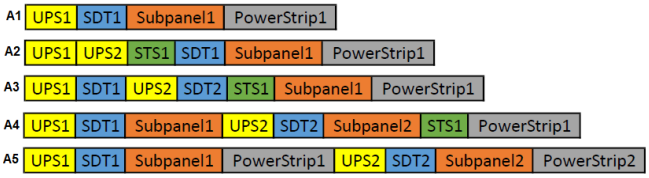

Figura 28. Cromossomos referentes a cada uma das arquiteturas básicas adotadas.

Foi observado, experimentalmente, que com um número de gerações superior a 1000, a população de cromossomos não evoluía de forma significativa, sendo este o número de gerações utilizado para a otimização das arquiteturas analisadas. Desta forma, após 1000 gerações, a população de cromossomos produz um conjunto de soluções satisfatórias, ou seja, otimizadas.

Para este trabalho, a função *fitness* atribui pesos a cada objetivo e os unem em uma função única. Essa função foi elaborada de maneira a maximizar a disponibilidade e minimizar a exergia operacional e o custo total. A Equação 7 apresenta a função *fitness* adotada.

$$Pontuacao = (Disp \times 10) - (Custo \div 1000) - Exergia \quad (7)$$

A Equação 7 determina a qualidade da solução contida no cromossomo. Quanto maior o valor da *Pontuacao*, melhor é o

cromossomo. A disponibilidade e o custo total foram ajustados para a mesma ordem de grandeza da exergia operacional afim de manter o equilíbrio entre estas variáveis.

5.1.2 Resultados

A Tabela 2 apresenta uma comparação dos resultados obtidos na otimização das cinco arquiteturas utilizando o algoritmo genético proposto com o algoritmo da força bruta. Nessa tabela, AG representa algoritmo genético, FB são os resultados obtidos pela força bruta, e DP apresenta o desvio percentual da diferença entre a força bruta e o algoritmo genético. Além disso, é apresentado o tempo de execução demandado para se obter os resultados sob cada abordagem.

Tabela 2. Otimização das Arquiteturas Adotadas.

Arq A1	C (USD)	Ex (J)	Disp (9s)	T (s)
AG	14447,30	19,84	3,5486	1,02
FB	14420,84	18,92	3,5278	5,95
DP (%)	0,1831	4,6511	0,5859	580,18
Arq A2	C (USD)	Ex (J)	Disp (9s)	T (s)
AG	18984,57	30,19	3,5752	1,16
FB	18965,65	30,77	3,6320	360,64
Desvio (%)	0,0996	1,8850	1,5644	31039,42
Arq A3	C (USD)	Ex (J)	Disp (9s)	T (s)
AG	19409,09	29,66	3,6231	1,25
FB	19302,60	27,22	3,6000	5620,21
DP (%)	0,5486	8,2088	0,6384	446084,53
Arq A4	C (USD)	Ex (J)	Disp (9s)	T (s)
AG	19613,98	30,49	3,6718	1,35
FB	19585,69	30,77	3,7394	91730,10
DP (%)	0,1441	0,9154	1,8071	6763260,63
Arq A5	C (USD)	Ex (J)	Disp (9s)	T (s)
AG	18679,81	20,55	7,0975	1,35
FB	18725,14	22,39	7,3533	62669,53
DP (%)	0,2420	8,1739	3,478483	4628701,75

Na primeira linha da tabela, pode-se observar o custo, a exergia consumida , a disponibilidade obtida em número de 9s ($(-\log(1-disp))$), e o tempo de execução. Na segunda linha, são observados os resultados da otimização através do algoritmo genético para a arquitetura A1. Na terceira linha são visualizados os resultados obtidos por força bruta para a mesma arquitetura. Na quarta linha é observado o desvio percentual obtido através da comparação entre os resultados da otimização pelo algoritmo genético e pela força bruta. A partir da quinta linha, segue-se a mesma ordem de resultados obtidos em cada uma das cinco arquiteturas adotadas neste estudo de caso.

É possível verificar que a técnica de otimização obtem resultados consideráveis em todas as arquiteturas, observando-se um aumento na disponibilidade e uma redução na exergia operacional e no custo. Pode-se verificar ainda que os resultados obtidos através da otimização com o algoritmo genético são muito próximos dos resultados obtidos por força bruta. No entanto, o tempo gasto para se executar cada uma das duas técnicas é bem distinto. A medida que a complexidade das arquiteturas aumenta, o tempo gasto para otimizar com força bruta cresce exponencialmente. Já no caso da otimização através do algoritmo genético, o tempo gasto permanece aproximadamente o mesmo e significativamente menor que o tempo gasto pela força bruta. Por exemplo, o tempo gasto para otimizar a arquitetura A5 utilizando AG é aproximadamente 45000 vezes menor que o tempo gasto pela força bruta e com resultados muito próximos.

5.2 Estudo de caso II

Este estudo de caso apresenta a otimização de infraestruturas elétricas de *data center* enquadradas na classificação TIER (especificamente, TIER I e TIER II) de acordo com a norma ANSI/TIA-942 [5].

5.2.1 Modelos

O objetivo é otimizar a disponibilidade, exergia operacional e o custo em um curto espaço de tempo, conforme demonstrado no estudo de caso I. A comparação com o algoritmo de força bruta se torna inviável devido a complexidade das arquiteturas analisadas neste estudo caso em relação ao primeiro.

TIER I É a arquitetura mais básica, com redundância *N*, possuindo um caminho único de distribuição (ver figura 29). Possui um gerador como alternativa ao fornecimento de energia.

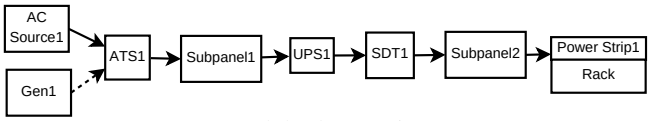

Figura 29. Modelo da Arquitetura TIER I.

Para representar a redundância do mecanismo de ativação do gerador, a Figura 30 apresenta um modelo SPN *cold stanby*. Este modelo representa a ativação do gerador mediante a falha da concessionária de energia. O tempo de ativação adotado foi de 5 min. A disponibilidade, para este modelo, é obtida através da expressão de probabilidade: P{(#CON_ON = 1) OR (#GEN1_ON = 1)}

Ao calcular a disponibilidade do modelo da Figura 30, este valor é utilizado no bloco CON_GEN do modelo RBD ilustrado na Figura 31. A avaliação desse modelo fornece a disponiblidade de sistema TIER I.

Obtida a disponibilidade do modelo RBD anterior, este resultado é utilizado no modelo EFM (ver Figura 32) para o cálculo da exergia operacional e do custo. Estas métricas juntamente com a disponibilidade são utilizadas pelo algoritmo genético proposto para o cálculo da função *fitness* gerando a pontuação dos cromossomos para que a otimização seja realizada.

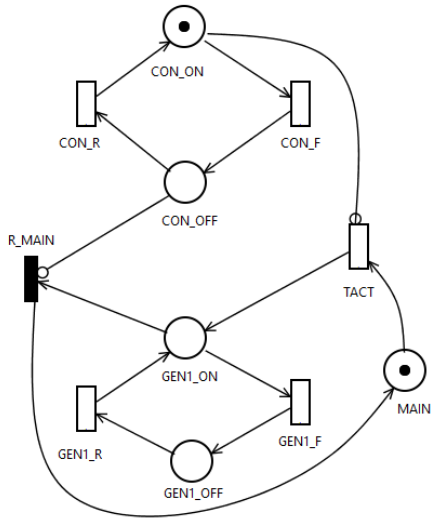

Figura 30. Modelo SPN referente à redundância concessionária/gerador da Arquitetura TIER I.

Figura 31. Modelo RBD referente à Arquitetura TIER I.

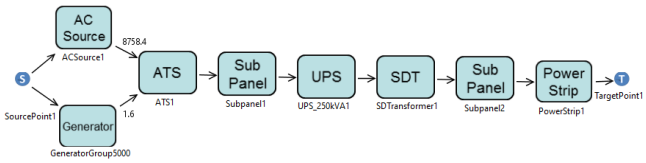

Figura 32. Modelo EFM referente à Arquitetura TIER I.

TIER II A Figura 33 representa o modelo da arquitetura TIER II, segundo nível de complexidade da classificação TIER, possuindo uma redundância $N+1$ nos componentes de fornecimento elétrico, e com um caminho único de distribuição.

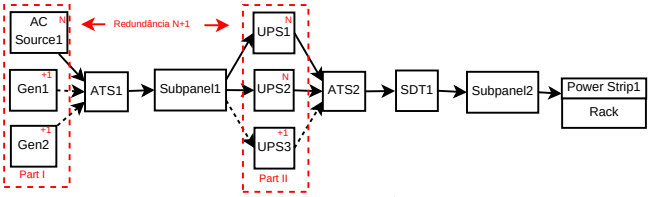

Figura 33. Modelo da Arquitetura TIER II.

Dois modelos SPN foram criados para avaliar a disponibilidade em cada redundância (parte em destaque na Figura 33). Em ambos modelos, se o *token* estiver em lugares com denominação terminada em "_ON", significa que o componente está operante. Caso contrário, ou seja, estando em lugares com denominação terminada em "_OFF", significa que o componente está inoperante. Caso as transições com denominação terminada em "_F" forem disparadas, isso representa a falha do equipamento. Entretanto se as transições com denominação terminada em "_R" forem disparadas, isso significa que o equipamento foi reabilitado.

Para representar a redundância concessionária/ gerador1/ gerador2 (*Part I* da Figura 33), a Figura 34 apresenta um modelo SPN de maior complexidade baseado no modelo *cold stanby*. Este modelo ilustra a disposição da concessionaria de energia, com a alternativa de dois geradores caso haja interrupção no fornecimento de energia elétrica. Caso a concessionária falhe, o gerador1 é acionado. Caso este falhe, o gerador2 é acionado. Esse subsistema (*Part I*) é considerado em funcionamento se ao menos a concessionária ou um dos geradores estiver em funcionamento. Para o cálculo da disponibilidade deste modelo, a seguinte expressão de probabilidade é utilizada: P{(#CON_ON = 1) OR (#GEN1_ON = 1) OR (#GEN2_ON = 1)}.

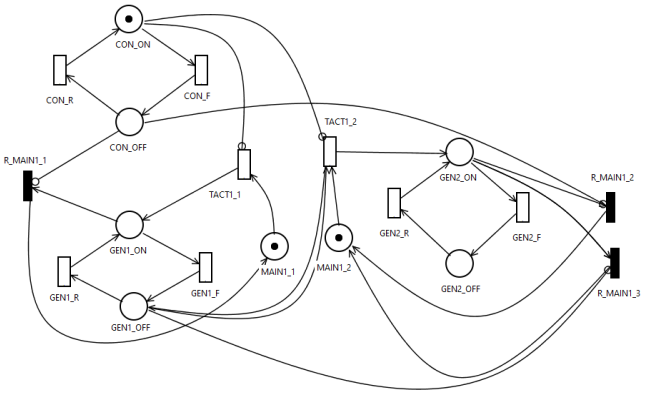

Figura 34. Modelo SPN referente à redundância concessionária/gerador1/gerador2.

O lugar CON_ON inicia com um *token*, e a transição CON_F fica habilitada, definindo que a concessionária está operante. Ao disparar, o *token* segue para o lugar CON_OFF definindo a falha da concessionária. As transições TACT1_1 e CON_R ficam habilitadas, e com o disparo da primeira transição o módulo reserva (gerador1) é ativado removendo um *token* do lugar MAIN1_1 e o colocando em GEN1_ON. Para que o gerador1 falhe, a transição GEN1_F é disparada transferindo o *token* para GEN1_OFF (falha do gerador1), habilitando a transição TACT1_2. Com o diparo desta transição, o *token*, que estava em MAIN1_2, é transferido para GEN2_ON, acionando o gerador2.

Para que o módulo principal (concessionária) esteja desativado, um *token* está no lugar CON_OFF habilitando a transição CON_R para reparar o módulo principal. Disparada esta transição, o *token* é deslocado de CON_OFF para CON_ON reativando o módulo principal. Caso um dos módulos reserva esteja ativado (um *token* nos lugares GEN1_ON e/ou GEN2_ON), as transições instantâneas R_MAIN1_1, R_MAIN1_2 e R_MAIN1_3 são automaticamente disparadas removendo o *token* de volta para um dos lugares MAIN1_1, MAIN1_2 e MAIN1_3, desativando qualquer um dos geradores.

A Figura 35 apresenta o modelo SPN proposto para representar o subsistema Part II que consiste em uma redundância

N +1. Este modelo ilustra a disposição de dois UPSs funcionando simultaneamente, e, caso um falhe, o UPS reserva é acionado automaticamente. Esse subsistema (*Part II*) é considerado em funcionamento se ao menos dois UPSs estiverem em funcionamento. A disponibilidade é calculada pela seguinte expressão de probabilidade: P{((#UPS1_ON = 1) AND (#UPS3_ON = 1)) OR ((#UPS2_ON = 1) AND (#UPS3_ON = 1)) OR ((#UPS1_ON = 1) AND (#UPS2_ON = 1))}.

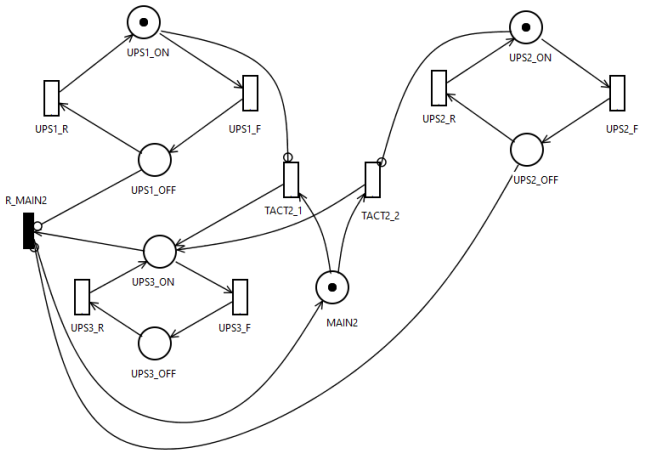

Figura 35. Modelo SPN referente à redundância dos UPS1/UPS2/UPS3.

Os lugares UPS1_ON e UPS2_ON representam o estado de funcionamento dos módulos principais (UPS). Para que qualquer um destes módulos falhe, as transições UPS1_F ou UPS2_F tem que ser disparadas. Uma vez disparada a transição UPS1_F, um *token* é removido do lugar UPS1_ON e colocado no UPS1_OFF. Na ausência de *tokens* em UPS1_ON, a transição TACT2_1 é habilitada. Com o disparo desta transição, o módulo reserva (UPS3) é ativado removendo um *token* do lugar MAIN2 e adicionando em UPS3_ON. Assumindo o disparo da transição UPS2_F, um *token* é removido do UPS2_ON e colocado no UPS2_OFF. Na ausência de *tokens* em UPS2_ON, a transição TACT2_2 é habilitada.

Para que o módulo principal (UPS1) esteja desativado, um *token* está no lugar UPS1_OFF habilitando a transição UPS1_R para reparar o módulo principal. Disparada esta transição, o *token* é deslocado de UPS1_OFF para UPS1_ON. Observando que o módulo reserva está ativado com um *token* em UPS3_ON, a transição instantânea R_MAIN2 é automaticamente disparada, removendo o *token* de UPS3_ON para MAIN2, desativando o módulo reserva (UPS3). Comportamento similar ocorre se o módulo desativado for o UPS2.

Obtidas as disponibilidades dos modelos SPN, o valor referente a redundância concessionária/gerador1/gerador2 (*Part I*) é atribuído ao bloco CON_GEN, e a disponibilidade referente a redundância UPS1/UPS2/UPS3 (*Part II*) é atribuída ao bloco GroupUPS, ambos do modelo RBD representado na Figura 36. Com a avaliação desse RBD, obtém-se a disponibilidade do modelo completo. A Figura 37 apresenta o correspondente modelo EFM para representar o comportamento do fluxo de energia nessa arquiteura TIER II.

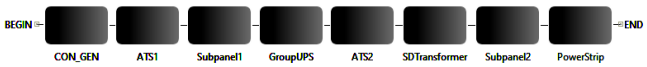

Figura 36. Modelo RBD referente à Arquitetura TIER II.

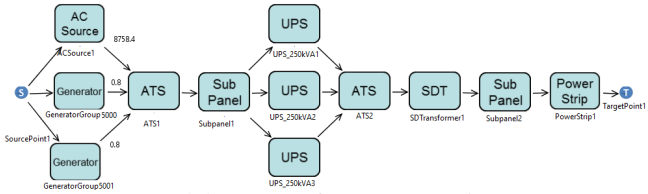

Figura 37. Modelo EFM referente à Arquitetura TIER II.

Aplicação do Algoritmo Genético

É gerada uma população inicial com 100 indivíduos. Feito isso, o algoritmo genético prossegue suas etapas. Dois indivíduos são escolhidos pelo processo de seleção para o cruzamento com 5% de chance de haver mutação. Os dois cromossomos escolhidos geram dois novos cromossomos que são inseridos à população. Toda a população é avaliada com o objetivo de se descartar os dois piores cromossomos (de menor pontuação). A função *Fitness* utilizada foi a mesma empregada no estudo de caso I e descrita na Equação 7. O processo se repete, a partir da seleção até que a população de cromossomos apresente as soluções. Cada repetição do processo é uma geração.

Foi observado que a população de cromossomos não evoluía com um número de gerações maior que 5000, chegando a convergência. Sendo assim, este número de gerações foi adotado para a otimização das arquiteturas analisadas. Desta forma, após as 5000 gerações, a população de cromossomos obtém um conjunto de soluções otimizadas.

5.2.2 Resultados

A Tabela 3 apresenta uma comparação entre os dados da população inicial gerada e os resultados obtidos na otimização das duas arquiteturas TIER pelo algoritmo genético proposto.

Tabela 3. Otimização das Arquiteturas TIER Adotadas.

TIER I	C (USD)	Ex (J)	Disp (9s)	T (s)
P Inicial	122322,20	203,39	3,3643	-
Otimização	116004,52	140,13	3,4005	463,34
M (%)	5,16	31,10	1,06	-
TIER II	**C (USD)**	**Ex (J)**	**Disp (9s)**	**T (s)**
P. Inicial	292795,44	283,82	3,3478	-
Otimização	269377,66	175,30	3,4158	905,07
M (%)	7,99	38,23	1,99	-

É possível verificar que a otimização pelo algoritmo genético, obtem resultados consideráveis em todas as arquiteturas, observando-se um aumento na disponibilidade e uma redução

na exergia operacional e no custo. Apesar de uma complexidade bem maior que a do estudo de caso I, o tempo de execução demando pelo algoritmo genético (aproximadamente 15 min) é aceitável levando em consideração a quantidade de variáveis do problema.

Pode-se observar que quanto mais complexa a arquitetura, maior a sua disponibilidade, exergia e custo. Vale ressaltar que o objetivo da classificação TIER é classificar os *data centers* de acordo com a redundância da infraestrutura elétrica. Cada arquitetura apresentada neste estudo, deve ser adotada de acordo com a demanda de disponibilidade dos serviços a serem oferecidos no *data center*, ou seja, o quanto eles precisam ser tolerantes a desligamentos por falha ou para manutenção.

6. Conclusão

Este trabalho propôs um algoritmo genético integrado a modelos formais de avaliação de disponibilidade, sustentabilidade e custo, suportados pelo Mercury, de maneira a otimizar as métricas de disponibilidade, exergia e custo de arquiteturas elétricas de *data center*. Esse algoritmo proposto se mostrou eficiente ao retornar um grupo de soluções próximas da melhor solução encontrada pelo algoritmo de força bruta e em um tempo muito inferior. Essa diferença de tempo se torna cada vez mais acentuada a medida que se aumenta a complexidade da arquitetura a ser otimizada. Além disso, foi observada a facilidade da integração do algoritmo genético proposto com os modelos RBD, SPN e EFM através do ambiente Mercury com sua linguagem de *script*. Aliado a isso, a metodologia adotada propõe o passo-a-passo da otimização de arquiteturas em quaisquer níveis de complexidade.

Como trabalho futuro, iremos aplicar a estratégia de otimização nas três infraestruturas de *data center*, infraestrutura de TI, infraestrutura de refrigeração e a infraestrutura elétrica, em diversos graus de complexidade computacional.

Contribuição dos autores

- Márcio Austregésilo: Principal autor e escritor do artigo.
- Gustavo Callou: Orientador e revisor do artigo.

Referências

[1] CALLOU, G. et al. Estimating sustainability impact of high dependable data centers: A comparative study between brazilian and us energy mixes. *Computing*, Springer, v. 95, n. 12, p. 1137–1170, 2013.

[2] CALLOU, G. et al. An integrated modeling approach to evaluate and optimize data center sustainability, dependability and cost. *Energies*, Multidisciplinary Digital Publishing Institute, v. 7, n. 1, p. 238–277, 2014.

[3] FIGUEIREDO, J. et al. Estimating reliability importance and total cost of acquisition for data center power infrastructures. In: *Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on*. [S.l.: s.n.], 2011. p. 421–426.

[4] SILVA, B. et al. An integrated environment for performance and dependability evaluation of general systems. In: *45th Dependable Systems and Networks Conference*. [S.l.: s.n.], 2015. v, p. 1–6.

[5] ASSOCIATION, T. I. *Telecommunications Infrastructure Standard for Data centers ANSI/TIA-942*. [S.l.]: Telecommunications Industry Association, 2005.

[6] ROSA, R.; ARANDA, M.; ANTONIOLLI, P. Segurança física em datacenters: estudo de caso. *Refas-Revista Fatec Zona Sul*, v. 3, n. 4, p. 1–22, 2017.

[7] AVIZIENIS, A.; LAPRIE, J.; RANDELL, B. Fundamental Concepts of Dependability. *Technical Report Series-University of Newcastle upon Tyne Computing Science*, 2001.

[8] MEYER, J. F.; SANDERS, W. H. Specification and construction of performability models. In: *Proceedings of the Second International Workshop on Performability Modeling of Computer and Communication Systems*. [S.l.: s.n.], 1993. p. 28–30.

[9] AVIZIENIS, A. et al. Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, v. 1, n. 1, p. 11 – 33, jan.-march 2004.

[10] PRADHAN, D. K. *Fault-tolerant computer system design*. [S.l.]: Prentice-Hall, 1996.

[11] MACIEL, P. et al. Performance and dependability in service computing: Concepts, techniques and research directions. In: ____. [S.l.]: Igi Global, 2011. (Premier Reference Source), cap. Dependability Modeling.

[12] SOUSA, E. *Modelagem de desempenho, dependabilidade e custo para o planejamento de infraestruturas de nuvens privadas*. Tese (Doutorado em Ciência da Computação) — Centro de Informática, Universidade Federal de Pernambuco, 2015.

[13] ANDRADE, E. et al. Availability modeling and analysisof a disaster-recovery-as-a-service solution. *Computing*, Springer, V, p. 1–26, 2017.

[14] KOTAS, T. J. The exergy method of thermal plant analysis. Butterworth Publishers, Stoneham, MA, 1985.

[15] SZARGUT, J.; MORRIS, D.; STEWARD, F. Energy analysis of thermal, chemical, and metallurgical processes. Hemisphere Publishing, New York, NY, 1988.

[16] ALVES, G. et al. Supply chain management - applications and simulations. In: ____. [S.l.]: InTech, 2011. cap. 8 - Business and Environment Performance Evaluation in Supply Chains: a Formal Model-Driven Approach, p. 157–182.

[17] FERREIRA, J. et al. An algorithm to optimize electrical flows. In: IEEE COMPUTER SOCIETY. *Proceedings of the 2013 IEEE International Conference on Systems, Man, and Cybernetics*. [S.l.], 2013. p. 109–114.

[18] MURATA, T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, IEEE, v. 77, n. 4, p. 541–580, 1989.

[19] MARSAN, M. et al. Modelling with Generalized Stochastic Petri Nets. *ACM SIGMETRICS Performance Evaluation Review*, ACM Press New York, NY, USA, v. 26, n. 2, 1998.

[20] TRIVEDI, K. *Probability and Statistics with Reliability, Queueing, and Computer Science Applications*. 2. ed. [S.l.]: Wiley Interscience Publication, 2002.

[21] ARAúJO, C. *Avaliação e modelagem de desempenho para planejamento de capacidade do sistema de transferência eletrônica de fundos utilizando tráfego em rajada*. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal de Pernambuco, Centro de Informática, 2009.

[22] SILVA, B. *A framework for availability, performance and survivability evaluation of disaster tolerant cloud computing systems*. Tese (Doutorado) — Centro de Informática, Universidade Federal de Pernambuco, Recife, 2016.

[23] MARSAN, M. A. et al. Modelling with generalized stochastic petri nets. *ACM SIGMETRICS Performance Evaluation Review*, ACM, v. 26, n. 2, p. 2, 1998.

[24] BALBO, G. Introduction to stochastic petri nets. *Lectures on Formal Methods and Performance Analysis: First EEF/Euro Summer School on Trends in Computer Science, Berg en Dal, The Netherlands, July 3-7, 2000: Revised Lectures*, Springer, 2001.

[25] ARAúJO, C. et al. Availability evaluation of digital library cloud services. In: *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*. [S.l.: s.n.], 2014. p. 666–671.

[26] EBELING, C. *An Introduction to Reliability and Maintainability Engineering*. [S.l.]: Waveland Press, 1997.

[27] CALLOU, G. *Planning of Sustainable Data Centers with High Availability: An Integrated Modeling Approach to Evaluate and Optimize Sustainability, Dependability and Cost of Data Center Systems*. [S.l.]: Lambert Academic Publishing, 2014.

[28] GOLDBERG, D. E. *Genetic algorithms in search, optimization, and machine learning*. [S.l.]: Reading: Addison-Wesley, 1989.

[29] VASCONCELLOS, D.; ABRIL, I.; MARTÍNEZ, V. Compensación de potencia reactiva en sistemas desbalanceados utilizando algoritmos genéticos. *Ingeniare. Revista chilena de ingeniería*, SciELO Chile, v. 20, n. 3, p. 284–292, 2012.

[30] LUCAS, D. *Algoritmos Genéticos: uma Introdução*. 2002. Universidade Federal do Rio Grande do Sul. Apostila elaborada sob a orientação de Luís Otávio Álvares, para a disciplina de Ferramentas de Inteligência Artificial.

[31] PAPPA, G. *Seleção de atributos utilizando Algoritmos Genéticos multiobjetivos*. Dissertação (Mestrado) — Programa de Pós Graduação em Informática Aplicada da Pontifícia Universidade Católica do Paraná. Curitiba, 2002.

[32] HINTERDING, R. Representation, mutation and crossover issues in evolutionary computation. In: IEEE. *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*. [S.l.], 2000. v. 2, p. 916–923.

[33] KAR, I.; PARIDA, R. N. R.; DAS, H. Energy aware scheduling using genetic algorithm in cloud data centers. In: *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*. [S.l.: s.n.], 2016. p. 3545–3550.

[34] LUO, L. et al. A resource optimization algorithm of cloud data center based on correlated model of reliability, performance and energy. In: *2016 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. [S.l.: s.n.], 2016. p. 416–417.

[35] BOSSE, S. et al. Introducing greenhouse emissions in cost optimization of fault-tolerant data center design. In: *2016 IEEE 18th Conference on Business Informatics (CBI)*. [S.l.: s.n.], 2016. v. 01, p. 163–172.

[36] ZIAFAT, H.; BABAMIR, S. M. Towards optimization of availability and cost in selection of geo-distributed clouds datacenter. In: *2016 IEEE 10th International Conference on Application of Information and Communication Technologies (AICT)*. [S.l.: s.n.], 2016. p. 1–5.

[37] OLIVEIRA, D. M. et al. Advanced stochastic petri net modeling with the mercury scripting language. In: *11th EAI International Conference on Performance Evaluation Methodologies and Tools*. [S.l.: s.n.], 2017.

[38] MARWAH, M. et al. Quantifying the sustainability impact of data center availability. *SIGMETRICS Performance Evaluation Review*, v. 37, p. 64–68, 2010.

[39] IEEE Gold Book 493, Design of Reliable Industrial and Commercial Power Systems. [S.l.]: IEEE, 2007.